
# Fight crime.
# Unravel incidents... one byte at a time.

# GIAC Certified Forensic Analyst (GCFA) Practical Assignment

Version 1.2

or

A Proposal for a Binary Comparison Technique



Gerardo Lamastra

25th March 2003

**Abstract**

This document presents the results of three different activities performed in the context of the GCFA certification program. In the first part, we present the analysis of an unknown binary; the goal of the analysis is the identification of the program; we also discuss a possible approach for binary program comparison and show preliminary results of this technique. The second part of the practical discusses the forensic analysis performed on the disk of a dead Linux machine. The machine has been compromised, and we have to identify what happened. The third part illustrates some legal issues in the context of computer forensic investigation.

# Contents

# Part I

# Part 1 - Analysis of an Unknown Binary

## 1.1 Basic information

This section of the document illustrates the strategy adopted for the analysis of an unknown executable. The ability required for this purpose is strongly related with Reverse Engineering. Traditionally, Reverse Engineering is a set of techniques aimed at reconstructing the source code from which a given program has been compiled. The ultimate goal of the Reverse Engineer is the knowledge of the internal program structure. Reverse Engineering has been actively used to dissect protection mechanisms implanted into commercial software; however, it also provides an extremely valid approach to determine the nature and the characteristics of a software, which has been illegally installed on our system. The analysis is composed by the following different stages:

- File Analysis, devoted to classify the file and gather few basic structured information from it.
- String analysis, for a first identification of the unknown binary based on intelligible text sequence embedded in it.
- Internet Search, to see if we can find the source code on Internet.
- Dynamic Analysis, which consists in running the binary in a sandbox environment, to study its behavior
- Complete Binary Identification, through the compilation of the program from its source (if it can be found somewhere), or complete source code reconstruction.

The binary under analysis has been downloaded from the web-site of SANS [18] as a zipped archive. First of all, we analyze the "package" that wraps our mysterious guest. ZIP files contains some useful info, which can be dug out using `unzip` and `zipinfo`.

```
# zipinfo -2lv binary_v1.1.zip
```

This command, with the long and verbose output, reveals a lot of interesting information about the origin of the file. Using the more compact form:

```
# zipinfo -2l binary_v1.1.zip
```

We have:

```
Archive:  binary_v1.1.zip   7309 bytes   2 files
-rw-rw-rw-  2.0 fat       39 t-      38 defN 22-Aug-02 14:58 atd.md5
-rw-rw-rw-  2.0 fat    15348 b-    7077 defN 22-Aug-02 14:57 atd
2 files, 15387 bytes uncompressed, 7115 bytes compressed:  53.8
```

This clearly identifies two files: atd.md5 is a text file, while atd is a binary file. Both were compressed on a FAT or NTFS file-system. We also have the Modified Access Time, "14.57 22-Aug-02", which gives an idea of when the file was copied on the system where it was zipped.

```
# unzip -Xa binary_v1.1.zip
```

This command, executed as root, could recover the original UIDGID, if they were available. Of course, since the file seems to have been zipped under FAT or NTFS, such data is not available and unzip will assign root.root as user and group. Since the file comes from the Windows World, we also use the -a switch to convert the text file into UNIX format.

The second step is to check the MD5 signature, so to guarantee that nothing has been modified and we are effectively analyzing the file we were intended to do.

```
# md5sum -c atd.md5
atd:  OK
```

So, what is the purpose of the original atd program (not this evil one)? Quoting from the man page: *"atd runs jobs queued by* **at(1)***"*. In other words, atd is a program which is used to schedule program execution, usually only for a single time. By default, atd is owned by root, group root (this is usually indicated as root.root, following the `chown` syntax). This suggests that this program was also installed as root.root on the victim machine. However, we are going to provide stronger proofs for this admission, as we proceed with the analysis. Concluding this section, it is interesting to notice that the attacker used, as decoy program, something whose size is pretty close to the size of the evil software:

```
# ls -l atd /usr/bin/atd
-rw-r--r--    1 root     root      15348 Aug 22  2002 atd
-rwxr-xr-x    1 root     root      14384 Mar 28  2002 /usr/sbin/atd*
```

Although the real atd size was obtained from my Mandrake 9.0 Installation, this data can be considered quite general.

After the preliminary analysis, it is time to delve deep into the mysteries of this case, by analyzing the file itself.

## 1.2 File Analysis

We begin to analyze the binary with `file`. This tool performs three different checks on the file in order to identify it: file-system check, "magic number" check and text-file check. The man page [12] extensively documents the way file does its job. In this particular case, it is easy for file to identify a binary executable. More precisely:

```
# file ./atd

./atd: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
   dynamically linked (uses shared libs), stripped
```

It is possible to obtain other useful info by using some specific ELF tools[14]; `objdump` use the interface provided by the BFD [15] library to analyze the internals of the binary. Objdump can provide a lot of interesting info, including a list of the dynamic relocation entries and also disassemble the file. The `-f` switch is used to gather information about the ELF header.

```
# objdump -f atd
atd:     file format elf32-i386
   architecture: i386, flags 0x00000112:
   EXEC_P, HAS_SYMS, D_PAGED
   start address 0x08048db0
```

Collecting data from `file` and `objdump`, we can be pretty sure that this is an ELF dynamic executable for INTEL 386. If the command `objdump -R atd` is executed, it is possible to obtain a list of the functions which this binary imports from dynamic libraries. This list is very useful in order to

begin to have an idea about the program purpose. The list is reported here, using a functional grouping criteria:

- **Libc Global Variables & Internal Functions**:

```
_IO_stderr_, optarg,  __fpu_control, _errno,
 __strtol_internal, __libc_init, __setfpuc, environ
```

  These can be interesting to identify what Libc version this program has been linked with.

- **Standard Libc Functions**:

```
strcpy, strncmp, bcopy, bzero, sprintf, difftime, atexit,
strdup, getopt, exit, popen, usleep, perror, time, longjmp
```

  I believe that the `popen` function is noteworthy here, since popen is usually used to execute a command by forking a shell. See the man page [23] for further details.

- **File Related Operations**:

```
open, close, read, fgets, fprintf, ioctl, umask, chdir
```

  This indicates that the binary may perform file I/O activity.

- **Process Management Functions**:

```
kill, fork, signal, wait, alarm, setsid,
geteuid, getppidm getpid, getuid
```

  The presence of `fork` suggests that the binary is a multi-process program. Also, it may require some special privileges to be run, because of `getuid` and `geteuid` system calls.

- **System V Shared Memory Functions**:

```
shmat, shmctl, shmdt, shmget, semget, semctl, semop
```

  These also suggest some form of multi-process behavior. Moreover, we do not see these functions very often, today. These function are used to manipulate System V Shared Memory and Semaphores.

- **Socket Functions**:

```
getprotobynumber, socket, inet_addr, sendto,
inet_ntoa, gethostbyname,setsockopt
```

  This indicates that the binary may use the network.

Summarizing these initial results, we have a binary program which may run shell commands, fork and generate children processes, do some file and network operations. Is any bell ringing? There are all the signs of some kind of backdoor. However, it is too often to jump to a conclusion. More investigation is needed before we can truly confirm this hypothesis.

For the sake of completeness, it is important to note that `readelf`, another ELF binary analysis can provide substantially the same information provided by objdump.

## 1.3 Strings Analysis

String sequence analysis is one of the richest sources of information when it comes to binary analysis. Strings [16] is a program that extracts printable characters in binary file. Well, it can be used on any file, but, of course, it gives its best on binaries of any kind (not only executables, try it on your favorite .DOC file).

When strings is run on the binary, we immediately find a lot of valuable information. An ELF [24] file is usually organized into sections; there is a .text section, which contains the executable machine code, a set of sections (.plt, .got, .dynsym and .dynstr) used for dynamic linking with shared libraries, and .data .bss and .rodata sections, which contains all the data which has been encoded into the binary. Among these sections, it is the .rodata which usually has the most interesting stuff.

The section layout of an ELF program is reflected into the output of strings. First of all, we have strings representing the function names which will be looked up into dynamic libraries by the dynamic linked (ld.so). We already derived the function names using objdump or readelf, so we do not discuss them anymore. The first two rows of the string output are the dynamic libraries required to run this particular program. They are `/lib/ld-linux.so.1` and `libc.so.5`. The first is the dynamic linker itself, while the second is the C Standard Library, version 5. This gives us an interesting clue: this program has been compiled on some older Linux version, since all modern Linux distributions have adopted glibc 2.x, also known as libc.so.6; libc.so.5 is only installed if compatibility with older binaries is needed .

If everything is installed as required, the dynamic linker can provide this information too:

```
# ldd ./atd
      libc.so.5 => /usr/i486-linux-libc5/lib/libc.so.5 (0x40015000)
```

Back to the string analysis now. As I told before, the string output follows the section layout in the ELF and the most interesting stuff lies in the .rodata section (we can verify this executing objdump again:

```
# objdump -s -j .rodata atd.
```

There is something that immediately attracts attention:

```
LOKI2   route [(c) 1997 guild corporation worldwide]
```

So, we have (possibly) the name of the program, *LOKI2*, and the author, *route*, who is a well known figure in the hacker underworld. Other strings tell us about error messages and status information. The following strings:

```
remote interface:   %s
active transport:   %s
active cryptography:    %s
server uptime:      %.02f minutes
client ID:      %d
packets written:    %ld
```

suggest that this is the server component of some kind of a network communication program and also that the program could use cryptography to prevent an intermediate user to analyze the data.
We also have strings detailing some usage information:

```
lokid -p (i|u) [ -v (0|1) ]
/quit all
/quit
/stat
/swapt
```

Another thing which raises my attention is the following error message:

```
Cannot set IP_HDRINCL socket option
```

This suggests that our binary is setting socket options (this is also corroborated by the presence of `setsockopt` into the dynamic relocation symbols. If we look at the man page for the IP protocol, [20], we can find that:

"IP_HDRINCL: *If enabled the user supplies an IP header in front of the user data. Only valid for SOCK_RAW sockets. See raw(7) for more information. When this flag is enabled the values set by IP_OPTIONS, IP_TTL and IP_TOS are ignored*".

So, our program may use raw socket. But to use raw socket it must have root privileges on the system where it runs. As I said in section 1.1, this binary should have been installed as root on the victim machine, or at least executed by someone that has root privileges. I am going to give a complete proof for this statement in the next section.

Of course, all these strings can just be there to deceive our analysis. It is very easy to inject strings in a binary, for example by adding another section containing them. Injectso [9] is a tool that can do something like this, but other exist. In this case, we know that the extracted strings are where they are expected to be, in the .rodata section (This can be verified using objdump, as we did in 1.3). However, if the source code is available, the attacker can easily manipulate these information and modify the strings.

In conclusion, the information gathered until now can be trusted only with some confidence. We need to analyze the program behavior, if we want to be sure that we are really facing a standard LOKI2 server.

## 1.4 Searching the Net

Before running a dynamic analysis, I collect some information on the web. Just trivially search for "Loki2" to get interesting results. LOKI2 is a well known hacker tool, that has been cited in various publications [33] and web-sites. It was originally developed in 1996 as a proof of concept by route and published on Phrack Magazine in two papers [6, 7] detailing the concepts behind the program and the actual implementation. LOKI2 is a "Covert Channel" tool: it uses the familiar ICMP protocol to transport information between two Internet connected machines. ICMP_ECHO packets can carry information, although there is little need for this in normal situations. However, it is possible to exploit this fact to connect two machines which allow ICMP traffic to be sent or received.

Today, it is much more difficult to use effectively this tool, because most modern firewall installation, by default, block ICMP packets that arrive from outer networks. However, the idea to use another

protocol for transporting a different data stream is pretty general and can be applied in several different scenarios. Indeed, LOKI2 can also implement a DNS covert channel.

Searching for LOKI2 with google gives instant access to the program source code, which can be directly extracted from the paper [7]. The source code gives us a simple way to check that we are effectively facing a LOKI2 daemon: use the client program to interact with it. Also, we can compile the program and see if we can find a binary match. This may be not so easy to do, because we need to build the binary using the same build environment that was used to compile it in first place.

## 1.5 Dynamic Analysis

Dynamic Analysis is very similar to debugging. Only, we usually do not know exactly what the program is going to do. Well, in this particular situation, we have a good guess since we speculate that atd is really a LOKI2 server program.

In order to proceed with Dynamic Analysis, we need to set up a restricted and protected environment. VMWare [11] has been actively used for this purpose. VMWare provides a virtual computer which can be used to run dangerous software in a software sandbox, so to limit the amount of damage that the program can do.

Using VMWare, we can replicate a Linux environment where we can run our binary. Then, we can run the client in the virtual environment or run the client in the host operating system and use VMWare Virtual Networking to connect to the client. In order to run the program, we also need to install the legacy dynamic libraries: libc.so.5 and ld-linux.so.1 (See 1.3). Under Mandrake, this package are ld.so1-1.9.11-10mdk.rpm and libc-base-5.3.12-38mdk.rpm.

Of course, just launching the tool reveals little or nothing:

```
# ./atd

[fatal] invalid user identification value: Unknown error
```

It is possible to peek at what is going on in the kernel by using `strace` [25]. Strace is a program which has the ability to intercept and track all the syscall done by a binary program. This turns to be very useful for analyzing the behavior of a program. Another useful tool, very close to strace, is `ltrace` [8], which does a similar job, but traces also references to dynamic library functions. Let us inspect the output from strace:

```
# strace atd
execve("./atd", ["./atd"], [/* 47 vars */]) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE,
         MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40007000
mprotect(0x40000000, 21868, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
...
```

This is just the startup sequence; there is no much information here. If we continue to inspect the trace, we find the first interesting stuff:

```
...
geteuid()                               = 501
```

```
getuid()                                = 501
getgid()                                = 501
getegid()                               = 501
geteuid()                               = 501
brk(0x804c818)                          = 0x804c818
brk(0x804d000)                          = 0x804d000
...
```

OK, it has checked what are our credentials, and he has discovered that we are uid=501. The brk() syscalls are signs that something is being malloc()ed.

```
write(2, "\n[fatal] invalid user identifica"..., 58
[fatal] invalid user identification value: Unknown error
) = 58
close(0) = 0
close(0) = -1 EBADF (Bad file descriptor)
_exit(0)
```

And this is the end, for now. ltrace gives us a little more clue about what is happening:

```
...
geteuid()                                       = 501
getuid()                                        = 501
<... __libc_init resumed> )                     = 0x400a9e34
atexit(0x0804a8e0)                              = 0
geteuid()                                       = 501
perror("\n[fatal] invalid user identifica"... <unfinished ...>
...
```

We can infer that our friends has checked credentials, found that we are not root and exited out with an error message.

The next step is to run it as root. This time, the output is longer and the programs remains alive, waiting for something to happen. It can be easily identified using `ps`. If we look at the data gathered with strace, we find that after executing the startup code, the program opens two different sockets:

```
socket(PF_INET, SOCK_RAW, IPPROTO_ICMP) = 3
sigaction(SIGUSR1, {0x804a6b0, [],
  SA_INTERRUPT|SA_NOMASK|SA_ONESHOT}, {SIG_DFL}, 0x4004c358) = 0
socket(PF_INET, SOCK_RAW, IPPROTO_RAW)  = 4
setsockopt(4, SOL_IP, IP_HDRINCL, [1], 4) = 0
```

They are raw socket, so they can be used, in general, to forge arbitrary packets. The first one is set to use ICMP protocol, while the second is a generic IP packet, where the packet header are created by the TCP/IP kernel networking code.

```
getpid()                                = 1844
getpid()                                = 1844
shmget(2086, 240, IPC_CREAT|0)          = 10878988
```

```
semget(2268, 1, IPC_CREAT|0x180|0600)   = 262152
shmat(10878988, 0, 0)                   = 0x40008000
write(2, "\nLOKI2\troute [(c) 1997 guild cor"..., 52
```

After doing this operation, the program starts to execute an allocation of System V Shared Memory and Semaphore. The Key Identifier (the first parameter of the syscall), which has to be unique on a system basis, is set to the process identifier plus 242 for the first handle and 424 for the second one. This can be verified running the trace several time and taking a look at the source code. This similarity enforces our trust in the fact that this is basically a LOKI2 daemon. Then, we get the message that we already noticed during the strings analysis.

Eventually, we can observe the "classical" daemon behavior, as describe in [34]:

```
fork()                                  = 1845
  [pid  1845] --- SIGSTOP (Stopped (signal)) ---
  [pid  1845] setsid()                    = 1845
  [pid  1845] open("/dev/tty", O_RDWR)    = -1
                ENXIO (No such device or address)
  [pid  1845] chdir("/tmp")               = 0
  [pid  1845] umask(0)                    = 022
  [pid  1845] sigaction(SIGALRM, {0x8049218, [],
               SA_INTERRUPT|SA_NOMASK|SA_ONESHOT},
               {SIG_DFL}, 0x4004c358) = 0
  [pid  1845] alarm(3600)                 = 0
  [pid  1845] sigaction(SIGCHLD, {0x8049900, [],
               SA_INTERRUPT|SA_NOMASK|SA_ONESHOT},
               {SIG_DFL}, 0x4004c358) = 0
  /* The process continues with pid 1845! */
close(4)                  = 0
close(3)                  = 0
semop(262152, 0xbffff71c, 2) = 0
shmdt(0x40008000)         = 0
semop(262152, 0xbffff71c, 1) = 0
_exit(0)                  = ?
```

The program forks, detaches itself from the controlling terminal and becomes a process group leader. Standard file descriptor are closed, the active directory is moved to /tmp, and the program start a blocking read on the ICMP socket.

At this point, we can compile a client using the LOKI2 source code and see if the client and the server interact appropriately. We also set up `tcpdump`, which allow to capture a trace of the packets exchanged between client and server. To make things easier, we run client and server on the same machine, on the loopback interface. The client interacts just fine with the server; here is a transcript of the command executed with the client, which implements a shell-like interface:

```
# ./loki -d 127.0.0.1

LOKI2   route [(c) 1997 guild corporation worldwide]
loki> ls
```

```
fonts
ssh-XXsIeZuq
ssh-XXSrdhPh
loki> /quit

loki: clean exit
route [guild worldwide]
Packets read: 4
```

On the server side, we get the following output:

```
LOKI2   route [(c) 1997 guild corporation worldwide]

lokid: client <1952> freed from list
```

If we take a look at the tcpdump traffic, it is difficult to spot the difference between LOKI2 and a standard ICMP flow. The most interesting sign of a weird activity comes from the sequence number which is constant for LOKI2 (and equal to 01:f0) while it is constantly incremented for a regular ICMP flow. It is easy to observe this using a graphic packet analyzer, such as Ethereal [10]. If we stick to tcpdump, we can use the -X switch to read our packet flow and compare it with a regular PING flow. The highlighted red box identify the sequence number. The first output refers to regular pings, the second one is LOKI2 traffic.

```
20:26:58.535648 localhost.localdomain > localhost.localdomain:
  icmp: echo request (DF)
0x0000   4500 0054 0000 4000 4001 3ca7 7f00 0001  E..T..@.@.<.....
0x0010   7f00 0001 0800 e001 bc06 0100 028a 6f3e  ..............o>
...
20:26:58.535719 localhost.localdomain > localhost.localdomain:
   icmp: echo reply
0x0000   4500 0054 6d7e 0000 4001 0f29 7f00 0001  E..Tm~..@..)....
0x0010   7f00 0001 0000 e801 bc06 0100 028a 6f3e  ..............o>
...

20:26:00.783349 localhost.localdomain > localhost.localdomain:
  icmp: echo request
0x0000   4500 0054 6d77 0000 4001 0f30 7f00 0001  E..Tmw..@..0....
0x0010   7f00 0001 0800 15e9 b606 01f0 b115 790a  ..............y.
...
20:26:00.783424 localhost.localdomain > localhost.localdomain:
  icmp: echo reply
0x0000   4500 0054 6d78 0000 4001 0f2f 7f00 0001  E..Tmx..@../....
0x0010   7f00 0001 0000 1de9 b606 01f0 b115 790a  ..............y.
...
```

If we go back to our trace, we can easily identify what happened when the client packet was received:

```
<... read resumed> "E\0\0Tm\201\0\0@\1\17&\177"..., 84) = 84
fork()                                  = 1740
 [pid  1740] --- SIGSTOP (Stopped (signal)) ---
 [pid  1740] semop(360452, 0xbffff8c0, 2) = 0
 [pid  1740] time(NULL)                   = 1047497399
 [pid  1740] semop(360452, 0xbffff8c4, 1) = 0
 [pid  1740] pipe([0, 5])                 = 0
 [pid  1740] fork()                       = 1741
 [pid  1740] close(5)                     = 0
 [pid  1740] fstat(0, {st_mode=S_IFIFO|0600, st_size=0, ...}) = 0
 [pid  1740] old_mmap(NULL, 4096, PROT_READ|PROT_WRITE,
                  MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40009000
 [pid  1740] read(0,  <unfinished ...>
   [pid  1741] --- SIGSTOP (Stopped (signal)) ---
```

```
[pid  1741] close(0)                    = 0
[pid  1741] dup2(5, 1)                  = 1
[pid  1741] close(5)                    = 0
[pid  1741] execve("/bin/sh", ["sh", "-c", "ls\n"],
                  [/* 36 vars */]) = 0
            /* After, we are tracing the shell... */
```

It is easy to notice that the read operation resumes, and a new process is forked to handle the request. The request is processed by popen (popen can be noticed using `ltrace`), which ultimately forks again and executes a shell. This is again a fairly common behavior in Unix daemon process.

We conclude this section by analyzing some of the artifacts that the process may leave behind itself. LOKI2 uses System V Shared Memory for interprocess communication. The kernel provides a specific interface for this IPC mechanism. When a shared memory section, a semaphore or a message queue is created by a given process, other process can access it using that key. However, if the process is terminated and IPC objects are not cleanly released, they will remain dangling on the system. We can analyze the current set of active IPC object using the command `ipcs` and can remove objects using `ipcrm`. If atd is killed with a `killall -1 atd` or `killall -9 atd`, we can see that some IPC objects remains allocated.

```
------ Shared Memory Segments --------
key        shmid      owner     perms      bytes      nattch
0x00001db6 20709376   root      0          240        1

------ Semaphore Arrays --------
key        semid      owner     perms      nsems
0x00001e6c 131072     root      600        1

------ Message Queues --------
key        msqid      owner      perms     used-bytes   messages
```

Of course, if we convert 0x1db6 (SHM KEY) in decimal we have 7606; if we subtract 242 we get 7364. If we do the same on the SEM KEY, we have:

$$dec(0x1e6c) - 424 = 7788 - 424 = 7364$$

We have already seen that the IPC handle have been derived by summing a fixed constant to the pid of process, so 7364 was the pid of the dead process. If we find a couple of dangling IPCs handle, which satisfy the following equation:

$$KEY_{shm} - 242 = KEY_{sem} - 424$$

it is highly possible that such handles are residues of LOKI2 activity.

## 1.6 Binary Comparison and Identification

The ultimate success when dealing with an unknown binary is either completely disassembling it and reverse-engineering its source code completely, as it has been done in the Reverse Challenge [30]; or to find a perfect match between our mysterious binary and a source code we were able to find and compile. In this case, it is possible to reproduce exactly the original binary starting from the source code. As we have already seen, the binary has been compiled on a Libc5 system, so we need to do the

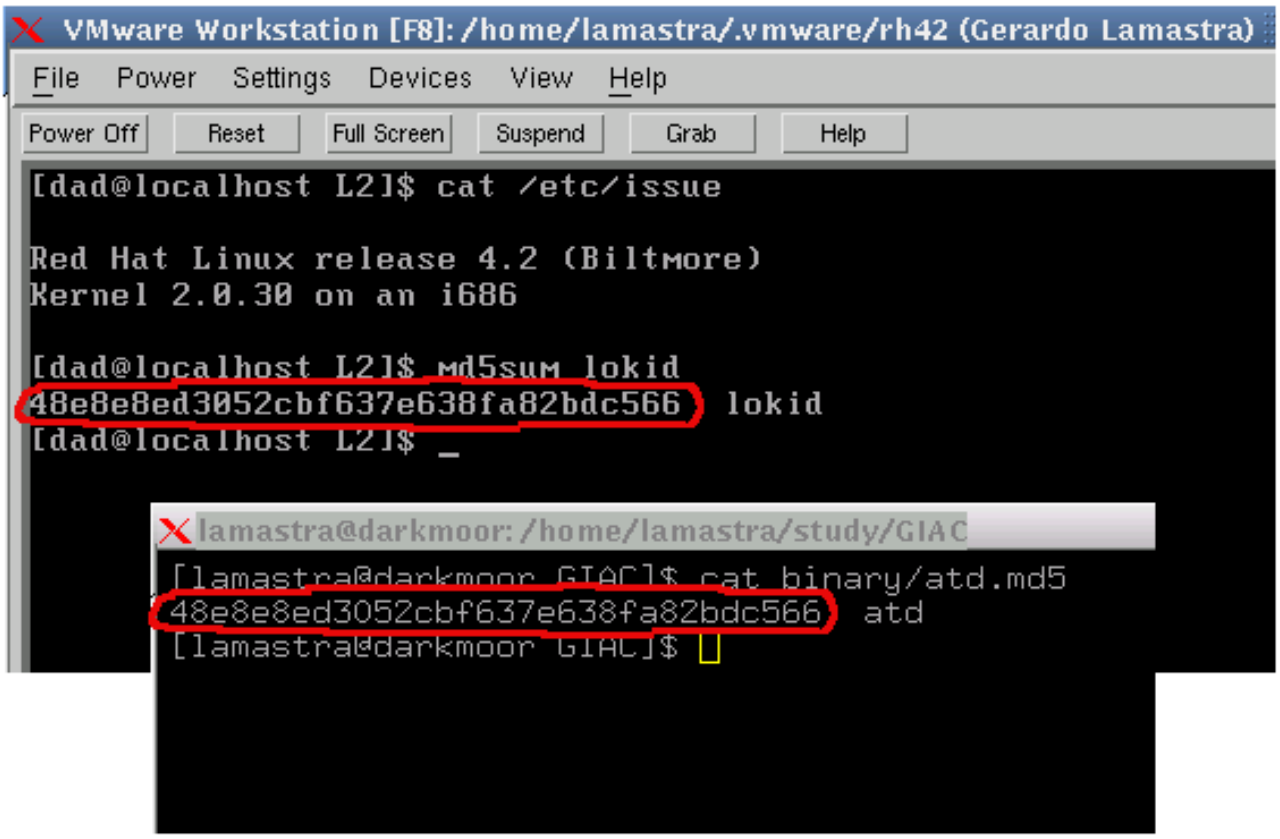


Figure 1.1: MD5 match between LOKI2 daemon and atd

complete process on such a system. In order to do so, we can use again VMWare and install a Libc5 Linux distribution, compile the source code and see if the MD5 fingerprint is identical. RedHat 4.2 is a possible choice. The figure 1.1 documents our results.

However, it is often very complex too establish such a match. If the attacker has access to source code, it is very easy for him to modify it slightly so to make almost impossible a perfect match like the one we obtained in this situation. If we analyze our major sources of information during this analysis, we can see that we heavily rely on strings information and dynamic behavior. Strings provided a very easy way to identify the code. However, it is very easy to remove strings, especially those with a significant meaning, from a given source code. This would alter the MD5 signature. Moreover, if we use a slightly different environment for building our source code, the final result is the same, yet the MD5 signature is very different. This is part of the way MD5 algorithm works.

It would be much more interesting to have a way of comparing binary program using a “fuzzier” paradigm. For example, human fingerprint analysis does not rely on perfect match algorithm. Genetic analysis too does not rely on perfect match. All these techniques are based on some sort of fuzzy algorithm, which establish, with a certain degree of accuracy, that two samples are substantially equivalent.

If we analyze a generic binary executable, we can always find at least three different section that compose it. They are usually named .text, .data and .bss; the .text section contains machine code, which is CPU specific; .data and .bss contains initialized and un-initialized data. Of course, several other section may exist (for example, dynamic linking or debugging insert other section in the code).

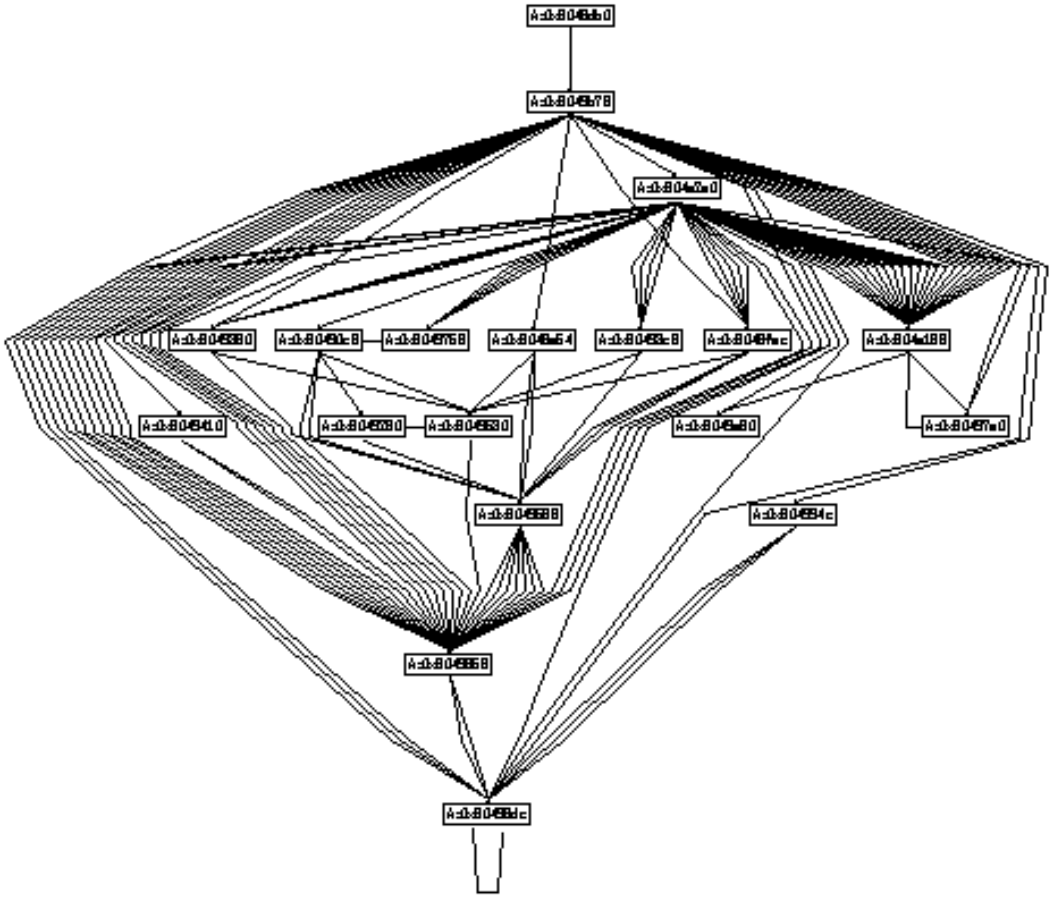

Figure 1.2: Call Graph of atd, AKA lokid (libc5 version)

Interesting enough, this model is fairly general and can be applied to many different executable format. When we compare binary programs, we are mainly interested to the code sections; this does not mean that data does not affect the analysis. However, in the context of the identification of the behavior of an unknown program, we suppose that code section can give very interesting clues.

One of the biggest problem of binary comparison is to find an appropriate representation of the executable code, which could provide the following properties:

- High degree of abstraction: several different binary programs, all derived from the same functional source code, should have similar representation.
- It should be possible to derive the abstract representation from the original binary code using an algorithm.
- It should be possible to compare different algorithm with an efficient algorithm.

The binary code can be decomposed into functions (or subroutines), and the program behavior can be represented, from a static point of view, using a graph, called “Call Graph”. The nodes of the graph are the functions, and an arc between two nodes identifies a caller/callee relationship. Dynamically linked functions can be easily considered in this model. Call Graph are intensively used as a basic data

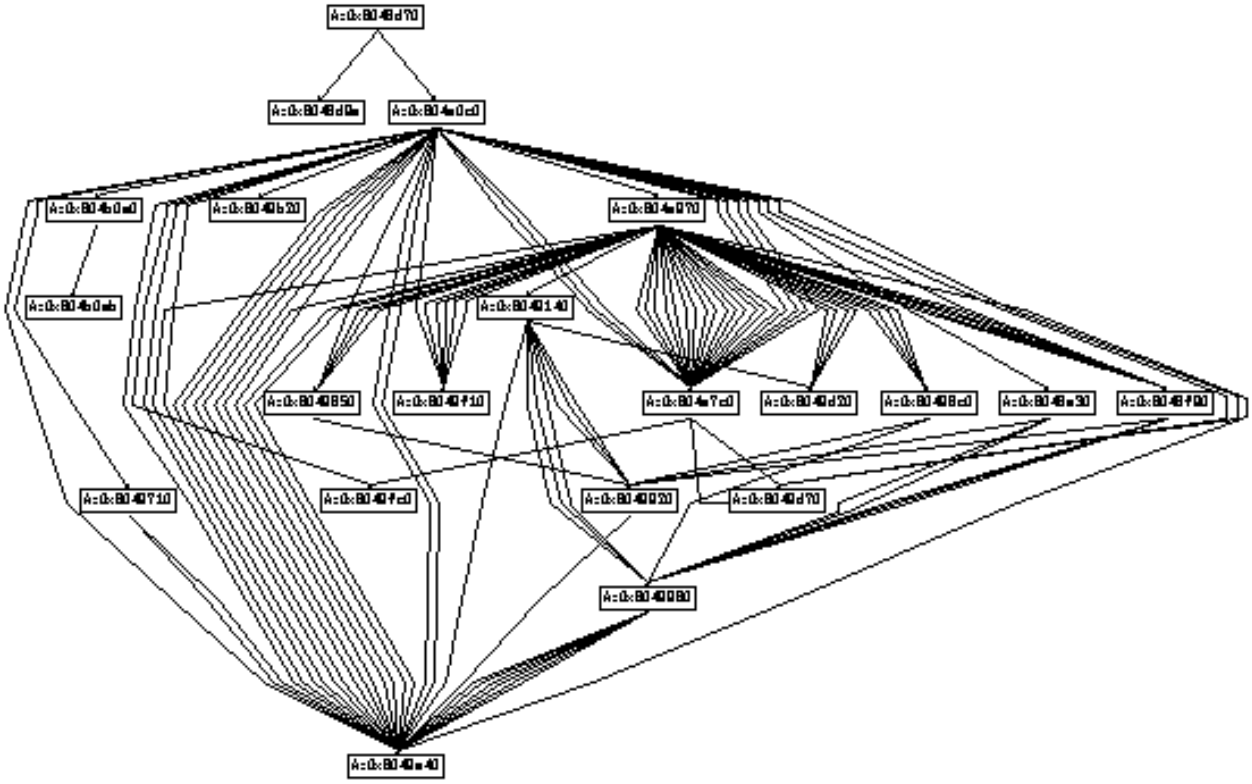

Figure 1.3: Call Graph of lokid (glibc 2.1 version)

structure for compiler design and implementation. We can use Call Graph as a very simple model to compare binary code. Of course, this model has strong limitations, however it can help the investigator to understand if two program have a certain similarity or they are completely different.

It is easy enough to identify functions in a binary code: just analyze the whole code, searching for `call` instructions and keep tracks of the called address: these are the function entry point. Usually the compiler align functions sequentially, so a very simple way for finding the return point of a given function is to look backward from the entry point of the following function, until we find a `ret` instruction. This is, again, a very simple model that can be refined, if we take into account hardware architecture.

We can take a look at the call graph of atd, lokid (compiled on a different system with a different build environment) and the original atd program (See pictures 1.2, 1.3, 1.4). There is indeed a strong similarity between the lokid that was compiled on a different system (md5 signature does not match with the atd signature) and the original atd binary, while the difference with the original atd program is pretty sharp. Again, of course, this approach has to be considered only as a simple tool for helping the investigator; more sophisticated models are required to perform an accurate analysis which can give results which can be accepted in a courtroom.

Another interesting model that can be used to perform a comparison between single functions, can be derived from computational biology. Genes can be easily represented as strings over an alphabet of four symbols. When three symbols are grouped together, we have an amino-acid. A sequence of amino-acid is essentially a protein, which is one of the basic building block of living beings. Different gene triplets can map to the same amino-acid, so different gene strings could map the same protein.

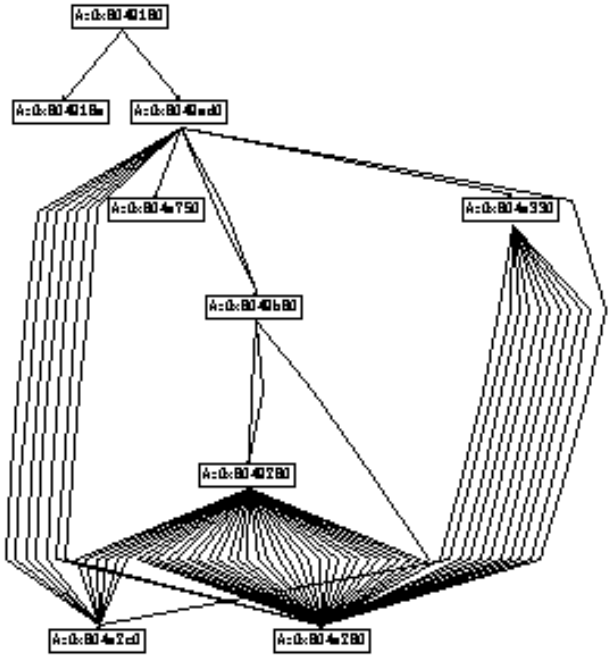

Figure 1.4: Call Graph of real atd (glibc 2.1 version)

We have three possible level of interpretations for this model:

- gene symbol
- amino-acids
- proteins

One of the most interesting problem of computational biology is the analysis of gene sequences, so to find optimal matching. Of course, it would be completely inappropriate to discuss about computational biology in this practical; there are a lot of good papers available on the Internet; a very good one is the following Ph.D. thesis [27] which contains several interesting algorithm but also a very good introduction to Computational Biology and its problems.

If we apply this model to computer program, we can imagine an interesting correspondence:

- gene symbol → machine-code
- amino-acids → intermediate compiler sequence
- proteins → functions

Although it can be very complicated to identify a match between sequences of machine code instructions and high-level language construct (mapped into intermediate compiler sequences), we can still try to use this model and reuse all the algorithms developed in the context of computational biology, especially those aimed at comparing gene symbol sequences.

If we scan the binary code and create a string where the single character represents a pseudo-code instruction, we can obtain an abstract representation of a function. It is very important to choose a

good “dictionary” in order to minimize the number of symbols. For example, all the mathematical and logical operations can be symbolically represented with an unique symbol. All the stack operation can be represented with a single symbol. Call and return are better represented with individual symbols, since they can be used to mark the beginning and the ending of a function. There is a big degree of arbitrariness when we define this mapping; for the sake of simplicity, it is possible to use the following approximation, based on the INTEL i386 architecture.

- **N**: `nop` operations, which can be easily discarded when doing comparison or sequence analysis.
- **A**: Arithmetic operations: `add`, `neg`, `sub`, `mul`, `div`, and all the other instructions used by the ALU.
- **C**: Comparison operations like `cmp`, `test` that do not modify data (only flags).
- **M**: All memory related operations (essentially `mov`). we found that it is better to use a different symbol for stack based operations
- **S**: stack operations, which include `push`, `pop`, `enter`, `leave`.
- **X**: all system like operations like `int`, `in`, `out` and all the other privileged operations.

There are several inaccuracies in these abstract mapping; for example, some kind of arithmetic can also be used to implement a comparison; however, the purpose of this discussion is essentially to outline a possible research direction.

It is easy enough, by using the `BFD` library [15], to implement a program which disassembles the .text section of an ELF binary, identifies single functions that compose the binary and outputs an abstract representation of these functions, according to the dictionary mentioned beforehand. The source code of this program (called `bdiff`) is publicly available on [21]. bdiff can also output Call-Graph using the VCG [17] language for graph display. Interesting enough, this function has been also implemented in one of the best known tools for binary analysis, that is the IDA disassembler. IDA too uses a reduced version of VCG to display call graphs.

When we take the strings generated by bdiff as abstract representation of functions, we can use several algorithm from Computational Biology to perform some kind of comparison. One of the easiest way to compare strings is based on the “Alignment Problem” [27]. Basically we search an alignment of the strings, so that identical symbols match, and we allow the use of a special symbol, usually a dash, to represent a symbol insertion (or, if we look at the corresponding string, a deletion). Than we score the alignment we obtain, using some distance function (the so called “edit distance”, which just measures the number of different symbols, is the simplest distance that can be used for this purpose).

In the following, we show the analysis performed on a single functions as an example; of course, it is necessary to take into account the call graph structure and compare functions accordingly. Currently, the function coupling has been done by hand, thanks to the reduced number of subroutines present in these binaries. However, this part too can be done using an algorithm which performs a graph comparison.

We selected the following “string-functions” analyzing the call graph:

```
8049758: esmasssmmmammscsmscscmssssrm
8049d20: esammsasmammmscsssscmcamsssrmm
```

then, we run the alignment algorithm on it:

```
./align
./align by Gerardo Lamastra
-a   dump alignment string
-i   insertion/deletion cost (double)
-m   modification cost (double)
string1 string2
# ./align -a esmasssmmmammscsmscscmssssrm \
            esammsasmammmscsssscmcamsssrmm
es-masssmmmammscsmscscms--sssr-m
esammsasmam-mmscs-ssscmcamsssrmm
L(s1):28 L(s2):30 L_max:30 Score:11 L(align):32 Matched:21
```

L(s1), L(s2) and L(align) are the two string lengths and the length of the corresponding alignment, where symbols may have been inserted. The score is a measure of the edit distance, and we have 21 different symbol matches. The algorithm, based on the description given into [27], is implemented using a Divide-and-Conquer approach.

We can run similar tests over all the functions that compose our binaries. Of course, this can be very time-consuming and also produce several different false positive matches or mismatches. The technique is far from being a mature technique and has been presented in this practical because it was originally developed within it. However, since the same kind of problem have been faced in the context of Computational Biology, it is also possible that this approach could be extended, allowing to search a big signature database for abstract functional descriptions which relate an "unknown binary" with some of the best known attack tools.

## 1.7 Legal Implications

The analysis in this section will be based on Italian laws, as required in the practical specification. The main difference between the Italian and the American legal system lays in their origin. American system is based on the "Common Law", while the Italian system is based on the "Civil Law". It is not appropriate to delve into details of these definitions [1], however, we can state that in Civil Law doctrine has precedence over jurisprudence, while the opposite is true for Common Law. This basically means that an Italian Judge has to follow a set of rules (basically the Civil and Penal Codes) when administering the justice, while an American Judge essentially can motivate his/her decision on previous cases. The articles we are going to cite are part of the Penal Code (C.P.) which is essentially the set of rules which determines what can be considered illegal, from a penal point of view, and what are the appropriate punishments.

If it is possible to demonstrate that the program was installed by an intruder, we have the following violations:

- **Art 615, ter. c.p.** This article establish that it is illegal to access a computer system without permission. If such a violation can be established, This article is a consequence of Law number 547 of 1993. The possible consequence is jail for a period variable between 6 months and 3 years.

[1] See [35] for a detailed analysis about the differences between Common Law and Civil Law

Since the program acts essentially as a remote backdoor, we can also have further violations, especially if it is possible to demonstrate that the system (or the company) suffered a damage as a consequence of the intruder action. In this case, it is possible to invoke the following violations:

- **Art.635 bis c.p.** This violation applies if the system has been damaged or suffered some service interruption, also due to re-installation of software because of attack an clean-up.
- **Art. 621 c.p.** This violation applies if secret data has been revealed to other person as a consequence of the backdoor installation.

It is important to observe that the Administrator Privilege is required to run this program. So, if the program use[2] can be tracked down to an authorized local user, the previous violations may be still admissible, because the user has escalated privilege, so he/she has illegally accessed the computer system.

If the program has been installed by a legitimate administrator, only policy violations can be effectively demonstrated. Indeed this program acts as a kind of remote access mechanism, so if the administrator is allowed to access the machine remotely, and there is no specific policy that establishes an exact way this has to be done, it would be very difficult to accuse the administrator.

## 1.8 Interviewing a Suspect

The goal of a suspect interview is to discover something we still do not know about the case under investigation. We know that a covert-channel backdoor has been installed on an attacked system. We could be interested in understanding the motivation behind this activity, or we could be interested in how the intruder broke in.

Preparing an interview would require at least some knowledge about the suspect. We should know if the suspect is the administrator of the victim machine (or the administrator of other machines in a close administration domain), just a common user or an outsider which broke into out system. It would be also useful to have some knowledge about the computer skills of the suspect, especially whether or not he/she knows security practices and counter-measures.

### 1.8.1 Interviewing the administrator

If we are facing an administrator, one of the possible goal of the interview could be to understand why the administrator installed and used a covert channel tool. As discussed in [29], a good way to start the interview would be to sympathize with the suspect:

"I have seen that you are doing an excellent job here with all these different systems. Can you tell me what are your preferred tools to monitor your machines and keep them all up and running?"

"What are the most common tools used in the company for monitoring and machine administration?"

"Have you ever had problems about accessing your machines remotely, because of some sort of firewall or routing problem? What kind of tool would you use to diagnose and eventually solve that problem?"

[2] Although the program can be owned from a common user, it is needed to be executed as root to make it work

We can continue with this kind of questions for a while. Maybe, at this point, if the administrator thinks he has not done anything bad, he/she could easily admit that he/she installed LOKI2 for some "benign" purpose. Of course, this statement should be further investigated. If we do not get any collaboration, we can push things a bit harder on him/her.

"I know that sometime security people put their firewalls here and there, and this makes remote administration painful. Are you using some specific technique to circumvent their devices?"

"You know, some weird ICMP packets have caused our Intrusion Detection System to flash various alarms; indeed, this traffic was pretty weird, and did not seemed to be regular ICMP. We traced back this traffic as originating from your workstation; can you tell us something about this traffic?"

"Look, here we have a serious company policy violation: we have discovered that you are using some inadmissible mechanism to access those machines. Until now, no serious damage has been revealed, but before this thing escalates up to management, it is better for you to tell me now all you know about this weird ICMP traffic we have detected".

### 1.8.2 Interviewing an external intruder

In this situation, we can be interested in understanding how the hacker succeeded in installing the backdoor. We still can use sympathy to establish a link with the suspect.

"We have noticed that you have used a very interesting trick to broke into our system and control it remotely. We are really interested in understanding more about this.
If the suspect does not acknowledge its activity and refuse to "confess", we can switch to fear in order to push him; at the same time, it is very important to offer a way to get out. As interviewers, our primary goal is to seek and obtain truth.

"We have tracked down a flow of weird ICMP packets from our server to an IP address assigned to your account. Since this system has been compromised, before we start a formal investigation, we would like to settle this thing quietly. If you can help us to undercover the weaknesses in our system, all the possible charges against you could be dropped"

"Look, this system is an element of a very important computer networks; if something has been damaged, you could incur in a lot of troubles. Until now, we have managed to keep this thing at a low profile, but it is very important for us to assess exactly what has been done on this system, how did you circumvent security measures and what kind of sensitive data have you put your eyes on".

### 1.8.3 Conclusions

Interviewing the suspect can be a giant time-saving activity, especially if we cannot exactly identify the mysterious binary. If we succeed in obtaining a full confession, we could have the program source code and analyze it instead of performing all the steps we outlined in the previous sections.

# Part II

# Part 2a - Forensic Analysis of a System

## 2.1 The Synopsis

It is Friday January, 24th 2003. The company network lab requires the analysis of a compromised system. The machine is a “sort of honeypot” which has been setup by the administrator of the lab for an internal project. The machine has been connected to an ADSL link (256K upstream/1M downstream) using a Cisco 800 router. Actually, the router has been used like a modem and the machine has been assigned a public IP address. The following diagram sketches the configuration:

```
[LINK ADSL]----[Cisco 800]----[Hub]----Victim Machine
                                 |
                              Other Machine
```

The lab has a security door which requires a special magnetic badge to access in the area where the machine is located. The administrator informs us that only 27 people out of about 200 have the required privilege to enter the area. The machine is located on a desk, and appears to be connected to the Cisco 800 using a 3Com 10/100 Hub. There is another machine connected to the hub, extremely similar to the victim machine.

The machine has been turned off by the administrator few days ago, pulling the cord (he says he was in a big hurry, when he decided to shut it down). He also says that he has identified a network traffic spike, consisting of FTP traffic going in and out the victim machine and has decided to turn the machine off and terminate the experiment. The only analysis option is a post-mortem analysis.

## 2.2 Hardware Identification

The victim machine is one of the old models used throughout the company; it is an HP Kayak. We collect the following information regarding the hardware:

- Make: Hewlett Packard
- Model: Kayak Workstation XV
- Company Inventory Number: XXX-XXXX-XXXX
- CPU: Pentium 2, 266 MHz, (ID: FR80326707)
- Memory: 128 MB (L2 Cache: 512 KB)
- Video Board: Matrox MGA-G200
- Disk Controller: Adaptec AIC-786x (on board)
- Disk Controller (2nd): Adaptec AIC-7880
- Disk: Seagate ST34572W (8 GB), attached to 2nd controller
- Network card: HP LAN (00:60:B0:B3:E2:BC)
- Other interfaces/ports: 2 serials, 1 parallel, USB

- Other devices: floppy disk, IDE CD-ROM (Matshita CD-585)
- Keyboard: standard 105 US keyboard
- Mouse: PS/2
- BIOS: Phoenix BIOS 4.0 (Rel 6.0) HB.11.07US
- BIOS Boot Sequence: CD-ROM, HD, Floppy
- Operating System Installed: Linux Mandrake 9.0
- IP Addressing: dynamic
- Hostname: moria

The administrator inform us that the machine has to be reused in a different project, so we cannot seizure any hardware element, including the disk. For this case, disk would have been our only proof; we could have used a tag like this:

- Case Number: 1
- Tag Number: 1
- Investigator Name: Gerardo Lamastra
- Starting Date: Jan 24 2003
- Place where the item was recovered: Network Lab, Company XXXXXXXXX, Computer System Inventory Number: XXX-XXXX-XXXX
- Item: SCSI Hard Disk, Make: Seagate, Model: ST34572W

When we ask about the other machine, the administrator says that he was using it to monitor for excessive outgoing traffic; it was indeed this machine to signal the increase in FTP traffic that motivated the end of the experiment. There is no useful data that can be extracted from the other machine, so we leave it alone.

## 2.3 Forensic Imaging Process

We start the forensic process, setting up the following configuration: another machine, the forensic workstation, is connected with the victim machine using the hub. All the other systems are unplugged by the hub. Boot order is verified, so to guarantee that the CD-ROM will be booted first. We use a home-made Linux rescue disk, which has all the needed tools. The disk has been built using a standard Mandrake Rescue, which comes with `dd`, `nc` and `md5sum` which are all the required tools to perform the process. The forensic workstation is setup using a private address 192.168.1.2. A netcat listener (`nc -l -p 1234 > <file>`) is activated on port 1234. The victim system is booted; network is setup for a single point-to-point connection with the forensic workstation:

```
# ifconfig lo 127.0.0.1
# ifconfig eth0 192.168.1.1
```

First of all, we collect the information about the partition table:

```
# fdisk -l /dev/sda

Disk /dev/sda: 255 heads, 63 sectors, 553 cylinders
Units = cylinders of 16065 * 512 bytes

Device Boot    Start       End    Blocks   Id  System
/dev/sda1   *      1         6     48163+  83  Linux       /boot
/dev/sda2          7       325   2562367+  83  Linux       /usr
/dev/sda3        326       442    939802+  83  Linux       /home
/dev/sda4        443       553    891607+   5  Extended
/dev/sda5        443       491    393561   83  Linux       /
/dev/sda6        492       524    265041   83  Linux       /var
/dev/sda7        525       553    232911   82  Linux swap

# fdisk -l /dev/sda | nc 192.168.1.2 1234
```

We also collect the boot sector:

```
# dd if=/dev/sda bs=512 count=1 | nc 192.168.1.2 1234
1+0 records in
1+0 records out
# dd if=/dev/sda bs=512 count=1 | md5sum | nc 192.168.1.2 1234
1+0 records in
1+0 records out
```

We check the md5sum on the forensic workstation, to be sure that the copy was successful (the first two commands are related to those executed on the victim machine):

```
# nc -l -p 1234 > bootsect
# nc -l -p 1234 > bootsect.md5
# md5sum bootsect
d4a47e37d6da503645f6b96fb00a9c87  bootsect
# cat bootsect.md5
d4a47e37d6da503645f6b96fb00a9c87  -
```

Labelling was used on the original partitions; this makes easier to identify partitions without mounting it. We dump this data to the forensic workstation, in a file called `part-table.txt`. Next, we collect md5 signatures:

```
# md5sum /dev/sda1 | nc 192.168.1.2
# md5sum /dev/sda2 | nc 192.168.1.2
# md5sum /dev/sda3 | nc 192.168.1.2
# md5sum /dev/sda5 | nc 192.168.1.2
# md5sum /dev/sda7 | nc 192.168.1.2
```

These are the results, grouped in a single file, which is stored on the forensic workstation. The dash which resulted from the operation has been substituted with the symbolic name of the file:

```
9f558febfa360401fae694ed07623b88  moria-sda1
8915a94c37f723a405a1775c053a2419  moria-sda2
a8976c82abdc8c570fffd66636bb3c5b  moria-sda3
b1879fccb530ce3d398d72e0b5fff6c3  moria-sda5
7467a3546d6f451100b7fae090801a5d  moria-sda6
ca0af974d3436e56b90b6bf4eff90757  moria-sda7
```

The last step is the effective copy:

```
<victim> # dd if=/dev/sda<X> bs=512 conv=noerror
                | nc 192.168.1.1 1234
<fw>     # nc -l -p 1234 | dd of=moria-sda<X>
```

We need to redo partition 2, because the first time we had an MD5 mismatch. The process is very slow, and takes the entire day.

## 2.4 Filesystem analysis

Few days after, on February 3rd 2003, we start the real analysis, using a Linux workstation. First, we try to mount the partitions, using a coherent scheme:

```
# mount -o ro,loop,noatime moria-sda5 /hacked/
# mount -o ro,loop,noatime moria-sda1 /hacked/boot
# mount -o ro,loop,noatime moria-sda2 /hacked/usr
# mount -o ro,loop,noatime moria-sda3 /hacked/home
# mount -o ro,loop,noatime moria-sda6 /hacked/var
```

The mount command automatically finds the proper filesystem which results to be ReiserFS. This is a very bad news, since autopsy and task does not handle this particular filesystem. Much worse, the moria-sda3 does not mount. The error indicates that the fs type is wrong, but we did not specified any. We check md5 again, but everything matches. We try to take a look at the internal structure of the file, and indeed it seems a ReiserFS. The standard ReiserFS signature “ReIsEr2Fs” is present inside the file. We decide to copy the file and run `reiserfsck` on it. Maybe the file-system got into an inconsistent state because of the abrupt machine shutdown. We copy moria-sda3 as moria-sda3-2, and use `losetup` to attach the file to a loop device. The first run does not work either, and reiserfsck advice to use the `--fix-fixable` switch. We do it, and let reiserfsck[3] repair the partition:

```
reiserfsck, 2002 - reiserfsprogs 3.6.3Will check consistency
                   of the filesystem on /dev/loop0
Will fix what can be fixed w/o --rebuild-tree
Will put log info to 'stdout'
```

[3] output lines have been broken for better text formatting

```
Do you want to run this program?[N/Yes] (note need to type Yes):
###########
reiserfsck --fix-fixable started at Fri Feb 28 19:33:23 2003
###########
Replaying journal..
Journal header's mountid: 18
latest transaction found is of mountid 16
Should those transactions be replayed?(Y/n)[n]
trans replayed: mountid 16, transid 308, desc 1921,
         len 4, commit 1926, next trans offset 1909
trans replayed: mountid 16, transid 309, desc 1927,
         len 5, commit 1933, next trans offset 1916
trans replayed: mountid 16, transid 310, desc 1934,
         len 2, commit 1937, next trans offset 1920
...
1108 transactions replayed
Checking S+tree ok
Comparing bitmaps..Trying to fix bitmap ..
 bitmaps were not recovered.
    You can either run rebuild-tree or live with 6201 leaked blocks
Checking Semantic tree...
on-disk bitmap does not match to the correct one.
No corruptions found
There are on the filesystem:
    Leaves 12
    Internal nodes 1
    Directories 99
    Other files 180
    Data block pointers 265 (2 of them are zero)
    Safe links 0
###########
reiserfsck finished at Fri Feb 28 19:33:27 2003
###########
```

The repair is not complete, but the filesystem now mounts correctly and displays the expected directory structure. Of course, the MD5 has been altered because of the process and now we have a different entity, at least from a legal standpoint. However, we have several opportunities:

- The reiserfsck does not alter the filesystem by adding data which can be then linked to the illegal activity. It is possible to inspect the program source code in order to assess that it operates correctly. So we can ask to admit this modified item as a regular proof.
- We could accept the risk, and run the reiserfsck on the original system before we seizure the original data. In this case, slight portion of the filesystem could be modified in such a way we could loose interesting evidence about what has been done on the system. This is like doing a live analysis of some kind.

- We can do the analysis on the checked copy, and try to extrapolate results on the bad filesystem, using some tool for block level analysis.

In this particular situation, we have discovered that the /home partition does not contains anything unusual, so any option can be safe.

When the complete filesystem has been made available, we start with the standard analysis; first of all, we gather some basic information:

```
/etc/issue/issue.net
  Welcome to
  Mandrake Linux release 9.0 (dolphin) for i586
  Kernel 2.4.19-16mdk on an i686

/etc/sysconfig/autologin
  AUTOLOGIN=yes
  USER=federico
  EXEC=/usr/X11R6/bin/startx

/etc/sysconfig/clock
  ZONE=Europe/Rome

/etc/sysconfig/drakconnect
  NET_INTERFACE=ppp0
  type=adsl_pppoe

/etc/sysconfig/drakconnect.adsl_pppoe
  login=XXXXXXX@virgilio.it
  passwd=XXXXXXX


/etc/sysconfig/iptables
  ...
  -A POSTROUTING -s 192.168.1.0/255.255.255.0 -o ppp0 -j MASQUERADE
  ...
  :INPUT ACCEPT [196:26161]
  :FORWARD ACCEPT [453:249872]
  :OUTPUT ACCEPT [187:17601]

/etc/sysconfig/keyboard
  KBCHARSET=C
  KEYBOARD=us
  KEYTABLE=us

/etc/sysconfig/mouse
  MOUSETYPE=ps/2
  XMOUSETYPE=PS/2
```

```
  FULLNAME="PS/2|Standard"
  XEMU3=yes
  WHEEL=no
  device=psaux

/etc/sysconfig/usb
  USB=yes
  KEYBOARD_AT_BOOT=no
  MOUSE=no
  KEYBOARD=no
  STORAGE=no
  VISOR=no
  PRINTER=no
```

This is pretty common stuff. Everything is coherent with the information we have. The ISP account data has been sanitized to remove sensitive information. We notice that the time-zone is (obviously, since this system is located in Italy) Europe/Rome; the Operating System is Linux Mandrake 9.0. The hardware configuration is coherent with the info we gathered in section 2.2. For reasons that will be clear in the next sections, please notice that the system has a PS/2 mouse and no USB mouse has been detected or installed.

The machine has been packed up with all the possible services; by browsing the /etc/rc.d/rc5.d/ directory we have the following list: iptables, network, portmap, syslog, partmon, nfslock, gpm alsa, sound, random, xfs, netfs, dm, atd, snmpd, named, sshd, rawdevices, xinetd, nfs, keytable, httpd, numlock, proftpd, internet, crond, smb, kheader, adsl, devfsd, linuxconf, local. This data is confirmed by taking a look at /var/lock/subsys, where we have a single file for each functionality that has been activated. We derive other information from the following files:

```
xinetd enabled services:
  fam, telnet

/etc/exportfs
  /usr/tmp (rw, insecure, root_squash)

/etc/smb.conf
  ...
  [homes]
    browaseable=yes
    writable=yes
  [tmp]
    path = /usr/tmp
    read only=no
    public=yes
  ...
```

The default runlevel is 5, which means that the system will boot in graphic mode; autologin, a fancy Mandrake feature, is active; so after a successful boot, the user federico will be automatically logged in.

httpd and proftpd have been installed with default profile; we have a static home-page in /var/www/, which is different from the standard Mandrake home-page. Also, proftpd has been installed with anonymous access and /var/ftp/pub/incoming is world writable. Upload is admitted.

Password analysis, done using John the Ripper [13] and an Italian word-list, reveals that some account have trivial passwords:

```
root:toor:0:0:root:/root:/bin/bash
piero:piero:501:501:Piero:/home/piero:/bin/bash
federico:corridore:502:502:Federico:/home/federico:/bin/bash

3 passwords cracked, 3 left
```

The other users are gerry, dario and daniela[4], whose password resisted John. No weird accounts are present in the /etc/passwd and /etc/group files.

Whoa, the system is surely too open for being a real production system. It is definitely a honeypot. We proceed by searching for "weird" file signatures:

```
# find /hacked -perm +02000  -type f -ls
...
# find /hacked -perm +04000  -type f -ls
```

No weird setuid/setgid files, only the standard that we would expect on a Linux/Mandrake. We can check the integrity of those files after, when we run the integrity test on the whole file-system.

```
# find /hacked -name "..*" -type f
# find /hacked -name "..*" -type d
# find /hacked/dev -type f
# find /hacked/usr/share/doc -type d
# find /hacked/usr/share/man -type d
# find /hacked/usr/man -type d
# find /hacked/usr/X11R6/man -type d
```

No `...` directories, not weird hidden directories into /dev/ or other highly populated directories. The system seems to be fairly clean until now.

### 2.4.1 Integrity Analysis

Under Linux, it is very easy to perform an integrity check of the whole file-system. The `rpm` command, which is used to install, remove and build a package, can also perform a complete integrity check, by comparing the size, MD5 signature, permission, type, owner and group. The syntax for this operation is:

```
rpm -Va --root /hacked
```

The –root switch indicates to run the command as the root file-system is the one given as argument. The data is displayed using a compact form, but it is very easy to interpret, using the rpm man page.

[4] user accounts are somehow inspired by colleague names

The check is run against the local database, so this is not completely trusted. However, we can still have some valuable information. Of all the checks, permission, size and MD5 mismatch are the most interesting. 150 files fails at least a single check. Browsing the list of the files, we can immediately understand that several files have been effectively modified because a change of configuration or some other admissible purpose. To isolate "interesting files", it is possible to use `grep` with a pattern like this: (`..5`) to detect modified MD5s or (`S..`) to detect wrong file size.

Browsing the list, we notice a really weird file which displays modification file: `/usr/bin/lddlibc4`. The file has a wrong length, and a wrong MD5 signature. Being a binary file, this is really something that has to be investigated with great detail.

We immediately run an rpm command to identify what package this file belongs to:

```
# rpm -qf /usr/bin/lddlibc4
glibc-2.2.5-16mdk
```

The first "un"educated guess would be a trojan of some kind. There is no man page for this program, but this seems to be a dynamic library dependency walker for very old libc versions.

We download a clean copy of the rpm file, and we run a close compare between the two files:

```
# cmp -l lddlibc4 /hacked/usr/bin/lddlibc4
cmp: EOF on /usr/bin/lddlibc
```

This means that the two file are essentially the same, only there is something sticked on the top of the "hacked" file. This excludes trojans or virus-like programs, because a virus (to be "useful") has to inject something into their host program; whence, lddlibc4 original code should have been modified. If we compare the file size, we can easily extract the extra data at the end of the file:

```
# ls -l lddlibc4 /hacked/usr/bin/lddlibc4
...   3556    lddlibc4
...   6654    /hacked/usr/bin/lddlibc4
# dd if=/hacked/usr/bin/lddlibc4 of=lddlibc4.diff bs=1 skip=3556
3098+0 records in
3098+0 records out
```

We take a closer look at the data (using `file` and `vim -b`), we just notice a random sequence of data, it seems to be impossible to make sense of it. We decide to continue the investigation, hoping we can find more evidence about this file as we go on.

### 2.4.2 Log Analysis

Logs are, of course, one of the richest source of informations. Linux logrotate will gzip older log file. To ease the analysis, we first gunzip the gzipped log files, than we merge them in a single file (of course, we do this on copies):

```
# find . -name "*.gz" -exec gunzip \{\} \;
# perl -e 'foreach $c (reverse(`ls <prefix>*`)) \
        { print `cat $c`; };' > <prefix>.complete
```

where prefix is one of “auth.log”, “boot.log”, “messagese”, “syslog”, and all the others.

First of all, by browsing auth.log, we gain further evidence that federico is the account usually used by the system administrator; in fact, we often find that sudo is used within that account to perform several administrative tasks. We also discover that named (the DNS server) and proftpd (the FTP server, including anonymous access) have been installed just after the main setup process. Auth.log also contains evidence of some telnet and ssh probes:

```
Dec 24 22:46:35 ... telnet pid=9852 from=80.181.185.102
Dec 25 00:20:37 ... telnet pid=10047 from=80.117.176.170
Dec 26 01:17:14 ... telnet pid=16172 from=62.11.80.7
Dec 26 01:17:18 ... telnet pid=16173 from=62.11.80.7
Dec 26 01:17:24 ... telnet pid=16174 from=62.11.80.7
Dec 28 01:56:42 ... telnet pid=29936 from=67.82.51.82
Dec 31 17:22:09 ... telnet pid=25285 from=80.181.185.209
Jan 10 11:44:30 ... telnet pid=31848 from=192.167.97.174
Jan 18 23:56:16 ... telnet pid=25575 from=62.211.249.165
Jan 19 20:00:02 ... telnet pid=1755 from=80.181.123.73
Jan 19 21:32:42 ... telnet pid=2060 from=80.181.123.73
Jan 20 13:22:01 ... telnet pid=7459 from=80.117.206.76
```

We find similar evidence in syslog, where we can relate a match for any of the previous line:

```
Dec 24 22:46:36 linux telnetd[9852]: ttloop:  peer died: Invalid
                or incomplete multibyte or wide character
```

This behavior corresponds to some sort of network scan (like an NMap Syn-Scan).

We can also detect some SSH scans; these are coming in two different “flavors”:

```
Dec 26 19:48:09 linux sshd[21982]: scanned from 212.112.228.1
              with SSH-1.0-SSH_Version_Mapper.  Don’t panic.
```

This appear to be generated by ScanSSH, by N. Provos (check it out on http://monkey.org/ provos/scanssh/). The following command helped to “collect” all the scanner’s address:

```
# grep ssh auth.log | awk ’{print $7 " " $8}’|grep from | \
  sed ’s/from //’ | sort |uniq
162.42.11.25
212.112.228.1
213.134.175.209
```

The other scans are tracked in the auth.log file with the message “Did not receive identification string from $< IP >$”. Using grep again we can sort out the incriminated address from the file:

```
162.42.11.25
194.38.176.241
212.112.228.1
213.132.38.101
213.154.70.102
213.46.44.142
62.146.219.165
80.181.123.73
```

This too seems to be related to people trying to exploit older version of SSH.

We can also find some evidence of statd RPC exploits:

```
Dec 19 21:40:55 linux rpc.statd[900]: gethostbyname error for
        ^X^X^Z^Z%8x%8x%8x%8x%8x%8x%8x%8x%8x%62716x%hn
        %51859x%hn\220\220\220\220\220\220\220\220\220\220...
```

we cannot find evidence about the IP address where the exploit come from. However, none of these attacks has been successful because the machine was installed with patched version of the servers. We can check this by looking at the RPM version of the installed program.

Samba server is another collector of attack tries; since samba logging is done on host-by-host basis in the samba directory, we can easily check this; the bigger majority of this logs reveal the typical signature of Nimda. It is almost incredible to believe, but we got 21684 different IP address doing samba scans that resembles Nimda activity.

The most noticeable intrusion can be located into FTP server logs. After few FTP scans, where we just get the connection opened and suddenly closed, we identify someone probing the `incoming` directory. This directory is usually created to allow anonymous uploads; this is often considered a security issue, because everybody can transform your site into a warez repository, or even something worse. This event can be tracked by looking at several sources:
xferlog:

```
...
Wed Jan 15 21:13:36 2003 0 d106245.upc-d.chello.nl
40 /var/ftp/incoming/-=_www.Mp3Xs.com_=- b _ i a
anonymous@on.the.net ftp 1 * c
Wed Jan 15 21:13:49 2003 8 d106245.upc-d.chello.nl
114843 /var/ftp/incoming/crash_bandicoot_front.jpg b _ i a
anonymous@on.the.net ftp 1 * c
Wed Jan 15 21:13:58 2003 4 d106245.upc-d.chello.nl
114843 /var/ftp/incoming/crash_bandicoot_front.jpg b _ o a
anonymous@on.the.net ftp 1 * c
Wed Jan 15 21:14:44 2003 0 d106245.upc-d.chello.nl
40 /var/ftp/incoming/mp3xs.com/elcon/_____________________
_____________________mp3xs.com/-=_www.Mp3Xs.com_=- b _ i a
anonymous@on.the.net ftp 1 * c
Thu Jan 16 01:59:45 2003 2633 d106245.upc-d.chello.nl
33812382 /var/ftp/incoming/mp3xs.com/elcon/_______________
_____________________________mp3xs.com/
Veiled_Leah_Andreone.mp3xs b _ i a
guest@ ftp 1 * c
...
```

syslog (or messages):

```
...
Jan 15 21:13:26 linux proftpd[5272]:
```

```
  linux (d106245.upc-d.chello.nl[213.46.106.245]) -
  FTP session opened.
Jan 15 21:15:10 linux proftpd[5272]:
  linux (d106245.upc-d.chello.nl[213.46.106.245]) -
  FTP session closed.
Jan 15 22:47:07 linux proftpd[5492]:
  linux (d106245.upc-d.chello.nl[213.46.106.245]) -
  FTP session opened.
Jan 15 22:48:18 linux proftpd[5492]:
  linux (d106245.upc-d.chello.nl[213.46.106.245]) -
  FTP session closed.
Jan 16 01:15:45 linux proftpd[5837]:
  linux (d106245.upc-d.chello.nl[213.46.106.245]) -
  FTP session opened.
Jan 16 02:00:17 linux proftpd[5837]:
  linux (d106245.upc-d.chello.nl[213.46.106.245]) -
  FTP session closed.
...
```

Of course, we can check out the MAC times for the relevant files in /var/pub/incoming; they match perfectly:

```
drwxr-xr-x   2 76  76       48 Jan 15 08:49 _ax/
-rw-r--r--   1 76  76   114843 Jan 15 21:13 crash_bandicoot_front.jpg
drwxr-xr-x   3 76  76       72 Jan 15 21:14 mp3xs.com/
drwxr-xr-x   3 76  76       80 Jan 15 15:56 test/
-rw-r--r--   1 76  76       40 Jan 15 21:13 -= www.Mp3Xs.com =-
```

So, we got a Netherlands ISP account, which is presumably used for some kind of broadband home service. This information can be pulled down from the RIPE web server:

```
inetnum:      213.46.96.0 - 213.46.111.255
netname:      TK-HLM-CABLE
descr:        Chello DHCP Brabant
country:      NL
```

If we take a further look at the file that have been downloaded, we find a .jpg image[5] (see picture 2.5).

The other interesting file has been deeply nested into a set of weird directories, containing spaces; its name is “Veiled_Leah_Andreone.mp3xs”, and the extension does not tell us anything. However, we just need to run `file` over it to discover that we are facing a RAR file, that is a kind of archive format, just like zip. Next steps: searching a rar decompresser for Linux on the Net, and using it to further analyze the content of the archive; as we can expect, we find a set of 11 sound-tracks and a CD cover, which is included as reference 2.6.

[5]The image has been marked with a red “Proof” writing, to make it clear that they have been used illegally by the intruder


Figure 2.5: The “Crash Bandicoot Picture”

Interesting enough, when we analyze web server logs, we only find Code Red (both I and II) access patterns, or other IIS unicode attempts. Of course, they all fail on the Apache Web Server (1.3.26). Nobody has even tried to access the web server home page, but a couple of robots (something which declares himself as “Micanet Site Leecher”). Another pretty common hack is the use of “CONNECT” method for proxying purposes. When we look at the Apache SSL logs, we can find tracks of someone trying to exploit the Slapper vulnerability. None of these operations has succeeded.

The only interesting thing we can find in the web server log is an FTP /incoming/ access:

```
217.229.172.147 - - [17/Jan/2003:16:55:57 +0100]
"GET /incoming/mp3xs.com/elcon/%20%20%20%20%20%20%20%20%20%20
%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20
%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20mp3xs.com/
Veiled_Leah_Andreone.mp3xs HTTP/1.1"
404 404 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows 98;
Win 9x 4.90)"
```

(the single log line has been broken into several different lines for typesetting purposes). This is clearly some kind of human-like activity, since it refers a file that is effectively present on the FTP server. The “%20” is a char encoding for a single space character. As we said before, the file was placed into a weird directory hierarchy, where one of the nodes had a name filled with whitespace characters. Maybe, someone finding the FTP server unaccessible (too many user logged), is trying to download things using the web server.

After this stuff has appeared onto the FTP server, we can observe several downloads, just looking at the `xferlog` file. We do not place the xferlog data here, since it is very long.

If we analyze syslog and kernel/info file we can easily dig out informations about reboots, to check for other kind of anomalies:

```
# grep started info.5
```


Figure 2.6: The “The CD Cover”

```
Dec 18 14:41:27 linux kernel: klogd 1.4.1 ... started.
Dec 18 15:02:52 linux kernel: klogd 1.4.1 ... started.
Dec 18 15:23:59 linux kernel: klogd 1.4.1 ... started.
Dec 18 15:43:16 linux kernel: klogd 1.4.1 ... started.
Dec 19 11:46:06 linux kernel: klogd 1.4.1 ... started.
Dec 19 12:15:14 linux kernel: klogd 1.4.1 ... started.
```

and they match the corresponding entries in syslog and message files:

```
# grep restart syslog
Dec 18 14:41:27 linux syslogd 1.4.1: restart.
Dec 18 15:02:52 linux syslogd 1.4.1: restart.
Dec 18 15:23:59 linux syslogd 1.4.1: restart.
Dec 18 15:43:16 linux syslogd 1.4.1: restart.
Dec 19 11:46:06 linux syslogd 1.4.1: restart.
Dec 19 12:15:14 linux syslogd 1.4.1: restart.
Dec 22 04:02:03 linux syslogd 1.4.1: restart.
Dec 29 04:02:02 linux syslogd 1.4.1: restart.
Jan  5 04:02:02 linux syslogd 1.4.1: restart.
Jan 12 04:02:03 linux syslogd 1.4.1: restart.
Jan 19 04:02:04 linux syslogd 1.4.1: restart.
```

The last five entries are not real reboots; they are due to logrotate actions, which is usually executed by crond daemon at regular times; in this case, it is 4:00am; (we can find further evidence of this in the cron logs).

### 2.4.3 History file analysis

We analyze the all the "history" files we can find on the system. Of course, shells and Internet browser leave history files; but also other program may use something to record previous commands that the user provided.

We start from `/root/.bash_history` which is the most obvious place where to look; at the beginning of the file, we can notice something "suspect":

```
ls
rm -f .bash_history
cat .bash_history
ifconfig
ps ax
ifconfig
tail /var/log/messages
/etc/init.d/adsl stop
/etc/init.d/adsl start
killall -1 pppd
ps ax
killall -1 /usr/sbin/pppd
ps ax
killall -9 /usr/sbin/pppd
ps ax
killall -9 /usr/sbin/pppoe
ps ax
/etc/init.d/adsl start
ifconfig
cd /etc/rc.d/
```

first of all, the `rm -f .bash_history` is a clear sign of history file tampering; someone is trying to hide something. We can see that our user also looks at the file to check that everything seems to be fine. Of course, since the shell will save the .history_file at exit, the last commands are appended to the file, which will be re-created if it has been canceled. This explains the presence of the `rm` command in our file. Moreover, we also find that this user has done something with the ADSL connection, which is the only network connection of our server. So these commands have been given while working at the console. Someone working locally on the machine has then cleared the history file. The system is into a protected environment (the lab had a door that only opens with a specific badge), so only some authorized personnel can access the machine. There is no other interesting element into this file.

Looking at the Internet history logs, we only find references to the local site and to `rpmfind.net` which is a repository of RPM files. There is nothing interesting here.

We continue by searching other possible log info; we can browse the `.viminfo` file and the `.mc/history` file, both present in the /root directory. The .viminfo file has a lot interesting information; first of all we find further evidence of .bash_history tampering

```
# Command Line History (newest to oldest):
...
```

```
:r ~/.bash_history
:w! ~/.bash_history
:w ~/.bash_history
...
```

The administrator has read and written the file.

The “register section” also contains some weird commands:

```
# Registers:
...
"3      LINE    0
        rm -f usbmouse.o
"4      LINE    0
        insmod ./usbmouse.o
"5      LINE    0
        ls
"6      LINE    0
        cd /tmp/
"7      LINE    0
        ls
"8      LINE    0
        umount /mnt/disk/
...
```

We have seen that the victim machine has a PS/2 mouse. What does it mean that the administrator has manipulated the usbmouse.o module, which is usually loaded by the system when required? Moreover, if we check usbmouse.o in the proper directory, we find that the module is usually kept in gzipped form; also, its MAC times appear unchanged by the default install.

The .mc/history file gives evidence that the administrator discovered the FTP intrusion, that is the files in the /var/ftp/pub/incoming directory:

```
...
[Dir Hist New Left Panel]
0=/root
1=/var/ftp/incoming/_ax
2=/var/ftp/incoming
3=/var/ftp/incoming/mp3xs.com
4=/var/ftp/incoming/mp3xs.com/elcon
5=/var/ftp/incoming/mp3xs.com/elcon/
6=/var/ftp/incoming/mp3xs.com/elcon
7=/var/ftp/incoming/mp3xs.com
8=/var/ftp/incoming
...
```

This enforces the evidence that the administrator logged after the file were loaded on the FTP server and used mc to browse through the subdirectories of the victim machine.

The ordinary accounts are substantially untouched. Only the federico account has been used, and we have already seen that this account is linked to the machine administrator. There is nothing interesting

here, however, some file appear to have been damaged, probably because of the file-system problems which we need to correct by using the proper fsck tool.

### 2.4.4 MAC Times Analysis

We processed MAC times using `mac_daddy`; we then analyzed the MAC time list, searching for specific information in order to validate some assumption we did previously. First of all, the list has 76270 records; many of these records just refer to standard and uninteresting values. We are only interested in values ranging from December 18 2002 till January 22 2003.

The first signs of the installation is at line 27502:

```
Dec 18 2002 14:15:24     9462 .a. -rw-r--r--  root     root
      /hacked/root/drakx/auto_inst.cfg.pl
Dec 18 2002 14:15:29  5562838 m.c -rw-r--r-- root     root
      /hacked/var/lib/urpmi/hdlist.Installation CD 1 (x86) (cdrom1).cz
                        54685 m.c -rw-r--r-- root     root
      /hacked/var/lib/urpmi/synthesis.hdlist.Installation CD 1
                              (x86) (cdrom1).cz
```

This can be traced to Mandrake Installation mechanism. However, we have few records that appears just before these ones:

```
Dec 17 2002 14:38:47     1114 m.. -rw-r--r-- root     root
            /hacked/etc/X11/fs/config
                          299 m.. -rw-r--r-- root     root
            /hacked/etc/crontab
                           86 m.. -rw-r--r-- root     root
            /hacked/etc/modules.conf
                           88 m.. -rwxr-xr-x root     root
            /hacked/etc/sysconfig/mouse
                           89 m.. -rwxr-xr-x root     root
            /hacked/etc/sysconfig/network
```

These files are usually set-up and written during installation, when the system perform the autoconfiguration. Maybe, this weird December 17 2002 date is just a clock mismatch, which has been fixed at the end of the installation, when the administrator is asked to configure the clock and pick up the timezone. The M time, however remains the one assigned when these file were lastly written.

Of course, the vast majority of MAC Time information is located in the installation phase. Many files indeed have been left untouched during the system life. At line 63284, we have what seems to be the first reboot:

```
...
Dec 18 2002 14:40:24    10207 .a. -rwxr-xr-x root     root
     /hacked/etc/hotplug/usb.agent
Dec 18 2002 14:40:25    48146 .a. -rw-r--r-- root     root
     /hacked/lib/modules/2.4.19-16mdk/kernel/3rdparty/3c990/3c990.o.gz
                        41070 .a. -rw-r--r-- root     root
```

```
     /hacked/lib/modules/2.4.19-16mdk/kernel/3rdparty/3c990fx/3c990fx.o.gz
...
```

Just before hotplug starts, we have the creation of the installation files on the /root/drakx/ directory. First reboot is also confirmed if we look at syslog output.

At line 64465, user federico appears for the first time. This is probably the first user to log in the system. We have already seen that federico is usually used by the administrator, and is also the autologin user.

```
Dec 18 2002 14:41:50        0 m.c -rw-r--r-- 502      502
     /hacked/home/federico/.drakfw
```

Indeed, 502 is the userid of federico. User piero (uid 501) has been inserted in passwd before federico, so it is possible that all users have been created simultaneously; we can check this by looking at the creation times for the /home/¡user¿/.bash_profile:

```
Dec 18 2002 14:32:51      191 mac -rw-r--r-- 504      504
      /hacked/home/daniela/.bash_profile
                          191 mac -rw-r--r-- 505      505
      /hacked/home/dario/.bash_profile
                          191 m.c -rw-r--r-- 502      502
      /hacked/home/federico/.bash_profile
                          191 m.c -rw-r--r-- 503      503
      /hacked/home/gerry/.bash_profile
                          191 ..c -rw-r--r-- 501      501
      /hacked/home/piero/.bash_profile
```

this file is usually created when the user is created.

At line 65082, some snmp related files make their first appearance on the system: the administrator is installing some other stuff which has not been installed during the initial process:

```
...
Dec 18 2002 15:32:39       80 m.c drwxr-xr-x root     root
     /hacked/etc/snmp
... [other snmp files follows]
```

At line 65261, the user root starts mozilla for the first time, since mozilla related files are created:

```
...
Dec 18 2002 15:58:10    17486 .a. -rw-r--r-- root     root
     /hacked/usr/lib/mozilla-1.1/chrome/locale.alias
Dec 18 2002 15:58:14      936 m.c drwxr-xr-x root     root
     /hacked/usr/lib/mozilla-1.1
                          6568 m.c drwxr-xr-x root     root
     /hacked/usr/lib/mozilla-1.1/components
                         69635 m.c -rw-r--r-- root     root
     /hacked/usr/lib/mozilla-1.1/components/xpti.dat
Dec 18 2002 15:58:19       80 m.c drwxr-xr-x root     root
```

```
     /hacked/root/.mozilla/default
                           80 m.c drwxr-xr-x root     root
     /hacked/root/.mozilla/default/drb9avl3.slt/chrome
...
```

At line 66535 the "first day" configuration seems to be finished and the computer stays quiet until next day:

```
Dec 18 2002 18:24:06       19 .a. -r--r--r-- root     root
     /hacked/etc/X11/app-defaults/XLogo
Dec 19 2002 10:21:34       25 m.c -rw------- root     root
     /hacked/root/.GnuDIP2.cache.XXX.XXXXXX.XX
Dec 19 2002 10:25:26   158092 .a. -rwxr-xr-x root     root
     /hacked/bin/tar
                         22501 ..c -rw-r--r-- root     root
     /hacked/var/www/html/banner.jpg
                           599 ..c -rw-r--r-- root     root
     /hacked/var/www/html/index.html
```

the administrator installs the web server page, and starts to test the GnuDIP package, which is used to interact with a Dynamic DNS. If we look at GnuDIP home-page, we see that this is a program that can be used to update the IP/DNS association for a machine which is connected to the Internet using a dynamic address. The "XXX" have been used to obscure the name used for the experiment (this was an explicit request of the victim's administrator).

At line 67507, we got the last access time for `tcpdump` which has been possibly used for network troubleshooting and tuning. We are still in a "post-configuration" stage.

```
Dec 19 2002 11:01:54       14 .a. lrwxrwxrwx root     root
     /hacked/usr/lib/libpcap.so.0
                        137520 .a. -rwxr-xr-x root     root
     /hacked/usr/lib/libpcap.so.0.7
                        340648 .a. -rwxr-xr-x root     root
     /hacked/usr/sbin/tcpdump
```

At line 67516, we have another evidence of lddlibc4 tampering. This is a weak evidence, but however corroborates our previous results. We find that `mdir` and `mtype` have been used at the same time when `lddlibc4` is accessed. We conjecture that a correlation may exist; since mdir and mtype are used to list and type files on a floppy disk, it is pretty unclear what this correlation could really be:

```
Dec 19 2002 11:06:47        6 .a. lrwxrwxrwx root     root
     /hacked/usr/bin/mdir
Dec 19 2002 11:07:01        6 .a. lrwxrwxrwx root     root
     /hacked/usr/bin/mtype
Dec 19 2002 11:07:50     6654 .a. -rwxr-xr-x root     root
     /hacked/usr/bin/lddlibc4
```

Configuration is still going on. The administrator creates the file chkadsl.sh, which is used for ADSL monitoring, and places the file in the cron table.

```
Dec 19 2002 12:04:30      307 m.c -rwxr--r-- root     root
     /hacked/root/chkadsl.sh
```

At line 68601, the FAT related modules are loaded; again, the administrator is doing something using an external disk? No FAT partition are present on the main hard-disk, so the most obvious place where a FAT can be located is a removable floppy. However, this is only a conjecture. There is a slight discrepancy here, because first our administrator was using mtools, now he[6] uses direct file-system access. The question is, are these guys the same person? Or there is a couple of people who is working on the machine?

```
Dec 19 2002 12:24:47       52 .a. -rw-r--r-- root     root
     /hacked/etc/filesystems
                         21210 .a. -rw-r--r-- root     root
     /hacked/lib/modules/2.4.19-16mdk/kernel/fs/fat/fat.o.gz
                          8248 .a. -rw-r--r-- root     root
     /hacked/lib/modules/2.4.19-16mdk/kernel/fs/vfat/vfat.o.gz
Dec 19 2002 12:24:48     1960 .a. -rw-r--r-- root     root
     /hacked/lib/modules/2.4.19-16mdk/kernel/fs/nls/nls_cp437.o.gz
                          1514 .a. -rw-r--r-- root     root
     /hacked/lib/modules/2.4.19-16mdk/kernel/fs/nls/nls_iso8859-1.o.gz
Dec 19 2002 12:25:20    10776 .a. -rwxr-xr-x root     root
     /hacked/bin/sync
```

At line 68609, we have the first sign of an “external” activity. Someone (“enzo”??) is trying to log in the samba server:

```
Dec 19 2002 15:43:15        0 ma. -rw-r----- root     root
     /hacked/var/log/samba/log.enzo
Dec 19 2002 16:01:02        0 ..c -rw-r----- root     root
     /hacked/var/log/samba/log.enzo
```

This is interesting: in less than a complete day, our machine has been scanned by Nimda. If we had installed a Windows machine and we did not apply patches before connecting it to the network, we would have a high probability of being compromised by Nimda (or Code Red). These worms are very hard to exterminate!

Configuration continues: this lines can be correlated to CD-ROM usage:

```
Dec 19 2002 16:07:00     9898 .a. -rw-r--r-- root     root
     /hacked/lib/modules/2.4.19-16mdk/kernel/fs/inflate_fs/inflate_fs.o.gz
                         15149 .a. -rw-r--r-- root     root
     /hacked/lib/modules/2.4.19-16mdk/kernel/fs/isofs/isofs.o.gz
                         51842 .a. -rw-r--r-- root     root
     /hacked/lib/modules/2.4.19-16mdk/kernel/fs/udf/udf.o.gz
```

At line 69471, we have other Nimda activity: the log files for zohe, sara and brazil have just been created. Finally, at line 69469 root exits the system: Christmas holidays have just begun:

```
Dec 19 2002 16:26:55       24 .a. -rw-r--r-- root     root
     /hacked/root/.bash_logout
                          3060 .a. -rwxr-xr-x root     root
     /hacked/usr/bin/clear
```

[6] assuming a man, because federico was used as name

From now on, we have only other “external activities”; mainly samba. At line 69878, we can find the first signs of the ftp “/incoming” bug being exploited:

```
Jan 15 2003 15:56:00       72 m.c drwxr-xr-x 76       76
       /hacked/var/ftp/incoming/test/| testtest
                           48 m.c drwxr-xr-x 76       76
       /hacked/var/ftp/incoming/test/| testtest/test
Jan 15 2003 15:56:38       80 m.c drwxr-xr-x 76       76
       /hacked/var/ftp/incoming/test
Jan 15 2003 21:13:36       40 m.c -rw-r--r-- 76       76
       /hacked/var/ftp/incoming/-= www.Mp3Xs.com =-
Jan 15 2003 21:13:49   114843 m.c -rw-r--r-- 76       76
       /hacked/var/ftp/incoming/crash_bandicoot_front.jpg
Jan 15 2003 21:14:06      216 m.c drwxrwxrwx 76       76
       /hacked/var/ftp/incoming
Jan 15 2003 21:14:12       72 m.c drwxr-xr-x 76       76
       /hacked/var/ftp/incoming/mp3xs.com
Jan 15 2003 21:14:35      120 m.c drwxr-xr-x 76       76
       /hacked/var/ftp/incoming/mp3xs.com/elcon
Jan 15 2003 21:14:44       40 m.c -rw-r--r-- 76       76
       /hacked/var/ftp/incoming/mp3xs.com/elcon/
                    mp3xs.com/-= www.Mp3Xs.com =-
Jan 16 2003 01:15:52      136 m.c drwxr-xr-x 76       76
       /hacked/var/ftp/incoming/mp3xs.com/elcon/mp3xs.com
Jan 16 2003 01:59:45 33812382 m.c -rw-r--r-- 76       76
       /hacked/var/ftp/incoming/mp3xs.com/elcon/
        mp3xs.com/Veiled_Leah_Andreone.mp3xs
```

At line 75918, we find M/C time for lddlibc4. This just confirms what we already know: the program has been tampered.

```
Jan 22 2003 15:27:49     8712 .a. -rwxr-xr-x root     root
     /hacked/sbin/lsmod
Jan 22 2003 15:27:56    88920 .a. -rwxr-xr-x root     root
     /hacked/sbin/insmod
                             6 .a. lrwxrwxrwx root     root
     /hacked/sbin/rmmod
                          6654 m.c -rwxr-xr-x root     root
     /hacked/usr/bin/lddlibc4
Jan 22 2003 15:27:58      888 .a. drwxr-xr-x 502      502
      /hacked/home/federico
```

but at the same time, these simple lines also reveal few interesting details. First of all, the lddlibc4 is written at the same time a `lsmod`, `insmod`, `rmmod` sequence is executed. So it seems that modifying the file is somehow correlated to module insertion/removal. Moreover, at the same time, the federico home is being accessed, so it seems that this activity may be correlated to normal root behavior. In other words, we conjecture that lddlibc4 may have been tampered by someone with full privilege on

the system, who is also operating directly on the victim's console. Maybe, this single information is worth the time we spent by looking at almost 80000 different entries in the time-line file.

### 2.4.5 String analysis

We have collected enough evidence till now. We cannot do any easy "undelete" magic on ReiserFS. There are few interesting tools that can be used on ReiserFS, they are the support tool written by Hans Reiser himself. Luckily, these tools have been written with a modular approach, so it is possible to use the ReiserFS core library to build your own file-system analysis tool. ReiserFS is built using two basic elements:

- **BTrees**, an highly optimized data-structure which can speed up file access and block location in the file-system.
- **Journaling**, which should allow the implementation of file-system operation as "transactions", which allow to maintain the integrity of the file-system.

All the data in a ReiserFS is organized into trees; a node of the tree can contain different kind of informations, essentially file-system metadata, journal data, raw data. For further information about ReiserFS internals, refer to [32] or [19].

The information we have gathered until this point seem to point out that there is a correlation between the following elements:

- lddlibc4
- usbmouse.o
- m-tools, and something that was on an external floppy disk
- kernel module manipulation tools, especially insmod/rmmon

We run `strings` on the raw partition data. This is both space consuming and time consuming, so we prefer to optimize the process using an empirical strategy: first of all, we analyze swap space (sda7) and root file-system (sda5). Swap space is always an interesting place to search, because of volatility order; the root file-system is another interesting place where to conduct the search, because /root directory is located there and, if our hypothesis is correct, several facts we are investigating can be connected to administrator activity. We search explicitly for "lddlibc4" in both the partitions, and we find some interesting results:

```
sda7.string:
  US/ASCII
  a.tgz
  lddlibc4
  usbmouse.o
  vlogger.c
  Makefile
  decode
  ze in shared object, consider re-linking
  cannot restore segment prot after reloc
```

We have some other interesting “strings” which lies too closely to the incriminated string:: a.tgz, usbmouse.o, vlogger.c, decode. This is the only occurrence of lddlibc4 in the whole swap space. We continue our search for the strings we just identified, and we can locate other interesting occurrences:

```
/root/.bashrc
/etc/init.d/adsl start
tar -xzf a.tgz
mozilla &
rpm -qa|grep nautilus|less
nslookup XXX.XXXXXX.it
mdir
LS_OPTIONS
rm -f a.tgz
G_BROKEN_FILENAMES=1
vi /etc/resolv.conf
nslookup www.cisco.com
mv addon-modules ../html.orig/
```

So, this seems to be a residual of some history file. We can see that a.tgz has been unpacked and then the original archive file has been removed. Continuing our search, we find:

```
mcopy a:a.tgz
```

This is another proof that confirms our hypothesis. a.tgz was on a diskette, and the administrator has probably copied it on the local system, untarred it, and used something inside it. Further search for vlogger.c, usbmouse.o and decode does not yield other interesting information.

We then analyze sda5 for these keywords: lddlibc4, a.tgz, decode, usbmouse.o, vlogger.c. The first match for lddlibc4 is indeed very interesting:

```
kernel_version=2.4.19-16mdkcustom
using_checksums=1
license=GPL
author=rd@vnsecurity.net
%.2d/%.2d/%d-%.2d:%.2d:%.2d
/usr/bin/lddlibc4
[%s tty=%s/%d]
USER/CMD %s
PASS %s
[%s%d] <%s uid=%d %s> %s
[%s%d] <%s> %s
[ALT-’]
[ALT-,]
[ALT--]
...
```

The “kernel_version”, “using_checksum” and “license” tags point the finger at the standard structure of a Linux Kernel Module. /usr/bin/lddlibc4 is just in the middle of this stuff. Moreover, we have also

the author of the module, or at least something we can easily put in a google search. Before we ask Google, we continue to search for our string and we have another interesting piece of data:

```
rmmod usbmouse
mount /dev/fd0 /mnt/floppy/
cd /mnt/floppy/
./decode ./lddlibc4 /usr/bin/lddlibc4 > /root/results.txt
mount
cd ..
umount floppy/
mount floppy/
cd floppy/
./decode ./lddlibc4 /usr/bin/lddlibc4 > /root/results.txt
cd /root/
vi results.txt
rm results.txt
rm results.txt
cd /mnt/floppy/
cp lddlibc4 /usr/bin/lddlibc4
cat go
insmod ./usbmouse.o
cd ..
umount floppy/
logout
```

This seems to be a cleared .bash_history file. We can do a good conjecture now: someone has removed the kernel module trick, then is using an external tool, decode, which he has placed on a floppy to perform some sort of “decoding”. The data is then placed in “result.txt” file. The file is analyzed, and then cleared. We add result.txt in the keyword list. The search for “lddlibc4” concludes with another fragment of data which strictly resembles the first piece we found:

```
kernel_version=2.4.18-6mdk
using_checksums=1
license=GPL
author=rd@vnsecurity.net
/usr/src/linux/include/linux/dcache.h
/usr/src/linux/include/linux/sched.h
/usr/src/linux/include/linux/file.h
[%s tty=%s/%d uid=%d %s]
USER/CMD %s
%.2d/%.2d/%d-%.2d:%.2d:%.2d
/usr/bin/lddlibc4
[%s%d] <%s uid=%d %s> %s
[%s%d] <%s> %s
[%s tty=%s/%d]
USER/CMD %s
```

```
PASS %s
[ALT-’]
[ALT-,]
[ALT--]
[ALT-.]
```

When we search for “vlogger.c”, we immediately find that this string lies near lddlibc4, embedded into pieces of other strings which usually constitute the content of an object file. As for lddlibc4, we found two different instances of this string, like there are two distinct copies of the file that contains both the /usr/bin/lddlibc4 and vlogger.c strings.

We got only a “false positive” for a.tgz, which is a match due to the “manpages-es-0.3.a.tgz” string. usbmouse.o also is a source of false positive, because of the presence of the “usbmouse.o.gz” on the file-system (a legitimate kernel module).

There is, however, no “result.txt” present in the strings extracted from the root partition, while “decode” is a fairly common word to be searched. If “decode” is effectively used to decode stuff which is hidden in lddlibc4, we can safely expect that the two strings will be always paired, when we search for them.

All the mystery seems to be confined into these two file-systems; the other partitions appear to be clean from lddlibc4 issues. It is time to search the Net. The first thing we search is for “vlogger.c lddlibc4 author=rdvnsecurity.net”. It is disappointing that this string does not yield any result. We try with restricted version, and eventually we get a result with “vlogger.c rd@vnsecurity.net”: a Phrack article again [31]. The paper discuss the design and implementation of a tty sniffer. The sniffer has the ability to log keystrokes as well as all pty session, so to include remote ssh and telnet sessions. The result of the file is placed in a /tmp/log/pass.log, but no coding is executed. There is no mention of lddlibc4.

Summarizing, we can then do the following hypothesis:

- usbmouse.o is inserted in the kernel; with a high degree of probability, usbmouse.o is the compiled version of vlogger.c. The original program has been probably modified to include an encoding process and the data saving stage on the tail of the /usr/bin/lddlibc4 file.
- “decode” can be used to decode the data saved by usbmouse.o
- this “hack” has been performed locally, and not remotely, by someone using a privileged account. There are no extra signs which make us think of some external (network) intrusion to be behind this.

At this point, it could be possible to perform some extra analysis on the lddlibc4 data tail. If the output format is something like:

```
snprintf(loginfo, sizeof(loginfo)-1,
    "<%s uid=%d %s> %s",
    date_time, task->uid, task->comm, tmp->buf);:
```

as we can derive by reading the source code, we can run different set of correlation tests in order to decode the information. This indication, can be used to search for a possible decoded file. As we have seen, someone has dumped the result of the decoding process on a file “result.txt”, which has not been immediately identified. Now, we have some extra element: we can search for a string with the following pattern: “uid=”. Unfortunately, this research does not yield any result.

### 2.4.6 Finding and extracting the data

As an alternative, we can try to recover the usbmouse.o. First of all, we need to isolate data blocks from file-system metadata. In order to do so, we can use the core ReiserFS library, which has been developed by H. Reiser for writing file-system level tools. The library is a part of the "reiserfsprogs-3.6.4". The library is not very well documented, but it is possible to understand some of its functions by analyzing the source code of the other utilities. The following code is strongly based on `debugreiserfs.c`. The program basically dumps the unused data blocks; we start with the data initialization, and we create an extra bitmap to hold the block numbers we are interested in. The bitmap is a compact representation of the block allocation in the file-system.

```
...
/* Ok, open file-system */
fs = reiserfs_open(name, O_RDONLY, &error, NULL);
if (no_reiserfs_found(fs))
  barf(BARF_ABORT,"Error: Cannot open reiserfs on %s\n", name);
/* Get the journal, I'm assuming is on the same device */
reiserfs_open_journal(fs, name, O_RDONLY);

/* Get the bitmap & select appropriate area
 * Default: UNUSED_BLOCKS
 */
if (!reiserfs_open_ondisk_bitmap(fs))
  barf(BARF_ABORT,"Error: Cannot open ondisk bitmap\n");
/* This will manipulate the bitmap to select all/used/unused blocks */
bm = process_bitmap(fs, scan_bitmap);
```

When we have the bitmap ready, we just process all the blocks within it; we identify the block, and if we find a THE_UNKNOWN block, OK that is what we are searching for:

```
/* We now start working on our manipulated bitmap */
for (b_scan = 0; b_scan < get_sb_block_count(fs->fs_ondisk_sb); b_scan++) {
  /* Skip unset bits */
  if (!reiserfs_bitmap_test_bit(bm, b_scan))
    continue;

  /* Read b_scan block & check it out... */
  bh = bread(fs->fs_dev, b_scan, fs->fs_blocksize);
  /* What sort of block is this? */
  type = who_is_this(bh->b_data, bh->b_size);
  switch (type) {
  case THE_UNKNOWN:
    /* raw data for a file... Maybe, this can be the most
     * interesting data for us...
     */
    dumpblock(outfile, bh);
    break;
```

```
...
```

this is only a fragment of the program, which can be found on [22]. Using this program, we can transform a ReiserFS image file into a sequence of data blocks. Then, we can try to locate a "lddlibc4" into these blocks, and going back from that point until we locate the ELF signature. So, if we are lucky enough, the blocks that make up the usbmouse.o program are just consecutive and we can extract the binary object. The file end is not as important as the beginning, and can be matched by looking at the general structure of kernel module objects. Vim is very good for performing this job: first we forward search for lddlibc4, then we back search for ELF so to find the beginning of the object file. We then use the "paste" command in visual mode to select the portion of the file we are interested in; the result is yanked into a new file, which is saved on disk.

This is like using a hammer to carve a gem, but at least we obtain something we can analyze. Unluckily, the program is not a perfect ELF object; objdump (and the bdiff program we developed for Part I) complains about it. If we try to pull out the call graph from this object file, we have a bitter surprise: relocation records (needed to complete the call graph) are not where objdump expect them to be. They are still in the file, and using `strings` or `vi` we can effectively locate them and see that they are "almost" where they should be. However, computers do not work very well with "almost". Especially when it come to rigidly structured files, like binary executables.

If we analyze the files working by hand, they are indeed very similar; although we cannot have a complete and perfect result, we believe that all this data can be used to build a solid proof in a courtroom, if there is the need to do so.

## 2.5 Conclusions

The most interesting section of this analysis is the conclusion. After gathering all this evidence to support the conjecture that the administrator has installed this sniffing tool (usbmouse.o AKA vlogger.c), it is time to ask. The answer is quick and easy: since the system was an honeynet, and the administrator was interested in monitoring SSH exploits he had heard of, he decided to install this TTY sniffer to analyze ssh sessions. When he discovered that the honeypot was being used as FTP repository, he decided to stop the test. It is easy to remember few words from K. Privette, on my fourth day of lesson: "Obviously, internal and external sysadmins can be key sources to utilize"; and also "If you could just get the offender to tell you 5 minutes of details you may be able to save weeks or even months of investigative resources". Indeed, it took some time to collect all the required information about the lddlibc4 mystery.

The real "intrusion" on this system was much more evident. The FTP server, with the incoming directory world writable has become a place for exchanging audio stuff. Once someone from Holland put the data on the disk, we suddenly become popular among digital music seekers on the Network. We have recovered the data and IP address from which the intrusion originated, and we have enough data to prove it.

# Part III

# Part 3 - Legal Issues of Incident Handling

## 3.1 Introduction: the Italian Legislation

The analysis of the case scenario presented in part three of the practical assignment is based on the Italian legislation.

The case basically present a law enforcement officer who is asking an ISP to provide logging information about an account, which seems to be the originator of an attack.

The differences between American and Italian law system, the Common Law and the Civil Law, have already been touched in section 1.7. The most important element that we need to take into account to analyze the proposed scenario is the Privacy Law, also known as Law 675/96 [3]. This law acknowledge the European Directive 95/46/CE [26], which essentially states that privacy is a fundamental right for all the citizens of the European Union.

The law basically prescribes that any subject that is required to handle personal information has to declare how this data will be handled. Of course, it only applies to "Personal Data". Network logs, which documents the network activity of a certain subject, are Personal Data, since they can be used to reveal information about sexual life, political beliefs, health status, race, religion.

Of course, the provider has to inform the signer of the contract [7] that the network activity is logged. The log is strictly confidential and can only be provided after a court warrant.

There is no specific law concerning the way to maintain and manage log files, while general and specific restrictions apply when personal data are stored and/or processed by electronic means. The Privacy Law is the first concern. There is lesser concern about proper client identification. If fact, Law Decree no. 103, dated March 17 1995, declares that a provider can acquire on-line the required information to activate a contract. However, it is very easy to forge false information and use those information to sign the contract. Of course, if the login account and the telephone number used to place the call can be linked together, it is possible to better identify who was effectively using the account. Formally, the A.I.I.P., the Italian Internet Provider Association, states that log data should be maintained for five years. However, there is no strict requirement by law to save or record this kind of data. Especially smaller provider often fail to do so. Since the privacy law puts a strong concern on the process used to handle log data, bigger provider too may decide to not archive logs. Also, the ISP is not required by law to record the any information about the Caller ID, which can be used to establish the telephone number that was used to place the call.

Newer rules, especially Presidential Decree (DPR) no. 318, dated July 28 1999 [4], demand a strong attention on the privacy of the logs and require a minimal set of security measures to be adopted, in order to prevent any illegal disclosure of such data. This Decree essentially states how privacy law should be practically applied.

Many issues of the current legislation on Computer Crime have been addressed in the following paper [28], available on the Italian Justice Ministry web-site. In the paper, the author states that there are a lot of concerns about the trade-off existing between the application of the Privacy law and the requirements of law enforcement and there is no commonly accepted set of rules, which defines exactly how the provider should manage the logs.

[7] A typical contract may be found on [1]

## 3.2 The Answers

Question A: if we consider the strong requirements of the Privacy law, we cannot release any "Personal Information" to law enforcement, if they do not provide a warrant. This is clearly stated in [3].

Question B: there is nothing that law enforcement can do to force the provider to maintain the logs, because the ISP is not even explicitly required to have logs. The lack of such enforcement is discussed in [28]. Of course, if a warrant is provided, the ISP has to provide all the help needed to trace any further connection from the suspect account, included access to the logs. If it is possible to prove that the ISP intentionally destroyed a proof, other "classical" violations may apply.

Question C: the law enforcement officer has to ask the public prosecutor to issue a warrant that allow him to access the logs. The article n. 226 of the Penal Procedure Code [2] documents the proper way to obtain such a warrant. This article has been modified after the September 11 event, being replaced by special law 244, date October 19 2001 [5], which gives special attention when there is terrorism suspect.

Question D: again, the Privacy law does not allow the administrator to perform any kind of activity which is not required for proper management of the service. The law states that Personal Data management shall be limited only to the extent required for providing the service.

Question E: if an external attacker has broken into the ISP network, Privacy Law does not apply, since such a hacker is not one of your customer. Data may be shared freely with the law enforcement, but there is no obligation in doing so. It is a provider's choice, until law enforcement does not provide a warrant. The administrator can perform any activity, on the system under his/her responsibility, to identify the hacker and assess the damage. Of course, the activity cannot be extended to systems outside the administration domain.

# Bibliography


[1] Tetanet general isp contract. On-Line: http://internet.teta.it/contratto.html.

[2] Law decree 271 - decreto legge n. 271, July 1989.

[3] Privacy law (legge n. 675). Gazzetta Ufficiale. On-Line: http://www.interlex.it/testi/l675_96.htm, December 1996.

[4] Regolamento recante norme per l'individuazione delle misure minime di sicurezza per il trattamento dei dati personali, a norma dell'articolo 15, comma 2, della legge 31 dicembre 1996, n. 675. Gazzetta Ufficiale. On-Line: http://www.privacy.it/dpr318-1999.html, July 1999.

[5] Law decree 374 - decreto legge n. 374. http://www.infoleggi.com/news/terrorismo.htm, October 2001.

[6] Route (aka daemon9). Project loki: Icmp tunneling. *Phrack Magazine*, 7(49):Article no. 6, August 1996.

[7] Route (aka daemon9). Loki2: The implementation. *Phrack Magazine*, 7(51):Article no. 6, September 1997.

[8] J. Cespedes. *ltrace: A Dynamic Library Call Interception Program*. Available on: http://freshmeat.net/projects/ltrace/.

[9] S. Cloves. *Injectso: a Tool to Inject Shared Libraries into Running Process*. Secure Reality. Available on: http://www.securereality.com.au/.

[10] G. Combs and al. *The Ethereal Network Analyzer*. Available on: http://www.ethereal.com/.

[11] The VMWare Corporation. *VMWare: Virtual Machines for x86 Architecture*. WebSite: http://www.vmware.com/.

[12] I. F. Darwin. *file, a Program for Determining the File Type*. Available on: http://www.gnu.org/.

[13] Solar Designer. *John the Ripper Password Cracker*. Available on: http://www.openwall.com/john/.

[14] Free Software Foundation. *GNU BinUtils*. Available on: http://www.gnu.org/.

[15] Free Software Foundation. *GNU BinUtils: The BFD library*. Available on: http://www.gnu.org/.

[16] Free Software Foundation. *GNU TextUtils*. Available on: http://www.gnu.org/.

[17] G. Sander I. Lemke. *The VCG Tool: A Visualization Tool for Compiler Graph*. University of Saarlandes, Germany. Available on: ftp://ftp.cs.uni-sb.de/pub/graphics/vcg/vcg.tgz.

[18] SANS Institute. *The Unknown Binary for GCFA Ver. 1.2b*, September 2002. Available on: http://www.giac.org/gcfa/binary_v1.2.zip.

[19] G. Kurz. Reiserfs on-disk structures. Available on: http://p-nand-q.com/reiserfs/reiserfs.htm.

[20] A. Kuznetsov. *Linux ip(4) man page*. Linux Kernel Documentation Project. For the complete command reference please look at the following document: /usr/share/doc/iproute-2.4.7/ip-cref.ps.

[21] G. Lamastra. *bdiff: a Tool for Binary Program Comparison*. Available on: http://feanor.sssup.it/ lamastra/forensic/.

[22] G. Lamastra. *unrmrfs: a Free Block Dumper for ReiserFS*. Available on: http://feanor.sssup.it/ lamastra/forensic/.

[23] Linux Kernel Documentation Project. *Linux popen(3) man page*.

[24] H. Lu. Elf: From the programmer's perspective. Technical report, NYNEX Science & Technology, Inc., April 1995. Available on: http://ftp.unicamp.br/pub/systems/Linux/GCC/elf.ps.gz.

[25] R. Sladkey P. Kranenburg, B. Lankester. *strace 4.0: A System Call Tracer for SunOS, Linux, and other Operating System*. Available on: http://www.wi.leidenuniv.nl/ wichert/strace/.

[26] European Parliament. Directive 95/46/ce on personal data protection. On-Line: http://www.interlex.it/testi/95_46ce.htm, 1995.

[27] C. N. S. Pedersen. *Algorithm in Computational Biology*. PhD thesis, University of Aarhus, August 1999. Available on: http://www.daimi.au.dk/ cstorm/papers/thesis.ps.

[28] R. Di Pietro. Inquadramento normativo in materia di fornitura di servizi telematici via internet. problematiche concernenti la sicurezza ed il controllo delle comunicazioni. On-Line: http://www.giustizia.it/cassazione/convegni/s15122000.htm.

[29] K. Privette and R. Salgado. *Forensic Framework and Best Practices: Managerial and Legal Issues*, 2002.

[30] The Honeynet Project. *The Reverse Challenge*. Available on: http://www.honeynet.org/reverse/.

[31] rd at thehackerschoice.com. Writing linux kernel keyloggers. *Phrack Magazine*, 11(59):Article no. 14, June 2002.

[32] H. Reiser. *ReiserFS v.3 Whitepaper*. NameSys. Available on: http://www.namesys.com/content_table.html.

[33] G. Kurtz S. McClure, J. Scambray. *Hacking Exposed*. McGraw-Hill, third edition edition, September 2001.

[34] W. R. Stevens. *Unix Network Programming*. Prentice Hall, second edition, January 1998.

[35] W. Tetley. Mixed jurisdictions : common law vs civil law (codified and uncodified). On-Line: http://www.unidroit.org/english/publications/review/articles/1999-3.htm.

Last Updated: June 18th, 2018

# Upcoming SANS Forensics Training


| | | | |
|---|---|---|---|
| SANS Minneapolis 2018 | Minneapolis, MN | Jun 25, 2018 - Jun 30, 2018 | Live Event |
| SANS Paris June 2018 | Paris, France | Jun 25, 2018 - Jun 30, 2018 | Live Event |
| SANS Cyber Defence Canberra 2018 | Canberra, Australia | Jun 25, 2018 - Jul 07, 2018 | Live Event |
| Minneapolis 2018 - FOR508: Advanced Digital Forensics, Incident Response, and Threat Hunting | Minneapolis, MN | Jun 25, 2018 - Jun 30, 2018 | vLive |
| SANS Vancouver 2018 | Vancouver, BC | Jun 25, 2018 - Jun 30, 2018 | Live Event |
| SANS London July 2018 | London, United Kingdom | Jul 02, 2018 - Jul 07, 2018 | Live Event |
| State of Michigan - FOR572: Advanced Network Forensics and Analysis | Lansing, MI | Jul 09, 2018 - Jul 14, 2018 | vLive |
| SANS Cyber Defence Singapore 2018 | Singapore, Singapore | Jul 09, 2018 - Jul 14, 2018 | Live Event |
| SANSFIRE 2018 | Washington, DC | Jul 14, 2018 - Jul 21, 2018 | Live Event |
| SANSFIRE 2018 - FOR585: Advanced Smartphone Forensics | Washington, DC | Jul 16, 2018 - Jul 21, 2018 | vLive |
| SANS Cyber Defence Bangalore 2018 | Bangalore, India | Jul 16, 2018 - Jul 28, 2018 | Live Event |
| Community SANS Columbia FOR500 | Columbia, MD | Jul 23, 2018 - Jul 28, 2018 | Community SANS |
| SANS Riyadh July 2018 | Riyadh, Kingdom Of Saudi Arabia | Jul 28, 2018 - Aug 02, 2018 | Live Event |
| SANS vLive: FOR610: Reverse-Engineering Malware: Malware Analysis Tools and Techniques | FOR610 - 201807, | Jul 30, 2018 - Sep 05, 2018 | vLive |
| Security Operations Summit & Training 2018 | New Orleans, LA | Jul 30, 2018 - Aug 06, 2018 | Live Event |
| SANS Boston Summer 2018 | Boston, MA | Aug 06, 2018 - Aug 11, 2018 | Live Event |
| SANS San Antonio 2018 | San Antonio, TX | Aug 06, 2018 - Aug 11, 2018 | Live Event |
| San Antonio 2018 - FOR508: Advanced Digital Forensics, Incident Response, and Threat Hunting | San Antonio, TX | Aug 06, 2018 - Aug 11, 2018 | vLive |
| SANS August Sydney 2018 | Sydney, Australia | Aug 06, 2018 - Aug 25, 2018 | Live Event |
| SANS New York City Summer 2018 | New York City, NY | Aug 13, 2018 - Aug 18, 2018 | Live Event |
| Community SANS Columbia FOR610 | Columbia, MD | Aug 20, 2018 - Aug 25, 2018 | Community SANS |
| SANS Virginia Beach 2018 | Virginia Beach, VA | Aug 20, 2018 - Aug 31, 2018 | Live Event |
| Data Breach Summit & Training 2018 | New York City, NY | Aug 20, 2018 - Aug 27, 2018 | Live Event |
| SANS Chicago 2018 | Chicago, IL | Aug 20, 2018 - Aug 25, 2018 | Live Event |
| Mentor Session - FOR508 | Copenhagen, Denmark | Aug 22, 2018 - Oct 06, 2018 | Mentor |
| SANS San Francisco Summer 2018 | San Francisco, CA | Aug 26, 2018 - Aug 31, 2018 | Live Event |
| SANS Copenhagen August 2018 | Copenhagen, Denmark | Aug 27, 2018 - Sep 01, 2018 | Live Event |
| SANS Tokyo Autumn 2018 | Tokyo, Japan | Sep 03, 2018 - Sep 15, 2018 | Live Event |
| SANS Amsterdam September 2018 | Amsterdam, Netherlands | Sep 03, 2018 - Sep 08, 2018 | Live Event |
| SANS Tampa-Clearwater 2018 | Tampa, FL | Sep 04, 2018 - Sep 09, 2018 | Live Event |
| SANS vLive - FOR585: Advanced Smartphone Forensics | FOR585 - 201809, | Sep 04, 2018 - Oct 11, 2018 | vLive |