Global Information Assurance Certification Paper

Analysis of an Unknown Binary

SANS GCFA Practical Assignment v.1.2
Part 1

Jacob Cunningham

Table of Contents

Syntax Conventions

The text of the document is in 12 point Arial font

`Commands executed at the shell, the output of commands, and references to files, directories or system binaries are all in 10 or 12 point Courier New font.`

Part 1: Analyze an Unknown binary

Introduction

The security of one of my employer's systems was compromised recently. The system administrator who responded to the incident found an unknown binary named `atd` that was installed by the intruders. The system administrator created an MD5 checksum of the binary and stored it in a file named `atd.md5`. The binary and associated md5 checksum file were then zipped into a file named binary_v1.2.zip and given to me for analysis.
I analyzed and identified the purpose of the binary using the forensic and reverse engineering techniques and tools described below.

I set up a PC running RedHat Linux 7.1 to perform the analysis. I connected the analysis system's network card to another Linux system via a 10Base-T cross over cable to monitor network traffic originating from the analysis systems with the Snort package during the analysis process. I assigned IP addresses in the private 192.168.100.0/24 range on both the analysis and monitoring hosts.

Binary Details

The first step in the analysis process was to extract the binary from the zip file. I used the command `unzip -X` to extract the files. This command extracts the files in the zip archive and preserves their original user and group (UID/GID) information.

Figure 1-1: Unzipping the archive

```
# unzip -X binary_v1.2.zip
Archive: binary_v1.2.zip
   inflating: atd.md5
   inflating: atd
```

The next step in analyzing the unknown binary was to gather basic information about the binary such as the MAC times (the date and time the file was last Modified, Accessed or Changed), file size, owner, and permissions.

`debugfs` is a Linux utility that can examine and edit an ext2 filesystem. I used this utility to gather the unknown binary's size, owner, permissions, and MAC times as shown in Figure 1-2. I accomplished this by first running `ls –i atd` which shows me the inode associated with the file. (See part1 for inode description). I then used `debugfs` to display the contents of the inode structure of the inode (34756) associated with the file using the command `debugfs –R "stat <34756>" /dev/had1`

Figure 1-2: debugfs output

```
# ls -i atd
  34756 atd
# debugfs -R "stat <34756>" /dev/hda1
debugfs 1.27 (8-Mar-2002)

Inode: 34756   Type: regular    Mode:  0644   Flags: 0x0   Version: 1
User:     0   Group:     0   Size: 15348
File ACL: 0    Directory ACL: 0
Links: 1   Blockcount: 32
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x3e314e73 -- Fri Jan 24 09:32:19 2003
atime: 0x3d653432 -- Thu Aug 22 14:57:54 2002
mtime: 0x3d653432 -- Thu Aug 22 14:57:54 2002
BLOCKS:
196931 196932 196933 196934 196935 196936 196937 196938 196939 196940
196941 196942 196943 196944 196945 196946
TOTAL: 16
```

The output from `debugfs` provided me with the following relevant information about the unknown binary. (bolded in Figure 1-2)

- The ctime (create time) of the binary is: Fri Jan 24 09:32:19 2003
  This reflects when the binary was created (unzipped from the zipfile) on my system.
- The atime (last access time) of the binary is: Thu Aug 22 14:57:54 2002
  This represents the last time the file was accessed. The time could indicate when the intruder accessed the file, or when the sys-admin accessed it to zip it.
- The mtime (last modify time) of the binary is: Thu Aug 22 14:57:54 2002
  The represents when the file was last modified. The time could indicate when the intruder accessed the file, or when the sys-admin accessed it to zip it.
- The unknown binary, `atd`, is 15348 bytes in size
- The binary is owned by User 0 (root) and group 0 (root)

- The file permissions of the atd binary are 0644.
  UNIX file's permissions contain three attributes.
  r – read permission – Allows the user to read the file
  w – write permission – Allows the user to write to the file
  x – execute permission – Allows the user to execute the file.
  There are three categories of users on the system. The user (owner), users who belong to the same pre-defined group (group) and all other users on the system (other). Each of these three attributes can be set for each of the three categories of users. The three permission attributes can be represented as letters (r,w,x) or as octal digits (shown below).

  Octal: Attribute:
  1 = execute only
  2 = write only
  3 = write and execute
  4 = read only
  5 = read and execute
  6 = read and write
  7 = read and write and execute

The unknown binary file permissions (0644) show that it is read-able and write-able by the user who owns the file, but is read-only for the group and all other users.

To ensure that the checksum process didn't alter the MAC times of the unknown binary, I verified the MD5 checksum of the binary after gathering MAC time information with `debugfs`.
I performed the MD5 checksum of the binary using the `/usr/bin/md5sum` program and verified that the value obtained matched the MD5 signature stored in the `atd.md5` file that was packaged with the `atd` binary. (See Figure 1-3) This indicates that my analysis so far had not changed the binary.

Figure 1-3: md5sum output

```
# md5sum atd
48e8e8ed3052cbf637e638fa82bdc566  atd

# cat atd.md5
48e8e8ed3052cbf637e638fa82bdc566  atd
```

`md5sum` calculates a hash of binary data using the MD5 hashing algorithm (as described in RFC 1321) . This MD5 hash or checksum of the data is unique only to that data. If the data is modified, the MD5 calculation will not match the previous one. For all intents and purposes it is impossible for two different pieces of data or files to have the same MD5 checksum. Consistently reproducing the same MD5 checksum for the same piece of data proves that the data has not

been modified. This concept is extremely important in the field of computer forensics, and is used to prove the integrity of evidence.

Next, I ran `/usr/bin/file` on the binary, as shown in Figure 1-4 to determine the file type.

Figure 1-4 out of `file` command

```
# file atd

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

The output of `file` in Figure 1-4 shows the unknown binary is a 32-bit ELF executable compiled for the Intel 80386 architecture. This also shows the binary is dynamically linked and stripped. A dynamically linked binary contains references to external shared libraries, and accesses code within these shared libraries when it is executed. A stripped binary is one that has had the symbol table removed. ELF binaries are only supported on UNIX systems, so I knew I was not dealing with a DOS/Window's binary.
After determining the binary was an ELF executable, it seemed odd to me that the execute permission bits were not set on the file, meaning the file could not be executed. I decided to determine if the system administrator who zipped the binary changed the file permissions. To do this, I ran `zipinfo –v` on binary_v1.2.zip to see what type of system the zipfile was created on. There was a lot of output generated from the command, but the significant piece of information about the zipfile is that it was created on a MS-DOS or NT FAT system. (Shown in Figure 1-5)

Figure 1-5: Output of zipinfo

```
# zipinfo –v binary_v1.2.zip
….
File system or operating system of origin:  MS-DOS, OS/2 or NT FAT
….
```

The zipfile binary_v1.2.zip was created on an MS-DOS based system so the original UNIX file permissions were not preserved and it also means that the MAC times on the unknown binary had been modified by the MS-DOS system.

To prove this I ran debugfs –R "stat <34755>" /dev/hda1 to get the inode contents of the `atd.md5` file that the system administrator included with the unknown binary.
The atime and mtime of the `atd.md5` file are both `Thu Aug 22 14:58:08`, which is 15 seconds later than the atime and mtime of the unknown binary, `atd`. This proves that the MAC times of the unknown binary were modified by the system administrator who created the zipfile, and do not represent the MAC times of the binary as it existed on the compromised system.

The next step was to run `/usr/bin/strings` on the binary in order to determine what the binary contains for ASCII string data. The `strings` command parses a file and shows any ASCII text strings such as comments in the code, error messages, passwords, author information, and sometimes library information if the file is dynamically linked. I ran `strings` with the command line option `-n 1`, which prints all string data greater than 1 character in length, rather than the default length of 4 characters.

Figure 1-6 contains a portion of the `strings` output containing keywords that were helpful in determining the function of the unknown binary, and my interpretation of the meaning of the string data. The strings data was produced by running the command: `/usr/bin/strings -n 1 atd`

Figure 1-6: Significant strings output

String data found in binary	My interpretation of the string
/lib/ld-linux.so.1	The binary uses ld-linux.so.1 shared library indicating it was compiled on a Linux system
libc.so.5	The binary is compiled against libc.so.5 shared library - indicates it was compiled on an older Libc5 version of Linux. More recent versions of Linux use Libc6
lokid: Client database full	status message from daemon, indicates it's lokid
2.0 lokid version: %s	Indicates the binary is lokid version (2.0)
XOR active cryptography: %s	Indicates that it uses XOR cryptography in the communications
lokid: inactive client <%d> expired from list [%d]	status message from daemon, indicates it's lokid
lokid -p (i|u) [-v (0|1)]	"help" instructions on command line options. Indicates it's lokid
LOKI2 route [(c) 1997 guild corporation worldwide]	Comment in source code: copyright statement by the author "route"
lokid: server is currently at capacity. Try again later	status message from daemon, indicates it's lokid
lokid: Cannot add key	status message from daemon, indicates it's lokid
lokid: popen	status message from daemon, indicates it's lokid compiled to use popen() to fork a process.
lokid: client <%d> requested an all kill	status message from daemon, indicates it's lokid
lokid: clean exit (killed at client request)	status message from daemon, indicates it's lokid
lokid: cannot locate client entry in database	status message from daemon, indicates it's lokid
lokid: client <%d> freed from list [%d]	status message from daemon, indicates it's lokid
lokid: unsupported or unknown command string	status message from daemon, indicates it's lokid
lokid: client <%d> requested a protocol swap	status message from daemon, indicates it's lokid
lokid: transport protocol changed to %s	status message from daemon, indicates it's lokid

The text from the strings output indicates to me that the unknown binary is the LOKI daemon, which is a well known "covert channel" backdoor program used by intruders to covertly log into compromised systems. (See Phrack Magazine, Issue 51)

It has previously been shown that the atime, mtime, owner and group information is not indicative of the values the binary had on the compromised system. The process used to retrieve and package the `atd` and `atd.md5` files into the zip file altered them, so it's impossible to determine when the binary was last executed. The strings data found in the file (Shown in Figure 1-6) indicate the binary is version 2.0 of the LOKI daemon. Figure 1-7 contains a summary of basic information I gathered about the unknown binary using the methods demonstrated above.

Figure 1-7: Summary of Binary Details

File Attribute	Value
Name	Atd
File Size:	15348 bytes
File Permissions:	0644
Ctime	Fri Jan 24 09:32:19 2003
Atime	Thu Aug 22 14:57:54 2002
Mtime	Thu Aug 22 14:57:54 2002
Owner	0 (root)
Group	0 (root)
Md5 checksum:	48e8e8ed3052cbf637e638fa82bdc566
Keywords found in file:	(See Figure 1-6)
Filetype:	ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, stripped. Compiled on a Linux system with libc5.

Program Description and Forensic Details:

So far in my investigation I have gathered basic information about the binary. The next part of the investigation is to identify exactly how the binary behaves.
The unknown binary is a dynamically linked, stripped ELF binary compiled on a Linux system that uses libc version 5.
I used the `ldd` command to identify all the shared libraries the binary has been linked against as shown in Figure 1-8.

Figure 1-8: Output of ldd command

```
# ldd ./atd
      libc.so.5 => /lib/libc.so.5 (0x4000040000)
```

The output of `ldd` confirms that the binary is linked against the `/lib/libc.so.5` shared library. Libc version 5 was used in older versions of Linux. The current versions of Linux now use Libc version 6.

The ELF binary contains many different sections that are relevant to the structure and execution of the binary. The `.comment` section of the ELF binary usually contains information about the compiler used to compile it.
Dumping the `.comment` section of the unknown binary using the command `objdump –j .comment –s atd` reveals that the binary `atd` was compiled using the GNU GCC compiler version 2.7.2.1 as shown in Figure 1-9.

Figure 1-9: output of objdump

```
# objdump –j .comment –s atd

atd:     file format elf32-i386

Contents of section .comment:
 0000 00474343 3a202847 4e552920 322e372e  .GCC: (GNU) 2.7.
 0010 322e3100 00474343 3a202847 4e552920  2.1..GCC: (GNU)
 0020 322e372e 322e3100 00474343 3a202847  2.7.2.1..GCC: (G
 0030 4e552920 322e372e 322e3100 00474343  NU) 2.7.2.1..GCC
 0040 3a202847 4e552920 322e372e 322e3100  : (GNU) 2.7.2.1.
 0050 00474343 3a202847 4e552920 322e372e  .GCC: (GNU) 2.7.
 0060 322e3100 00474343 3a202847 4e552920  2.1..GCC: (GNU)
 0070 322e372e 322e3100 00474343 3a202847  2.7.2.1..GCC: (G
 0080 4e552920 322e372e 322e3100 00474343  NU) 2.7.2.1..GCC
 0090 3a202847 4e552920 322e372e 322e3100  : (GNU) 2.7.2.1.
```

This is an older version of the GCC compiler indicating the unknown binary may have been compiled on a system with an old operating system. This is consistent with the hypothesis that it was compiled on an older operating system because the binary is linked against libc5.

All programs have a unique “footprint” when they are executed on a system. This “footprint” of the execution of a binary has several characteristics including:

- The system calls that the process executes.
- The signals sent/received by the process.
- The files, file descriptors and system devices used by the process.

The program `strace` in Linux executes a given binary and displays these characteristics. It is an invaluable tool for investigating the behavior of a running process.

Before running `strace` on the binary I had to make the file executable by setting the execute bits using the command `chmod +x atd`. I ran `strace` with the command line options `-ff` which creates individual output files for each forked process. Figure 1-10 shows the output of running the command: `strace -o atd-strace,out -ff atd`

Figure 1-10: Output of main process strace

Function Call from execution of unknown binary.	Description of Call
execve("./atd", ["./atd"], [/* 28 vars */]) = 0	call execve to execute the program
mmap(0, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40007000	map file descriptor to memory
mprotect(0x40000000, 21025, PROT_READ|PROT_WRITE|PROT_EXEC) = 0	set read,write,execute access to mapped memory
mprotect(0x8048000, 13604, PROT_READ|PROT_WRITE|PROT_EXEC) = 0	set read,write,execute access to mapped memory
stat("/etc/ld.so.cache", {st_mode=S_IFREG|0644, st_size=7473, ...}) = 0	get file status of /etc/ld.so.cache
open("/etc/ld.so.cache", O_RDONLY) = 3	open /etc/ld.so.cache (file descriptor 3)
mmap(0, 7473, PROT_READ, MAP_SHARED, 3, 0) = 0x40008000	map file descriptor 3 to memory (ld.so.cache)
close(3) = 0	close file descriptor 3 (ld.so.cache)
stat("/etc/ld.so.preload", 0xbffff9e8) = -1 ENOENT (No such file or directory)	get file status of /etc/ld.so.preload (doesn't exist)
open("/usr/local/qt/lib/libc.so.5", O_RDONLY) = -1 ENOENT (No such file or directory)	open /usr/local/qt/lib/libc.so.5 (doesn't exist)
open("/lib/libc.so.5", O_RDONLY) = 3	open /lib/libc.so.5 (file descriptor 3)
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3"..., 4096) = 4096	read from the file descriptor (3)
mmap(0, 786432, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x4000a000	map info from file descriptor to memory
mmap(0x4000a000, 555135, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED, 3, 0) = 0x4000a000	map info from file descriptor to memory
mmap(0x40092000, 21344, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 3, 0x87000) = 0x40092000	map info from file descriptor to memory
mmap(0x40098000, 204364, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x40098000	map info from file descriptor to memory
close(3) = 0	close filehandle 3 - libc.so.5

mprotect(0x4000a000, 555135, PROT_READ|PROT_WRITE|PROT_EXEC) = 0	set read,write,execute access to mapped memory
munmap(0x40008000, 7473) = 0	unmap memeory at location 0x40008000 (/etc/ld.so.cache)
mprotect(0x8048000, 13604, PROT_READ|PROT_EXEC) = 0	set read,execute access to mapped memory
mprotect(0x4000a000, 555135, PROT_READ|PROT_EXEC) = 0	set read,execute access to mapped memory
mprotect(0x40000000, 21025, PROT_READ|PROT_EXEC) = 0	set read,execute access to mapped memory
personality(PER_LINUX) = 0	set "execution domain". The man page states"personality is Linux-specific and should not be used in programs intended to be portable". This is further proof that the binary was built on and for a LiNUX SYSTEM.
geteuid() = 0	get the process owner's effective identity
getuid() = 0	get the process owner's UID
getgid() = 0	get the process owner's group ID
getegid() = 0	get the process owner's effective group ID
geteuid() = 0	get the process owner's effective identity
getuid() = 0	get the process owner's UID
brk(0x804c820) = 0x804c820	sets end of data segment in memory
brk(0x804d000) = 0x804d000	sets end of data segment in memory
stat("/etc/locale/C/libc.cat", 0xbffff520) = -1 ENOENT (No such file or directory)	get file status of /etc/locale/C/libc.cat
stat("/usr/share/locale/C/libc.cat", 0xbffff520) = -1 ENOENT (No such file or directory)	get file status of /usr/share/locale/C/libc.cat
stat("/usr/share/locale/libc/C", 0xbffff520) = -1 ENOENT (No such file or directory)	get file status of /usr/share/locale/libc/C
stat("/usr/share/locale/C/libc.cat", 0xbffff520) = -1 ENOENT (No such file or directory)	get file status of /usr/share/locale/C/libc.cat
stat("/usr/local/share/locale/C/libc.cat", 0xbffff520) = -1 ENOENT (No such file or directory)	get file status of /usr/local/share/locale/C/libc.cat
socket(PF_INET, SOCK_RAW, IPPROTO_ICMP) = 3	open raw socket for ICMP protocol (file descriptor 3)
sigaction(SIGUSR1, {0x804a6b0, [], SA_INTERRUPT|SA_NOMASK|SA_ONESHOT}, {SIG_DFL}) = 0	define action upon receipt of signal
socket(PF_INET, SOCK_RAW, IPPROTO_RAW) = 4	open raw socket for IP protocol (file descriptor 4)
setsockopt(4, IPPROTO_IP3, [1], 4) = 0	set options on the previously defined socket (fd4)
getpid() = 25965	get PID of current process
getpid() = 25965	get PID of current process
shmget(26207, 240, IPC_CREAT|0) = 5	allocate a shared memory segment
semget(26389, 1, IPC_CREAT|0x180|0600) = 4	initialize a semaphore
shmat(5, 0, 0) = 0x40008000	attach the shared memory segment to the address space of the calling process.
write(2, "\nLOKI2\troute [(c) 1997 guild c"..., 52	writes text "LOKI2 route [© 1997 guild corporation worldwide]" to STDOUT
LOKI2 route [(c) 1997 guild corporation worldwide]	
) = 52	
time([1043426226]) = 1043426226	gets time in seconds since the epoch

close(0) = 0	close filehandle 0
sigaction(SIGTTOU, {SIG_IGN}, {SIG_DFL}) = 0	define action upon receipt of signal
sigaction(SIGTTIN, {SIG_IGN}, {SIG_DFL}) = 0	define action upon receipt of signal
sigaction(SIGTSTP, {SIG_IGN}, {SIG_DFL}) = 0	define action upon receipt of signal
fork() = 25966	fork off a child process to run as daemon (child inherits parent's info including shared memeory segments)
close(4) = 0	close file descriptor 4 (raw IP socket)
close(3) = 0	close file descriptor 3 (ICMP socket)
semop(0x4, 0x2, 0, 0xbffff9a4) = 0	set options for semaphore (4)
Shmdt(0x40008000) = 0	detatches shared memory space from calling program
semop(0x4, 0x1, 0, 0xbffff9a4) = 0	set options for semaphore (4)
_exit(0) = ?	exit the parent process which forked the child

The `fork()` system call in the execution of the `atd` binary creates a child process that inherits the environment and memory of the parent process and listens for incoming connections on a network socket (file descriptor) created by the parent process. The `strace` of the child process forked off by the `atd` program is listed in Figure 1-11.

Figure 1-11: Strace of child process

Function Call	Description of Call
Setsid() = 477	creates a new session for this child process (pid 477)
open("/dev/tty", O_RDWR) = -1 ENXIO (No such device or address)	open /dev/tty
chdir("/tmp") = 0	change directory to /tmp
umask(0) = 022	set umask of 022, files will be created with permissions 0644
sigaction(SIGALRM, {0x8049218, [], SA_INTERRUPT|SA_NOMASK|SA_ONESHOT}, {SIG_DFL}) = 0	define SIGALARM signal actions
Alarm(3600) = 0	send SIGALARM to process after 3600 seconds
sigaction(SIGCHLD, {0x8049900, [], SA_INTERRUPT|SA_NOMASK|SA_ONESHOT}, {SIG_DFL}) = 0	set signal attribute to not notify when child process exits.
read(4, 0x804c78c, 84) = ? ERESTARTSYS (To be restarted)	read from filehandle (looking for loki client connect)

The `strace` output in Figure 1-10 and 1-11 shows the `atd` binary references several files (listed below) when it executes.

- Loads information from the file /etc/ld.so.cache which contains a list of directories to search for shared libraries.
- Checks /etc/ld.so.preload which is a list of libraries to load before the program
- Looks for libc.so.5 in /usr/local/qt/lib (not found) and finds it at /lib/libc.so.5

- Looks for libc.cat in /etc/locale/C, /usr/share/locale/C, /usr/locale/libc/C, /usr/local/share/locale/C

The child process spawned by fork() attempts to open `/dev/tty`, and fails, then executes `chdir` (change directory) to `/tmp`.

My research has shown that the unknown binary is LOKID v2.0. There is not evidence of it containing any malicious code that would cause it to act differently than expected. I compiled a loki client (See section "Program Identification" for details) and attempted to connect to the `atd` process running on the local system using the command: `lokid –d localhost` to verify that the unknown binary could communicate with the loki client. The client was compiled with XOR encryption enabled to match the encryption type found in the strings output of the unknown binary.

Figure 1-12: Connecting to atd process with loki client

```
sans:~/sans/L2# ./loki -d localhost

LOKI2   route [(c) 1997 guild corporation worldwide]
loki> /stat

lokid version:          2.0
remote interface:       127.0.0.1
active transport:       icmp
active cryptography:    XOR
server uptime:          46.58 minutes
client ID:              25899
packets written:        5
bytes written:          420
requests:               1
loki>
```

Figure 1-12 shows the communication between the `loki` client and the `atd` binary which was running as a daemon process on the system. I sent the LOKI protocol `/stat` command from the client to show the status of the current session. The `atd` binary replied and identified itself as `lokid` version 2.0, using XOR encryption and ICMP as a transport protocol.
Communicating with the unknown binary using the `loki` client is a strong indicator that the `atd` binary is version 2.0 of the LOKI daemon.

The LOKI daemon (LOKID2), as described in the Phrack 51 article, is "an information tunneling program". It has the capability of tunneling UNIX "shell commands inside of ICMP_ECHO/ICMP_ECHOREPLY and DNS namelookup query/reply traffic."
ICMP (Internet Control Message Protocol) traffic can be a critical part to the operation of every IP based network. It has several purposes including determining if a particular host is able to communicate on the network or

determining if routes to hosts exist. There are 15 ICMP message types, defined in RFC 792, which serve different purposes for network devices. LOKID2 implements communication between a server daemon and a client using ICMP types 0 and 8. ICMP type 0 (ICMP_ECHOREPLY) and ICMP type 8 (ICMP_ECHO). Under normal circumstances, the ping command sends an ICMP_ECHO to a host, which upon receiving it, responds with an ICMP_ECHOREPLY therefore showing it is connected and communicating on the network. The LOKID2 client, by default, uses the data portion in ICMP type 0 and 8 packets to transport shell commands, which are interpreted by the LOKID2 daemon, to the remote host running the daemon. It also has the ability to transport the covert channel communications in the data portion of a DNS reply packet. The covert channel established by LOKI can often pass traffic through firewalls undetected and gives the user of the client "back-door" access to the remote host as if they were logged on to that host with telnet/rlogin/ssh etc.

Program Identification

The evidence I gathered indicates that the unknown binary is the LOKI daemon and relies on the shared library `libc.so.5`, which is an older version of `libc`. This indicates that the binary was compiled on an older version of Linux. I downloaded the source code and documentation for LOKID2 at http://www.phrack.com/show.php?p=51&a=6.
The documentation clearly states that the LOKI 2 program is written for Linux kernel versions 2.0.x. This also indicates that the binary was compiled on an older version of Linux. Newer versions of Linux use the 2.2.x, or more recently, the 2.4.x kernels. I installed Slackware Linux 3.6 on the analysis station, which has kernel 2.0.35, libc 5.4.46 and includes GCC-2.7.2.3 as the bundled compiler. I also installed GCC version 2.7.2.1 to compile the LOKI daemon with same compiler version that compiled the `atd` binary.

I extracted the source code from the Phrack article using the `extract` utility provided by Phrack as shown below:

- Dowloaded gzipped Phrack issue 51, unzipped and un-tarred it. This created a directory named "phrack" with all the articles as separate files.
  `# gunzip –dc phrack51.tar,gz | tar xvf -`

- Copied and pasted the extract utility source code from the Phrack issue into the file `extract.c`. and compiled extract.c: `gcc –o extract extract.c`

- The LOKI 2 source code was contained in Article 6 of Phrack 51, so I extracted the source code from the Article 6 file using the extract utility.
  `# ./extract p51-6`
  This created a directory "`L2`" with the LOIKD2 source code in it.

I then looked at the `Makefile` included with the source code to see what compile options were available. LOKI 2 has the option of being compiled with various type of encryption to encrypt the network traffic. In addition the `Makefile` has the option of selecting to use popen() or open an pty to execute commands. Both XOR and the popen() option is also present in the `strings` output of the `atd` daemon in Figure 1-6, so I editied the `Makefile` as shown below to compile the source code with XOR encryption and using popen() to match the unknown binary.

```
CRYPTO_TYPE     =     WEAK_CRYPTO     #XOR
SPAWN_TYPE      =     POPEN
```

I then compiled the LOKI 2 binaries for Linux using the following command:

```
# /usr/bin/make linux
```

This compiles and strips the `lokid` (server daemon) and `loki` (client) binaries. The next step was to compare the binary that I compiled from source code with the unknown binary to confirm that the unknown binary is LOKI 2.
I checked the size, file type and MD5 checksum of the `lokid` binary to compare it against the `atd` binary. (Shown below in Figure 1-13)

Figure 1-13: Gathering basic info about lokid binary.

```
# ls –al lokid
-rw-rw-r—1 root other  15752  Jan 04 14:41 lokid

# file lokid
lokid: ELF 32-bit LSB executable, Intel 80386, version 1,
dynamically linked, stripped

#md5sum lokid
52aba5223634695a4332624d5815d01a lokid

# ldd lokid
      libc.so.5 => /lib/libc.so.5 (0x4000040000
```

The `lokid` binary is a dynamically linked, stripped ELF binary, which matches with the file type of `atd`, however, `lokid` is 15752 bytes is size, which is slightly larger than the `atd` binary. As expected the MD5 sum of `lokid` does not match the MD5 sum of the `atd` binary because of the file size difference.
The `ldd` output shows that lokid is only linked against the shared library libc.so.5 which is consistent with the `atd` binary.

I compared the strings output of the two binaries to determine if `atd` contained the same strings data as the lokid binary I compiled. The strings output of the two binaries was very similar. There were additional lines of binary data in the strings output of the `atd` binary, which I found by piping the strings output to `wc -l` to count the number of lines. (Shown in Figure 1-14)

Figure 1-14: Counting lines in strings output

```
# strings atd | wc -l
157
# strings lokid | wc -l
153
```

To further prove that the `atd` binary is the LOKID v2.0 daemon, I compared the output of `strace` that I performed on `atd` and `lokid`. using the `diff` command. `diff` compares the two files and outputs the lines (and line numbers) of the differences between the two file.
Figure 1-15 shown the steps taken to generate and compare the output, and the differences between the resulting files.
The comparison of these two files shows that there is no difference between the function calls executed by the `atd` and `lokid` processes.
The differences shown in the `diff` output in Figure 1-15 are all related to the fact that the two processes have different process id numbers (PIDs) and allocate different memory locations for themselves during execution.

Figure 1-15: Comparison of strace output

```
# strace -o atd-strace.out -ff ./atd
# strace -o lokid-strace.out -ff ./lokid
# diff atd-strace.out lokid-strace.out

1c1
< execve("./atd", ["./atd"], [/* 29 vars */]) = 0
---
> execve("./lokid", ["./lokid"], [/* 28 vars */]) = 0
4c4
< mprotect(0x8048000, 13604, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
---
> mprotect(0x8048000, 13956, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
9c9
< stat("/etc/ld.so.preload", 0xbffff9d4)  = -1 ENOENT (No such file or
directory)
---
> stat("/etc/ld.so.preload", 0xbffff9e8)  = -1 ENOENT (No such file or
directory)
20c20
< mprotect(0x8048000, 13604, PROT_READ|PROT_EXEC) = 0
---
> mprotect(0x8048000, 13956, PROT_READ|PROT_EXEC) = 0
```

```
30c30
< brk(0x804c820)                          = 0x804c820
---
> brk(0x804c9a0)                          = 0x804c9a0
32,36c32,36
< stat("/etc/locale/C/libc.cat", 0xbffff50c) = -1 ENOENT (No such file
or directory)
< stat("/usr/share/locale/C/libc.cat", 0xbffff50c) = -1 ENOENT (No such
file or directory)
< stat("/usr/share/locale/libc/C", 0xbffff50c) = -1 ENOENT (No such
file or directory)
< stat("/usr/share/locale/C/libc.cat", 0xbffff50c) = -1 ENOENT (No such
file or directory)
< stat("/usr/local/share/locale/C/libc.cat", 0xbffff50c) = -1 ENOENT
(No such file or directory)
---

> stat("/etc/locale/C/libc.cat", 0xbffff51c) = -1 ENOENT (No such file
or directory)
> stat("/usr/share/locale/C/libc.cat", 0xbffff51c) = -1 ENOENT (No such
file or directory)
> stat("/usr/share/locale/libc/C", 0xbffff51c) = -1 ENOENT (No such
file or directory)
> stat("/usr/share/locale/C/libc.cat", 0xbffff51c) = -1 ENOENT (No such
file or directory)
> stat("/usr/local/share/locale/C/libc.cat", 0xbffff51c) = -1 ENOENT
(No such file or directory)
38c38
< sigaction(SIGUSR1, {0x804a6b0, [],
SA_INTERRUPT|SA_NOMASK|SA_ONESHOT}, {SIG_DFL}) = 0
---
> sigaction(SIGUSR1, {0x804a810, [],
SA_INTERRUPT|SA_NOMASK|SA_ONESHOT}, {SIG_DFL}) = 0
41,45c41,45
< getpid()                                = 476
< getpid()                                = 476
< shmget(718, 240, IPC_CREAT|0)           = 13
< semget(900, 1, IPC_CREAT|0x180|0600)    = 12
< shmat(13, 0, 0)                         = 0x40008000
---
> getpid()                                = 3242
> getpid()                                = 3242
> shmget(3484, 240, IPC_CREAT|0)          = 16
> semget(3666, 1, IPC_CREAT|0x180|0600)   = 15
> shmat(16, 0, 0)                         = 0x40008000
47c47
< time([1043673566])                      = 1043673566
---
> time([1043777010])                      = 1043777010
52c52
< fork()                                  = 477
---
> fork()                                  = 3243
55c55
< semop(0xc, 0x2, 0, 0xbffff990)                          = 0
---
> semop(0xf, 0x2, 0, 0xbffff9a0)                           = 0
```

```
57c57
< semop(0xc, 0x1, 0, 0xbffff990) = 0
---
> semop(0xf, 0x1, 0, 0xbffff9a0) = 0
```

An analysis of the child processes also revealed that processes forked off by `atd` execute the same system calls as the processes forked by `lokid`. The system calls executed by the child processes of both daemons are shown in Figure 1-11.

The next step, after having proved that the binaries execute identical system calls, was to test if they had the same command line options and behave similarly when a client connects. I parsed the source code for `lokid` to determine the command line options. The snippet of source code that defines the command line arguments is shown in Figure 1-16.

Figure 1-16:lokid source code with command line args.

```
while ((c = getopt(argc, argv, "v:p:")) != EOF)
  {
     switch (c)
     {
        case 'v':                 /* change verbosity */
           verbose = atoi(optarg);
           break;

        case 'p':                 /* choose transport protocol */
           switch (optarg[0])
           {
              case 'i':          /* ICMP_ECHO / ICMP_ECHOREPLY */
                 prot = IPPROTO_ICMP;
                 break;

              case 'u':          /* DNS query / reply */
                 prot = IPPROTO_UDP;
                 break:
```

According to the source code, the `loki` daemon has two command line arguments.

- `-v (1|0)` - Specifying this option and the number 1 or 0 gives more or less verbose output when the daemon is running
- `-p` - Specifies which transport protocol to use for client/server communication
  - `-i` use ICMP (ECHO/ECHO_REPLY packets)
  - –u use UDP (DNS query reply packet)

I ran the `atd` binary with a command line option of "`-h`" to see if it printed out its command line options. (shown in Figure 1-17)

Figure 1-17: running atd with –h command line argument

```
#./atd -h
   ./atd: illegal option -- h
   lokid -p (i|u) [ -v (0|1) ]
```

The `atd` binary prints out the available command line arguments that are expected from the `lokid` binary as seen in the source code in Figure 1-16. It also states that the name of the program is `lokid` when displaying the command line arguments.

The `atd` and `lokid` binaries require that they be run as root. Root, also known as “Super User” is the highest privilege of user on a UNIX system and is used as an administrative account to run and control all system processes. The root account will always have a UID of 0. Line 50 of `lokid.c` contains the code below that checks if the UID and EUID of the user is 0.

```
if (geteuid() || getuid()) err_exit(0, 1, 1, L_MSG_NOPRIV);
```

The error message returned when a non-root user executes it is defined on line 239 the `loki.h` header file shown below.

```
#define L_MSG_NOPRIV "\n[fatal] invalid user identification
value"
```

This is an important point because the intruder would have needed root privileges to start the `atd` program. Both `atd` and `lokid` produced the same error when I attempted to run them as a non-root user (shown in Figure 1-18)

Figure 1-18: Attempting to run atd and lokid as non-root user

```
# ./atd

[fatal] invalid user identification value: Success
```

I then started the `atd` process using the command line arguments to enable verbose output (`-v`) and UDP transport (`-p u`), and connected to the daemon using the `loki` client. (See Figure 1-17). From the client, I sent the `/stat` LOKI protocol command to get a status of the connection. The `/stat` confirmed that the loki client was communicating with the `atd` process using UDP transport and XOR encryption.

Figure 1-17: Client connect to atd running w/UDP transport

```
Localhost:# ./atd –v 1 –p u
localhost:# loki -d localhost -v 1 -p u

LOKI2   route [(c) 1997 guild corporation worldwide]
loki> /stat

lokid version:          2.0
remote interface:       127.0.0.1
active transport:       udp
active cryptography:    XOR
server uptime:          1.58 minutes
client ID:              7977
packets written:        5
bytes written:          420
requests:               1
loki> /quit
```

To further determine if the `atd` and `lokid` processes behaved identically, I compared the output of `netstat` and `lsof` when each of the programs was running. The `netstat` command shows the status of network connections and listening sockets. The `lsof` utility lists open files and sockets that specific processes are accessing. By comparing the output of these utilities, I was able to determine that the `lokid` and `atd` processes both referenced and opened the same raw sockets as shown in Figure 1-18 and 1-19. Figure 1-18 shows the raw sockets both processes opened when running with ICMP as the transport protocol (using netstat). Figure 1-19 shows the raw sockets both processes opened when running with the UDP transport protocol (using netstat).

Figure 1-18: netstat output with atd and lokid running (ICMP)

```
# netstat -a
….
raw        0      0 *:255                   *:*
raw        0      0 *:1                     *:*
….
```

Figure 1-19: netstat output with atd and lokid running (UDP)

```
raw        0      0 *:17                    *:*
```

The output of the `lsof` command related to the `atd` and `lokid` binaries running (shown in Figures 1-20 and 1-21) also show that the two daemons execute identically and reference the same raw sockets.

Figure 1-20 lsof output with atd and lokid

```
lokid      3196   root  cwd    DIR        3,1   1024     77521
[0301]
lokid      3196   root  rtd    DIR        3,1   1024         2
[0301]
lokid      3196   root  txt    REG        3,1  15784     77561
[0301]
lokid      3196   root    1u   CHR        4,1              6358
[0301]
lokid      3196   root    2u   CHR        4,1              6358
[0301]
lokid      3196   root    3u   raw                       205621
00000000:0001->00000000:0000 st=07
lokid      3196   root    4u   raw                       205622
00000000:00FF->00000000:0000 st=07
```

Figure 1-21: lsof output with atd running.

```
atd        3085   root  cwd    DIR        3,1   1024     77521
[0301]
atd        3085   root  rtd    DIR        3,1   1024         2
[0301]
atd        3085   root  txt    REG        3,1  15348     48981
[0301]
atd        3085   root    1u   CHR        4,1              6358
[0301]
atd        3085   root    2u   CHR        4,1              6358
[0301]
atd        3085   root    3u   raw                       204804
00000000:0001->00000000:0000 st=07
atd        3085   root    4u   raw                       204805
00000000:00FF->00000000:0000 st=07
```

The ELF binary structure has several sections (See the document “Tool interface Standards, Portable Formats Specification, Ver 1.1. “Executable and Linking Format (ELF)” for specific details.) Every ELF binary begins with a header that contains information about the binary including the byte order, entry point, and location of section header table etc. (see Appendix 1-1 for specific header) The entry point is the virtual address in the binary where the system transfers control to upon execution of the binary. The section header contains the byte offset within the file of each of the internal sections. These sections each contain information for different functions within the ELF binary. The `.comments` section for example, contains information about the compiler used to compile the binary (See Figure 1-9). The `.dynsym` section contains information about dynamic linking objects, for example information about how to resolve references (to local or global functions). By comparing the contents of

the sections of the `atd` and `lokid` binaries, I was able to gather more evidence proving that the two binaries were functionally identical.
I ran the `readelf -a` command to dump the information stored in all sections both binaries. (The full output of `readelf -a atd` is shown in Appendix 1-1). Upon comparing the output of the `readelf` command on each of the two binaries, I determined that they contained the same sections, and the references stored in the all the sections were identical, proving that the binaries were functionally identical.
There were some notable differences between the two binaries shown in Figure 1-22 (not a complete list of differences, just representative of all the key differences)

Figure 1-22: Notable differences in `readelf` output

```
atd bianry:
     Entry point address: 0x8048db0
     Start of section headers: 14508 (bytes into file)
     Dynamic segment at offset 0x3644

Lokid binary:
     Entry point address: 0x8048d90
     Start of section headers: 14944 (bytes into file)
     Dynamic segment at offset 0x36bc
```

The entry points were slightly different, and the offset locations of the internal sections were at different locations within each of the files. Both of these are acceptable differences and don't negate the conclusion that the binaries are functionally identical based on the following facts:

- The Whirlwind Tutorial on Creating Really Teensy ELF Executables for Linux" document states "…..Almost any [entry] address can be used as long as it's above 0x00000000, below 0x80000000 and page aligned."
- The "Executable and Linking Format(ELF)" document states "sections and segments have no specified order. Only the ELF header has a fixed position in the file."

These differences in the binaries could have been caused by differences in the compiler, linker or assembler used to compile them both.

Program Identification Summary:

Through the previous analysis, I have determined that the `atd` binary that was retrieved from the compromised system is LOKID v2.0. I was unable to compile the LOKID v2.0 daemon from source code and match the file size and MD5 checksum of the `atd` binary. I installed the same version of the compiler that was used to compile the `atd` binary, and I was still unable to match my compiled

version to the file size. This slight difference in size may be attributed to compiling the source with a different version of the libc5 library than was used to compile `atd`.
Although I was unable to compile the LOKID source and match the file size of `atd`, I am able to prove that the `atd` binary behaves exactly as the `lokid` binary does in the following ways:

- The binaries are both 32-bit ELF executable compiled for the Intel 80386 architecture, dynamically linked, and stripped
- They are both only linked against the /lib/libc.so.5 shared library
- They contain nearly identical strings data.
- The system calls executed by the binaries are identical
- The system calls executed by the child processes of the binaries are identical.
- The binaries have the same command line arguments, and the atd process identifies itself as `lokid` when querying the command line arguments.
- Both the `atd` and `lokid` process must be run as root user (UID 0), and give the same error when run as non-privileged user.
- The loki client communicates with the `atd` daemon process with both ICMP and UDP as it were the loki daemon.
- The output of the netstat command shows identical open raw sockets.
- The output of the lsof command shows identical process and open socket information
- `readelf` shows identical information in all sections of the ELF binary (except for noted acceptable differences)

Legal Implications:

In my forensic analysis I was unable to get forensic information from the compromised system that would have shown that the unknown binary had been executed on that system. The process of zipping them up for my analysis altered the MAC times of the binary, which can sometimes be used to determine if/when the binary was executed. I was able to prove that the binary is the LOKID2 "covert channel" backdoor, which allows unauthorized access to the system and I was able to prove that a user needs root privileges to run the daemon.
Executing the binary on the compromised system, could have been a violation of the Computer Fraud and Abuse Act (18 U.S.C §1030), which criminalizes "unauthorized access" or "damage" to a "protected computer".
A protected computer is defined in §1030(e)(2) as computers:

- "Exclusively for the use of a financial institution or the United States Government"
- "used in interstate or foreign commerce or communication"

Essentially every computer connected to the Internet in the U.S. is considered a protected computer. The compromised system is not a U.S.

government system, so in order for the binary running on the compromised system to be considered a crime under §1030, the intruder would have had to have done one of the following:

- caused damage in excess of $5,000 dollars in a 1 year period (can be aggregated to include damage to other systems by same intruder),
- caused impairment of medical records
- caused physical injury to a person
- posed a threat to public safety

Other evidence would have to be gained to determine if the intruder was in violation of §1030, the presence of the binary on the system is not enough to prove it.
If the intruder was found to be in violation of §1030 one of the penalties listed in §1030 (c) would apply. The penalties range from 1 year imprisonment and a fine, all the way up to a fine and 20 years imprisonment depending on the severity and number of violations.

The compromised system resides at my employer's location in Massachusetts, therefore Massachusetts computer crime laws apply as well.
The individual who installed and ran the LOKI daemon binary would be in violation of Massachusetts General Law (M.G.L) 266 §120F, which states:
" Whoever, without authorization, knowingly accesses a computer system by any means, or after gaining access to a computer system by any means knows that such access is not authorized and fails to terminate such access, shall be punished by imprisonment in the house of correction for not more than thirty days or by a fine of not more than one thousand dollars, or both. The requirement of a password or other authentication to gain access shall constitute notice that access is limited to authorized users."

The intruder would also be in violation of M.G.L 226 §33A, which states:
"Whoever, with intent to defraud, obtains, or attempts to obtain, or aids or abets another in obtaining, any commercial computer service by false representation, false statement, unauthorized charging to the account of another, by installing or tampering with any facilities or equipment or by any other means, shall be punished by imprisonment in the house of correction for not more than two and one-half years or by a fine of not more than three thousand dollars, or both. As used in this section, the words ""commercial computer service'' shall mean the use of computers, computer systems, computer programs or computer networks, or the access to or copying of the data, where such use, access or copying is offered by the proprietor or operator of the computer, system, program, network or data to others on a subscription or other basis for monetary consideration."

Based on my research it would be easier to prove that the individual who installed the `atd` binary on the system was in violation of M.G.L 226 §33a and §120F than it would be to prove they were in violation of U.S.C §1030.

The `atd` binary allows unauthenticated, unauthorized remote access to the host computer. In addition, the user has to run the `atd` program as root, indicating that they exceeded their privileges on the system and would therefore be in violation of state law, but none of the criteria for a Federal violation can be proven.

Interview questions:

As part of my job, I occasionally interview students who have violated our Computer and Network Acceptable Use Policies as part of the University's disciplinary process. Experience from these interviews has taught me that people respond differently to the interview process and it's important to be prepared and well informed before the interview. Furthermore, it's important to have a plan for dealing with different scenarios that may come up during the course of the interview.
In the process of interviewing the individual who installed LOKID2 on the system, I would take the following approach:

I start the interview off by taking a moment to look over my notes and the printouts of evidence such as logfiles etc. The interviewees are not allowed to see the contents of the notes or evidence, but it gives them the sense that I have concrete evidence against them. I begin by asking simple questions, for example:
What type of computer is your personal system?,
What operating system are you running on your personal computer ?
These questions give me a sense of their level of knowledge and a sense of how they might react to the situation. Their answers to the simple questions help me gauge if they are going to be arrogant, talkative, defensive, combative or helpful and willing to cooperate. I take different approaches to the interviews depending on how they respond.

The next step I usually take is to ask an open-ended question such as, "*why do you think I have asked to speak with you today?*" Some interviewees know why they are being interviewed, think I have undisputable evidence against them, tell me everything, apologize and swear it will never happen again. Those interviews are easy, but they seldom go that way. When they claim to not know why they are being interviewed, I often take the approach of: *" I have some evidence here indicating that you have been involved in an incident on one of our computer systems and I'd like to speak with you to see if perhaps we've misinterpreted this incident. I appreciate any information you may be able to give me to help sort this out. Right now it looks pretty bad, but I'm hoping that we can resolve this issue here so it doesn't have to go before the Dean of Students Judicial Board or the University Police."*
This gives the interviewee the sense that it's bad, and I am there to help. It perhaps gives them a chance to explain to someone who is willing to listen and understand.

I would then proceed to throw out a little information that I know and give the interviewee a chance to explain. "*I found an interesting program in your home directory, could you please explain why it's there?*" The intention with that question is to see if the interviewee will tell you he/she owns the binary and if he/she ran the binary.
Another approach is to say "*I've found a backdoor program in your home directory, could you please not run it anymore, it's causing problems on the system.*" This sets the stage for the interviewee to confirm that he/she ran the backdoor program.
Some of the more arrogant interviewees have the need to prove their computer skills, so questions such as "*That back door was easy to spot, did you think we wouldn't notice? Is that the best you could do?*" Sometimes in defending their knowledge of computers they admit to running the program and may also tell why.
The key things to keep in mind when interviewing are that you're going to have to find the angle to coerce information from the subject. Sometimes it is a matter of acting understanding as an interviewer, or insist that they can help by telling you everything, and sometimes you can gain information by making the subject defensive. It varies depending on the interview subject and it's important to be prepared for whatever information may come out of the interview.

Additional information:

I used several online resources to research reverse engineering techniques and tools that I used to analyze the unknown binary. I found the following URLs, which describe reverse engineering techniques in detail, on the Honeynet Reverse Challenge page.
http://www.honeynet.org/reverse/sol/sol-06/analysis.html
http://www.honeynet.org/reverse/sol/sol-21/analysis.html

The following references give great insight into the format of the ELF binary.
Tool interface Standards, Portable Formats Specification, Ver 1.1. "Executable and Linking Format (ELF)"
http://www.skyfree.org/linux/references/ELF_FORMAT.pdf

Raiter, Brian. "A Whirlwind Tutorial on Creating Teensy ELF Executables for Linux" 21 Jan 2003
http://www.muppetlabs.com/~breadbox/software/tiny/teensy.html

See the References section below for a full listing of sources to gather more information.

References:

Card, Rémy. Ts'o, Theodore. Tweedie, Stephen. “Design and Implementation of the Second Extended Filesystem” http://e2fsprogs.sourceforge.net/ext2intro.html Dec 13 2002

daemon9 “Project Loki: ICMP tunneling” Phrack Magazine, Issue 49 Aug. 1996 http://www.phrack.com/show.php?p=49&a=6

daemon9 “LOKI2 (the implementation) Phrack Magazine, Issue 51 Sept. 1997 http://www.phrack.com/show.php?p=51&a=6

Fenris Homepage 19 Nov 2002 http://razor.bindview.com/tools/fenris/

M.G.L 226 §120F “Unauthorized Access to computer systems” http://www.state.ma.us/legis/laws/mgl/266-120F.htm

M.G.L 226 §33A “Attempt to defraud commercial computer service” http://www.state.ma.us/legis/laws/mgl/266-33A.htm

Owen, Greg. GCFA practical http://www.giac.org/practical/Greg_Owen_GCFA.zip

Pesch, Roland H., Osier, Jeffery M. and Cygnus Support "The gnu Binary Utilities" 29 Oct 2002 http://www.skyfree.org/linux/references/binutils.pdf

Postel, John. “RFC 792” Sept. 1981 http://www.ietf.org/rfc/rfc792.txt

Raiter, Brian. “A Whirlwind Tutorial on Creating Teensy ELF Executables for Linux” 21 Jan 2003 http://www.muppetlabs.com/~breadbox/software/tiny/teensy.html

Rekhter, Y., Moskowitz, B., Karrenberg, D., deGroot, G.J., Lear, E. “RFC 1918” Feb. 1996 http://www.ietf.org/rfc/rfc1918.txt

Rivest R., “RFC 1321” Apr. 1992 http://www.ietf.org/rfc/rfc1321.txt

Smith, Craig. “Academic Underground: The Examiner” 17 Nov 2002 http://www.academicunderground.org/examiner/

Tool Interface Standards, Portable Formats Specification, Ver 1.1 "Executable and Linking Format (ELF)" 30 Oct 2002
http://www.skyfree.org/linux/references/ELF_Format.pdf

18 U.S.C §1030
http://www4.law.cornell.edu/uscode/18/1030.html

Zalewski, Michal. 29 Nov 2002
http://lcamtuf.coredump.cx/fenris/reverse.txt

Appendix 1-1: readelf output of unknown binary

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x8048db0
  Start of program headers:          52 (bytes into file)
  Start of section headers:          14508 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         5
  Size of section headers:           40 (bytes)
  Number of section headers:         21
  Section header string table index: 20

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg
Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00
0   0  0
  [ 1] .interp           PROGBITS        080480d4 0000d4 000013 00   A
0   0  1
  [ 2] .hash             HASH            080480e8 0000e8 0001a4 04   A
3   0  4
  [ 3] .dynsym           DYNSYM          0804828c 00028c 000420 10   A
4   1  4
  [ 4] .dynstr           STRTAB          080486ac 0006ac 000210 00   A
0   0  1
  [ 5] .rel.bss          REL             080488bc 0008bc 000020 08   A
3  11  4
  [ 6] .rel.plt          REL             080488dc 0008dc 000190 08   A
3   8  4
  [ 7] .init             PROGBITS        08048a70 000a70 000008 00  AX
0   0 16
  [ 8] .plt              PROGBITS        08048a78 000a78 000330 04  AX
0   0  4
  [ 9] .text             PROGBITS        08048db0 000db0 001b28 00  AX
0   0 16
  [10] .fini             PROGBITS        0804a8e0 0028e0 000008 00  AX
0   0 16
  [11] .rodata           PROGBITS        0804a8e8 0028e8 000c3c 00   A
0   0  4
  [12] .data             PROGBITS        0804c528 003528 000038 00  WA
0   0  4
  [13] .ctors            PROGBITS        0804c560 003560 000008 00  WA
0   0  4
```

```
  [14] .dtors            PROGBITS        0804c568 003568 000008 00  WA
0   0  4
  [15] .got              PROGBITS        0804c570 003570 0000d4 04  WA
0   0  4
  [16] .dynamic          DYNAMIC         0804c644 003644 000088 08  WA
4   0  4
  [17] .bss              NOBITS          0804c6cc 0036cc 00012c 00  WA
0   0  8
  [18] .comment          PROGBITS        00000000 0036cc 0000a0 00
0   0  1
  [19] .note             NOTE            000000a0 00376c 0000a0 00
0   0  1
  [20] .shstrtab         STRTAB          00000000 00380c 0000a0 00
0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor
specific)

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg
Align
  PHDR           0x000034 0x08048034 0x08048034 0x000a0 0x000a0 R E 0x4
  INTERP         0x0000d4 0x080480d4 0x080480d4 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.1]
  LOAD           0x000000 0x08048000 0x08048000 0x03524 0x03524 R E
0x1000
  LOAD           0x003528 0x0804c528 0x0804c528 0x001a4 0x002d0 RW
0x1000
  DYNAMIC        0x003644 0x0804c644 0x0804c644 0x00088 0x00088 RW  0x4

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .hash .dynsym .dynstr .rel.bss .rel.plt .init .plt
.text .fini .rodata
   03     .data .ctors .dtors .got .dynamic .bss
   04     .dynamic

Dynamic segment at offset 0x3644 contains 17 entries:
  Tag        Type                         Name/Value
 0x00000001 (NEEDED)                     Shared library: [libc.so.5]
 0x0000000c (INIT)                       0x8048a70
 0x0000000d (FINI)                       0x804a8e0
 0x00000004 (HASH)                       0x80480e8
 0x00000005 (STRTAB)                     0x80486ac
 0x00000006 (SYMTAB)                     0x804828c
 0x0000000a (STRSZ)                      528 (bytes)
 0x0000000b (SYMENT)                     16 (bytes)
 0x00000015 (DEBUG)                      0x0
 0x00000003 (PLTGOT)                     0x804c570
 0x00000002 (PLTRELSZ)                   400 (bytes)
 0x00000014 (PLTREL)                     REL
 0x00000017 (JMPREL)                     0x80488dc
 0x00000011 (REL)                        0x80488bc
```

```
 0x00000012 (RELSZ)                      32 (bytes)
 0x00000013 (RELENT)                     8 (bytes)
 0x00000000 (NULL)                       0x0

Relocation section '.rel.bss' at offset 0x8bc contains 4 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
0804c6d8  00001005 R_386_COPY        0804c6d8   _IO_stderr_
0804c72c  00001405 R_386_COPY        0804c72c   optarg
0804c730  00002205 R_386_COPY        0804c730   __fpu_control
0804c6d0  00003d05 R_386_COPY        0804c6d0   _errno

Relocation section '.rel.plt' at offset 0x8dc contains 50 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
0804c57c  00000107 R_386_JUMP_SLOT   08048a88   longjmp
0804c580  00000207 R_386_JUMP_SLOT   08048a98   strcpy
0804c584  00000307 R_386_JUMP_SLOT   08048aa8   ioctl
0804c588  00000407 R_386_JUMP_SLOT   08048ab8   popen
0804c58c  00000507 R_386_JUMP_SLOT   08048ac8   shmctl
0804c590  00000607 R_386_JUMP_SLOT   08048ad8   geteuid
0804c594  00000807 R_386_JUMP_SLOT   08048ae8   getprotobynumber
0804c598  00000a07 R_386_JUMP_SLOT   08048af8   __strtol_internal
0804c59c  00000b07 R_386_JUMP_SLOT   08048b08   usleep
0804c5a0  00000c07 R_386_JUMP_SLOT   08048b18   semget
0804c5a4  00000d07 R_386_JUMP_SLOT   08048b28   getpid
0804c5a8  00000e07 R_386_JUMP_SLOT   08048b38   fgets
0804c5ac  00000f07 R_386_JUMP_SLOT   08048b48   shmat
0804c5b0  00001107 R_386_JUMP_SLOT   08048b58   perror
0804c5b4  00001207 R_386_JUMP_SLOT   08048b68   getuid
0804c5b8  00001307 R_386_JUMP_SLOT   08048b78   semctl
0804c5bc  00001507 R_386_JUMP_SLOT   08048b88   socket
0804c5c0  00001707 R_386_JUMP_SLOT   08048b98   bzero
0804c5c4  00001907 R_386_JUMP_SLOT   08048ba8   alarm
0804c5c8  00001a07 R_386_JUMP_SLOT   08048bb8   __libc_init
0804c5cc  00001c07 R_386_JUMP_SLOT   08048bc8   fprintf
0804c5d0  00001d07 R_386_JUMP_SLOT   08048bd8   kill
0804c5d4  00001e07 R_386_JUMP_SLOT   08048be8   inet_addr
0804c5d8  00001f07 R_386_JUMP_SLOT   08048bf8   chdir
0804c5dc  00002007 R_386_JUMP_SLOT   08048c08   shmdt
0804c5e0  00002107 R_386_JUMP_SLOT   08048c18   setsockopt
0804c5e4  00002307 R_386_JUMP_SLOT   08048c28   shmget
0804c5e8  00002407 R_386_JUMP_SLOT   08048c38   wait
0804c5ec  00002507 R_386_JUMP_SLOT   08048c48   umask
0804c5f0  00002607 R_386_JUMP_SLOT   08048c58   signal
0804c5f4  00002707 R_386_JUMP_SLOT   08048c68   read
0804c5f8  00002807 R_386_JUMP_SLOT   08048c78   strncmp
0804c5fc  00002907 R_386_JUMP_SLOT   08048c88   sendto
0804c600  00002a07 R_386_JUMP_SLOT   08048c98   bcopy
0804c604  00002b07 R_386_JUMP_SLOT   08048ca8   fork
0804c608  00002c07 R_386_JUMP_SLOT   08048cb8   strdup
0804c60c  00002d07 R_386_JUMP_SLOT   08048cc8   getopt
0804c610  00002e07 R_386_JUMP_SLOT   08048cd8   inet_ntoa
0804c614  00002f07 R_386_JUMP_SLOT   08048ce8   getppid
0804c618  00003007 R_386_JUMP_SLOT   08048cf8   time
0804c61c  00003107 R_386_JUMP_SLOT   08048d08   gethostbyname
0804c620  00003307 R_386_JUMP_SLOT   08048d18   sprintf
0804c624  00003407 R_386_JUMP_SLOT   08048d28   difftime
0804c628  00003507 R_386_JUMP_SLOT   08048d38   atexit
```

```
0804c62c  00003707 R_386_JUMP_SLOT   08048d48   semop
0804c630  00003807 R_386_JUMP_SLOT   08048d58   exit
0804c634  00003907 R_386_JUMP_SLOT   08048d68   __setfpucw
0804c638  00003a07 R_386_JUMP_SLOT   08048d78   open
0804c63c  00003b07 R_386_JUMP_SLOT   08048d88   setsid
0804c640  00003c07 R_386_JUMP_SLOT   08048d98   close

There are no unwind sections in this file.

Symbol table '.dynsym' contains 66 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 08048a88     0 FUNC    GLOBAL DEFAULT  UND longjmp
     2: 08048a98    30 FUNC    GLOBAL DEFAULT  UND strcpy
     3: 08048aa8     0 FUNC    WEAK   DEFAULT  UND ioctl
     4: 08048ab8     0 FUNC    WEAK   DEFAULT  UND popen
     5: 08048ac8    42 FUNC    GLOBAL DEFAULT  UND shmctl
     6: 08048ad8     0 FUNC    WEAK   DEFAULT  UND geteuid
     7: 0804c644     0 OBJECT  GLOBAL DEFAULT  ABS _DYNAMIC
     8: 08048ae8   292 FUNC    GLOBAL DEFAULT  UND getprotobynumber
     9: 0804c6d0     4 NOTYPE  WEAK   DEFAULT   17 errno
    10: 08048af8  1132 FUNC    GLOBAL DEFAULT  UND __strtol_internal
    11: 08048b08    99 FUNC    GLOBAL DEFAULT  UND usleep
    12: 08048b18    42 FUNC    GLOBAL DEFAULT  UND semget
    13: 08048b28     0 FUNC    WEAK   DEFAULT  UND getpid
    14: 08048b38     0 FUNC    WEAK   DEFAULT  UND fgets
    15: 08048b48    59 FUNC    GLOBAL DEFAULT  UND shmat
    16: 0804c6d8    84 OBJECT  GLOBAL DEFAULT   17 _IO_stderr_
    17: 08048b58     0 FUNC    WEAK   DEFAULT  UND perror
    18: 08048b68     0 FUNC    WEAK   DEFAULT  UND getuid
    19: 08048b78    47 FUNC    GLOBAL DEFAULT  UND semctl
    20: 0804c72c     4 OBJECT  GLOBAL DEFAULT   17 optarg
    21: 08048b88    94 FUNC    WEAK   DEFAULT  UND socket
    22: 0804c528     4 OBJECT  GLOBAL DEFAULT   12 __environ
    23: 08048b98    54 FUNC    GLOBAL DEFAULT  UND bzero
    24: 08048a70     0 FUNC    GLOBAL DEFAULT    7 _init
    25: 08048ba8     0 FUNC    WEAK   DEFAULT  UND alarm
    26: 08048bb8    70 FUNC    GLOBAL DEFAULT  UND __libc_init
    27: 0804c528     4 NOTYPE  WEAK   DEFAULT   12 environ
    28: 08048bc8     0 FUNC    WEAK   DEFAULT  UND fprintf
    29: 08048bd8     0 FUNC    WEAK   DEFAULT  UND kill
    30: 08048be8    57 FUNC    GLOBAL DEFAULT  UND inet_addr
    31: 08048bf8     0 FUNC    WEAK   DEFAULT  UND chdir
    32: 08048c08    36 FUNC    GLOBAL DEFAULT  UND shmdt
    33: 08048c18   111 FUNC    WEAK   DEFAULT  UND setsockopt
    34: 0804c730     2 OBJECT  GLOBAL DEFAULT   17 __fpu_control
    35: 08048c28    42 FUNC    GLOBAL DEFAULT  UND shmget
    36: 08048c38     0 FUNC    WEAK   DEFAULT  UND wait
    37: 08048c48     0 FUNC    WEAK   DEFAULT  UND umask
    38: 08048c58    84 FUNC    GLOBAL DEFAULT  UND signal
    39: 08048c68     0 FUNC    WEAK   DEFAULT  UND read
    40: 08048c78    38 FUNC    GLOBAL DEFAULT  UND strncmp
    41: 08048c88   124 FUNC    WEAK   DEFAULT  UND sendto
    42: 08048c98   146 FUNC    GLOBAL DEFAULT  UND bcopy
    43: 08048ca8     0 FUNC    WEAK   DEFAULT  UND fork
    44: 08048cb8    79 FUNC    GLOBAL DEFAULT  UND strdup
    45: 08048cc8    44 FUNC    GLOBAL DEFAULT  UND getopt
```

```
    46: 08048cd8    67 FUNC    GLOBAL DEFAULT  UND inet_ntoa
    47: 08048ce8     0 FUNC    WEAK   DEFAULT  UND getppid
    48: 08048cf8     0 FUNC    WEAK   DEFAULT  UND time
    49: 08048d08   292 FUNC    GLOBAL DEFAULT  UND gethostbyname
    50: 0804a8e0     0 FUNC    GLOBAL DEFAULT   10 _fini
    51: 08048d18    38 FUNC    WEAK   DEFAULT  UND sprintf
    52: 08048d28    16 FUNC    GLOBAL DEFAULT  UND difftime
    53: 08048d38    52 FUNC    GLOBAL DEFAULT  UND atexit
    54: 0804c570     0 OBJECT  GLOBAL DEFAULT  ABS
_GLOBAL_OFFSET_TABLE_
    55: 08048d48    42 FUNC    GLOBAL DEFAULT  UND semop
    56: 08048d58   128 FUNC    GLOBAL DEFAULT  UND exit
    57: 08048d68    62 FUNC    GLOBAL DEFAULT  UND __setfpucw
    58: 08048d78     0 FUNC    WEAK   DEFAULT  UND open
    59: 08048d88     0 FUNC    WEAK   DEFAULT  UND setsid
    60: 08048d98     0 FUNC    WEAK   DEFAULT  UND close
    61: 0804c6d0     4 OBJECT  GLOBAL DEFAULT   17 _errno
    62: 0804a8d8     0 OBJECT  GLOBAL DEFAULT  ABS _etext
    63: 0804c6cc     0 OBJECT  GLOBAL DEFAULT  ABS _edata
    64: 0804c6cc     0 OBJECT  GLOBAL DEFAULT  ABS __bss_start
    65: 0804c7f8     0 OBJECT  GLOBAL DEFAULT  ABS _end

Histogram for bucket list length (total of 37 buckets):
 Length  Number     % of total  Coverage
      0  9          ( 24.3%)
      1  8          ( 21.6%)     12.3%
      2  10         ( 27.0%)     43.1%
      3  4          ( 10.8%)     61.5%
      4  5          ( 13.5%)     92.3%
      5  1          (  2.7%)    100.0%

No version information found in this file.
```

Forensic Analysis of Compromised Linux Host

SANS GCFA Practical Assignment v.1.2
Part 2 - Option 1

Jacob Cunningham

Table of Contents

Syntax Conventions

The text of the document is in 12 point Arial font

```
Commands executed at the shell, the output of commands, and
references to files, directories or system binaries are all
in 10 or 12 point Courier New font.
```

Part 2 – Option 1: Perform Forensic Analysis of a System

Introduction

This section of the paper is a write-up of the results of an in depth analysis of a Linux system that has been compromised. This analysis focuses on using forensic techniques to image the evidence media, and recovering evidence of the intrusion using the TASK and Autopsy forensic tools. Some familiarity with UNIX on the part of the reader is assumed.

Synopsis of Case Facts

The University's network consists of approximately 25,000 nodes connected to the Internet via 355mb/s leased circuit. The residential network accounts for about 50% of the total nodes connected to the campus network. It's somewhat rare for us to get contacted by students who think their systems have been compromised. We, in the networking department often find the compromised systems first when they start scanning for vulnerabilities or attempt a denial of service attack.

A user of the University's residential network contacted me on Oct 10, 2002 because he noticed user accounts on his personal Linux system that he had not created and suspected it had been hacked. The user who contacted me was not very computer savvy, but was observant enough to realize that something odd was happening on his system.

Upon finding the system may have been hacked, the user shut the system down and contacted me. If given a choice I would have preferred to gather some vital information about open network ports, running processes, and the contents of memory on the system before the user shut the system down, but there was still a plethora of forensic evidence to be gathered from the hard disk. I was concerned that the shutdown process may have been trojaned to cover the intruder's tracks.

After speaking to me, the user brought his computer system to my office where I inventoried it, tagged all the components as evidence (See Figure 2-1) and removed the hard disk for imaging.

Figure 2-1: Evidence listing and description

Evidence Tag #	Description
2002-10-10-1	"Home Built" Generic ATX PC 500 Mhz computer system (no case serial number) with Maxtor 20GB internal hard drive, 3.5" floppy drive, Mitsumi CDROM drive, 3Com 3c905b PCI Ethernet card, ATI Mach 64 PCI video card.
2002-10-10-2	Maxtor 541DX 20GB Ultra ATA/100 – 5400RPM Hard Drive S/N 2B020H1110511
2002-10-10-3	Mitsumi CRMC-FX810S CROM Drive S/N: DPU010136
2002-10-10-4	3Com 3c509b Ethernet card - MAC: 00:10:5a:e5:b4:fa
2002-10-10-5	ATI Mach 64 video card
2002-10-10-6	3.5" floppy

The potentially compromised system is a "home built" generic ATX 500MHZ Pentium II PC that was running RedHat Linux 7.0. The user explained that he set the system up to learn more about Linux and do some programming for a class he was taking. He also said he had installed RedHat Linux 7.0 on it, configured it to obtain an IP address via DHCP, and had it connected to the University Ethernet network in his room for only a few days before finding the unknown accounts and suspecting it was compromised. During the course of my investigation I discovered it was a default installation of RedHat 7.0 using kernel version 2.2.16-22. The system had been running several default exploitable daemons and services such as FTP (wu-ftpd-2.6.1) , Telnet, rsh, rlogin, portmap and statd.

Imaging the Evidence Disk

The system I created to do the forensic analysis is a PC running RedHat Linux 7.3. I performed a fresh install of the OS on a pristine disk to ensure the security and integrity of the OS before imaging and analyzing the evidence disk. This system has never been connected to the network, and to further ensure the security of the forensic system, no network card was installed. All software was transferred to the system via a known-clean CDROM.
The forensics system has an internal hard disk, a CDROM drive and two removable hard drive bays in the following configuration:

hda – boot disk (primary IDE Master)
hdb – CDROM (primary IDE slave)
hdc – removable drive bay (secondary IDE Master)
hdd – removable drive bay (secondary IDE slave)

The two removable hard drive bays attached to the secondary IDE controller are used to hold the image storage and evidence drives.
I performed the following tasks on the forensic system to create an image of the compromised system for analysis.

- Verified that hdc and hdd did not appear in `/etc/fstab` on the forensic system, so evidence and storage disks wouldn't be mounted at boot time.

- Placed a single partition 40gb drive with an ext2 filesystem in hdc removable drive bay. This hard drive was then sanitized using the command: `# dd if=/dev/zero of=/dev/hdc1`

- Set the evidence disk jumper to be a slave and placed the drive in the hdd removable drive bay. I had already confirmed the filesystem would not be mounted, therefore minimizing the risk of it getting altered.

- Booted the forensic system with all the disks installed.

- Generated an MD5 signature of the evidence partitions to compare against the MD5 signature of the resulting images. `md5sum` calculates a hash of binary data using the MD5 hashing algorithm (as described in RFC 1321) . This MD5 hash or checksum of the data is unique only to that data. If the data is modified, the MD5 calculation will not match the previous one. For all practical purposes it is impossible for two different pieces of data or files to have the same MD5 checksum. Consistently reproducing the same MD5 checksum for the same piece of data proves that the data has not been modified. It is important to create the md5 checksum before performing any operations on the evidence. Verifying matching MD5 checksums prove that the operations performed to image the media did not alter the original evidence in any way.

```
# md5sum /dev/hdd6
691560c798eb212ec5e750af5753c788    /dev/hdd6
# md5sum /dev/hdd5
835ba3f211ede3e529634997aafc7afe    /dev/hdd5
```

- Mounted the sanitized image storage disk. This makes the `hdc1` evidence storage partition available to the filesystem in the directory `/images`.
  `# mount /dev/hdc1 /images`

- Obtained a partition listing of evidence disk using `/sbin/fdisk`. This command provides information about the layout and type of the disk partitions without altering the contents of the disk. The evidence disk is a 20GB Maxtor hard drive (show in Figure 2-2 as hdd) with one large partition for the OS and a smaller one for swap space.

Figure 2-2: fdisk output

```
# fdisk -l /dev/hdd
      Disk /dev/hdd: 255 heads, 63 sectors, 2491 cylinders
      Units = cylinders of 16065 * 512 bytes

      Device Boot    Start       End    Blocks   Id  System
      /dev/hdd2          4      2491  19984860    5  Extended
      /dev/hdd5       2338      2370    265041   82  Linux swap
      /dev/hdd6          4      2337  18747792   83  Linux
```

- Imaged main partition and swap space of the evidence disk to a file on storage disk using `dd`. `dd` is a Unix command used to copy or convert files from one location or device to another.
  ```
  # dd if=/dev/hdd6 of=/images/sans-hdd6.img
  # dd if=/dev/hdd5 of=/images/sans-hdd5-swap.img
  ```

- Generated an MD5 signature of evidence image file and compared the hash of the image against the original. The md5 checksums match proving the image is identical to the original partition.
  ```
  # md5sum /images/sans-hda6.img
  691560c798eb212ec5e750af5753c788  sans-hda6.img

  # md5sum /images/sans-hdd5-swap.img
  835ba3f211ede3e529634997aafc7afe  sans-hdd5-swap.img
  ```

- I shutdown the forensics system and removed the evidence disk to minimize chance of corrupting or writing to the original evidence disk and then rebooted the forensics system to begin the analysis.

To protect the evidence hard drive, I placed it in an anti-static bag, and tagged the outside of the bag with the same evidence number that was on the drive (2002-10-10-2) with my signature and the date. I then locked the hard drive in my media safe, which is only accessible to my department's security officer and myself, and signed and dated the log book which is used for tracking all the items that are housed in the safe. The security officer and I have a clearly established protocol for adding and removing items from the safe, which includes tracking the location of the item in the log book. Maintaining this chain of custody ensured that the evidence disk did not get tampered with and its location was always known. This is one important piece of maintaining the integrity of the original evidence if it is questioned in a court of law.

Analyzing the Image with TASK and Autopsy

For the analysis of this potentially compromised system, I downloaded and compiled the latest version of forensics tools that I prefer to use.

Autopsy Forensic Browser v1.62
http://www.atstake.com/research/tools/autopsy/

TASK – The @Stake Sleuth Kit v1.52
http://www.atstake.com/research/tools/task/

The Coroner's Toolkit (TCT) is a package used for analyzing filesystems, file system images and recovering deleted files. Although TASK (The @Stake Sleuth Kit) is built upon TCT, I prefer to use TASK because of the added functionality it has over TCT. Specifically it has support for analyzing the Windows filesystems (FAT, NTFS) as well as UFS and ext2, this gives the examiner the advantage of using the same tools regardless of the filesystem type being analyzed. The Autopsy Forensic Browser is an HTML based front-end which employs the web browser as an interface to the TASK tools.

The evidence system disk had been formatted with an ext2 filesystems, which is common for that version of Linux. In an ext2 filesystem, files on the disk are described by inodes and the data is stored on the disk in data blocks. A general overview of inodes is best described by the document "Design and Implementation of the Second Extended Filesystem" found at
http://e2fsprogs.sourceforge.net/ext2intro.html
This document states:
"Each file is represented by a structure, called an inode. Each inode contains the description of the file: file type, access rights, owners, timestamps, size, pointers to data blocks. The addresses of data blocks allocated to a file are stored in its inode. When a user requests an I/O operation on the file, the kernel code converts the current offset to a block number, uses this number as an index in the block addresses table and reads or writes the physical block."

The tools contained in TASK can be used to view inode information for existing and deleted files and extract information from the physical disk blocks containing those existing and previously deleted files. This gives the examiner the ability to effectively un-delete and view the contents of any file and all of its attributes.

The first step in preparing the image for analysis was to mount the image to a mount point on the filesystem. To protect from modifying the evidence image and analysis system, I passed specific options to the mount command using the `-o` flag (See Figure 2-3)

`ro` – mount the image read-only. This disallows anything from writing to the image.

`loop` – mounts the image as a loopback device, which allows the image to be mounted as a filesystem.

`nodev` – prevents character and block devices in the image from being treated as devices by the analysis system.
`noexec` - disallows the execution of any binaries in the image filesystem on the analysis system.

Figure 2-3: Mounting the image

```
mkdir /mnt/image
mount –o ro,loop,nodev,noexec /images/sans-hda6.img
  /mnt/image
```

The next step was to edit the `fsmorgue` file in the Autopsy working directory to specify the name of the image, what type of filesystem it is, where it was mounted, and the time zone the imaged system was configured to use. This prepares Autopsy for the type of image it is analyzing.

Figure 2-4: fsmorge file

```
# fsmorgue file for Autopsy Forensic Browser
sans-hda6.img    linux-ext2     /       EST5EDT
```

I then started Autopsy with the following command.

```
# usr/local/src/autopsy-1.62/autopsy –m /images 888
localhost
```

This starts autopsy listening on port 888 of the localhost and uses /images as the morgue directory where it stores the `body` and `timeline` files. The autopsy sessions was then accessed by pointing a browser at the URL autopsy generated on the localhost. Although the Autopsy process is operating on a network port (888/tcp), it is inaccessible to anyone else because there is no network card in the system.

After Autopsy was running and reading the image, the next step was to first create a timeline to see what had changed on the system since the OS was installed. The user claimed to have installed the operating system a few days before noticing the intrusion, but it was necessary to verify this and other information that the user had given me. By verifying this information, such as OS version and the installation time, I could then generate a timeline to see when the alleged intrusion occurred.

I created the timeline using the “Timeline” button on the Main Menu in Autopsy. The timeline created by Autopsy gives a chronological visual representation of when all the files on a given filesystem were modified, accessed or changed. The Timeline function of Autopsy executes the following utilities from TASK behind the scenes:

- The fls utility collects the MAC times (the date and time the file was last Modified, Accessed or Changed) of allocated and un-allocated files and writes it to the "body" file with the command:

```
# fls –m / -f linux-ext2 –r sans-hda6.img > body
```

- Ils is then used to generate MAC time information for deleted files and appends that information to the "body" file using the command:

```
# ils –m sans-hda6.img >> body
```

In Autopsy, I then selected the "Create timeline using body" function. When selecting this, Autopsy gives you the option of inserting the inode location of the /etc/password and /etc/group files so the timeline contains user/group information for files rather than just numerical Ids. I parsed the "body" file using `/usr/bin/grep` to locate the inodes of the /etc/password and /etc/shadow files.

Figure 2-5: Retrieving inode numbers for /etc/passwd, /etc/group from body file

```
# cat body | grep "/etc/passwd"
0|/etc/passwd|0|195932|33188|-/-rw-r—r--|1|0|0|0|939
|1034194213|1034032865|1034032865|4096|0

# cat body | grep "/etc/group"
0|/etc/group|0|195773|33152|-/-rw------1|0|0|0|492
|1034032812|1033944837|1034032813|4096|0
```

The "Specify" option was used to specify the starting time and ending time of the timeline. The user claimed to have installed the operating system on Oct 6th, and the intrusion was noticed on Oct 9th, so Oct 1st was used as the start date and the current date of Oct. 17th was used as the end of the timeline. I then selected the timezone of the imaged system ("EST") to use in Autopsy and pressed the "Create" button.

Autopsy starts the TASK `mactime` utility which uses the body, password and group files plus the given date range to create the chronological timeline file as shown:

```
#  mactime –p 195932  –g 195773  –b /images/body
10/01/2002-10/17/2002 > timeline
```

The resulting "`timeline`" file, which is a chronological listing of all the files modified, accessed and changed on the image has the following format:

```
Date/time,  size (in bytes),  mac (specifies which file
attribute M,A,C changed), file permissions, owner, group
owner, inode number, filename
```

The example timeline entry in Figure 2-6 shows that the file /tmp/install.log: was 9829 bytes in size, was accessed on Oct 6th at 10:55:51, had UNIX file permissions "rw-r-r—" owned by user/group root/root, was referenced by inode number 162882

Figure 2-6: Example Timeline entry

```
Sun Oct 6 2002 10:56:44
    9829 .a. -/-rw-r--r-- root/boby root     162882   /tmp/install.log
```

Before looking over the timeline in detail, I first used the "File Browsing" function of Autopsy. The "File Browsing" function of Autopsy allows the examiner to navigate the disk image as if it were a hierarchical filesystem. I was able to gather information and view the contents of specific files. My intent in browsing the filesystemn was to confirm the information the user had given me about OS version and when the OS was installed, and to gather additional information about the system that might help in the investigation. To gather this information, I examined the following files on the evidence image:

- `/etc/issue` – From this file I was able to verify the OS version given to me by the user was correct. It contained:

  ```
  Red Hat Linux release 7.0 (Guinness)
  Kernel 2.2.16-22 on an i686
  ```

- `/tmp/install.log` – This file is written when the OS is installed. I checked the MAC times of this file and verified the operating system was installed on Oct 6th as told to me by the user.

  M: 2002.10.06 11:10:26 (EDT)
  A: 2002.10.06 10:56:44 (EDT)
  C: 2002.10.06 11:10:26 (EDT)

- `/var/log/boot.log` – This file contains information about when the operating system is booted and occasionally contains information about daemon processes. I noticed odd entries in this file about `portmap` and `ssh` daemon shutdowns. I noted the time and date that the anomalous events occurred as possible clues to the intrusion.

  ```
  Oct 7 19:18:52 localhost portmap: portmap
  shutdown succeeded
  Oct 7 19:19:10 localhost sshd: sshd shutdown
  succeeded
  ```

- `/etc/passwd,/etc/shadow` – This file contains user account information for the system. The owner of the system had said that he noticed a user "bobby" had been added to the system, so I checked these files and confirmed the users "bobby" and "boby" (with UID 0) exist. I also took note of the MAC times on the files as additional data points for the investigation. These files were modified approximately 2 minutes after the strange ssh and portmap daemon entries in `/var/log/boot.log`

  /etc/passwd:
  M: 2002.10.07 19:21:05 (EDT)
  A: 2002.10.09 16:10:13 (EDT)
  C: 2002.10.07 19:21:05 (EDT)

  Contents:
  ```
  bobby:x:501:501::/home/bobby:/bin/bash
  boby:x:0:0::/root:/bin/bash
  ```

  /etc/shadow:
  M: 2002.10.07 19:21:05 (EDT)
  A: 2002.10.09 13:26:23 (EDT)
  C: 2002.10.07 19:21:05 (EDT)

  Contents:
  ```
  bobby:$1$.mlvYnX4$bphcZdcVh8ONBKeM8XrGw0:11967:0:
  99999:7:::
  boby:$1$he7ZnLoq$/zZydt8zv4ddZs18dMYI2/:11967:0:
  99999:7:::
  ```

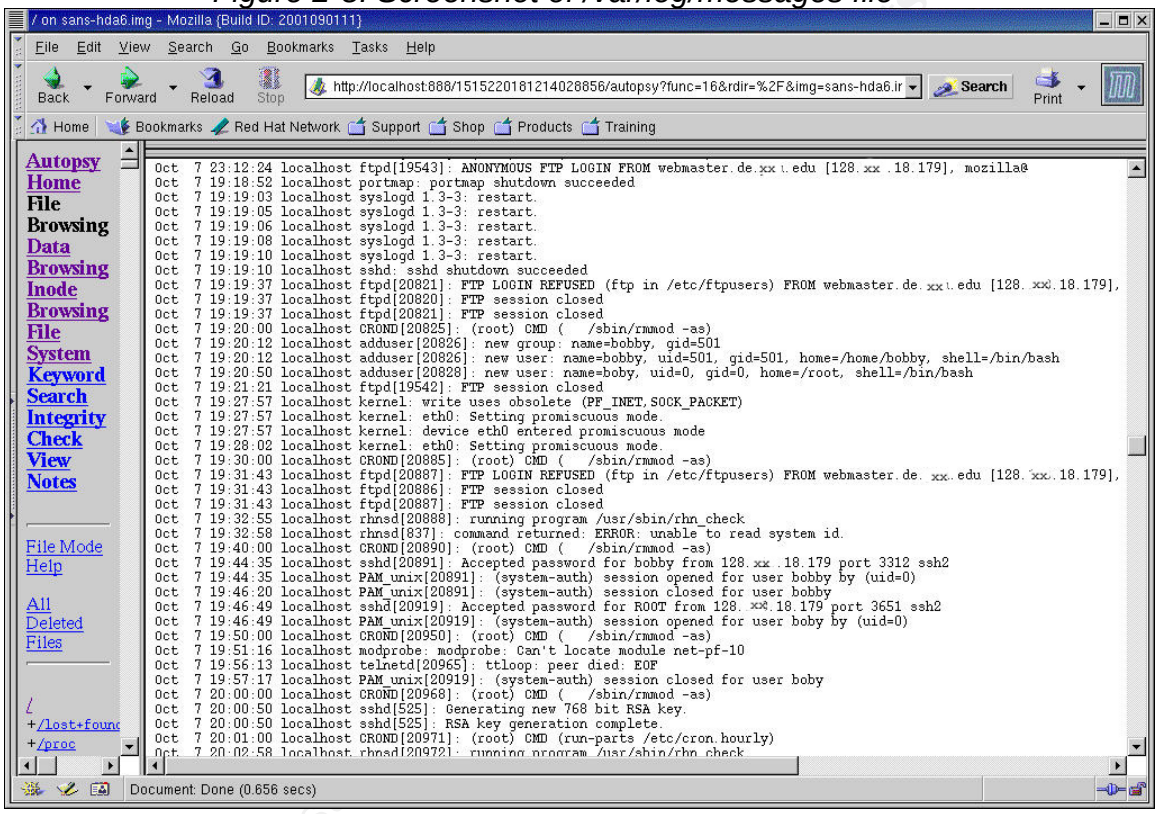
- `/var/log/wtmp` – This is a binary file that contains user login session data. I extracted the file using the "Export Contents" function in Autopsy and viewed its contents using the `/usr/bin/last` command. It shows users "bobby", "boby" and an anonymous FTP session logging in from webmaster.de.xxx.edu. The anonymous FTP login indicated that the intruder potentially gained access using a wu-ftpd exploit.

Figure 2-7: output of last command

```
# last -f sans-hda6.img-var.var.log.wtmp.raw
boby pts/2 webmaster.de.xxx.edu Mon Oct 7 19:46 -
19:57  (00:10)
bobby pts/2 webmaster.de.xxx.edu Mon Oct 7 19:44 -
19:46  (00:01)
ftp ftpd19543 webmaster.de.xxx.edu Mon Oct 7 19:12 -
down  (1+20:57)
```

- `/var/log/messages` - Contains various logging information from the system and daemons. It is often a significant source of information about system events. Intruders will sometimes modify or delete this file to cover their tracks. It turned out to contain information that was key to the investigation. Figure 2-8 contains a screen shot of a portion of the file containing the most usable information.

Figure 2-8: Screenshot of /var/log/messages file



The key items shown in Figure 2-8 are the events that happened between 19:12:24 and 19:57:17. In that time frame the system log had recorded the following suspicious events:

- An anonymous FTP login from webmaster.de.xxx.edu (also seen in wtmp file)
- `Portmap` daemon shutdown
- `Syslogd` restarted several times in quick succession
- `sshd` shutdown (also noted in `/var/log/boot.log` above)
- Attempted (refused) FTP login again from webmaster.de.xxx.edu
- Users “bobby” (with UID 0), “boby” and group “bobby” created on the system (also confirmed in `/etc/passwd`, `/etc/shadow` above)
- `eth0` – Network interface goes into promiscuous mode indicating a sniffer may have been started.

- Users “bobby”, “boby” and “root” log into the system from 128.xxx.18.179 using ssh2 on port 3312.

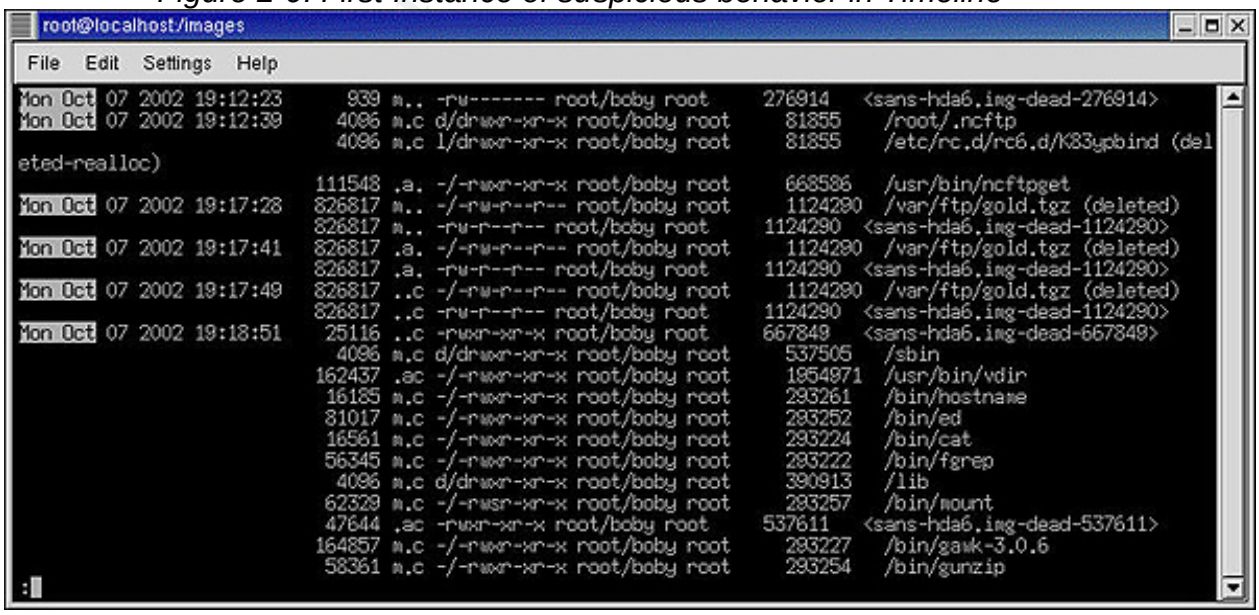
These are good indicators that the system had been compromised. There was no legitimate reason for these events to have occurred on this system. The information gathered so far provided some insight into where the intruder had come from, the approximate timeframe of the intrusion, and some activities the intruder had done. It is clear from this logfile that the intruder had gained root access via FTP, created some user accounts, installed a backdoor ssh daemon on port 3312, logged in from 128.xxx.18.179, and was possibly running a network sniffer.

Initial Analysis of timeline and Recovery of Deleted files

Before turning to the timeline file to corroborate some of the information I had just gathered from `/var/log/messages` and gather more information the intruder’s activities, I checked and noted the contents of `/etc/crontab` to establish when the `cron` daemon would run and what it would modify. Doing this allowed me to note the changes made to the system by the `cron` daemon and possibly eliminate it as being changes the intruder or user made.

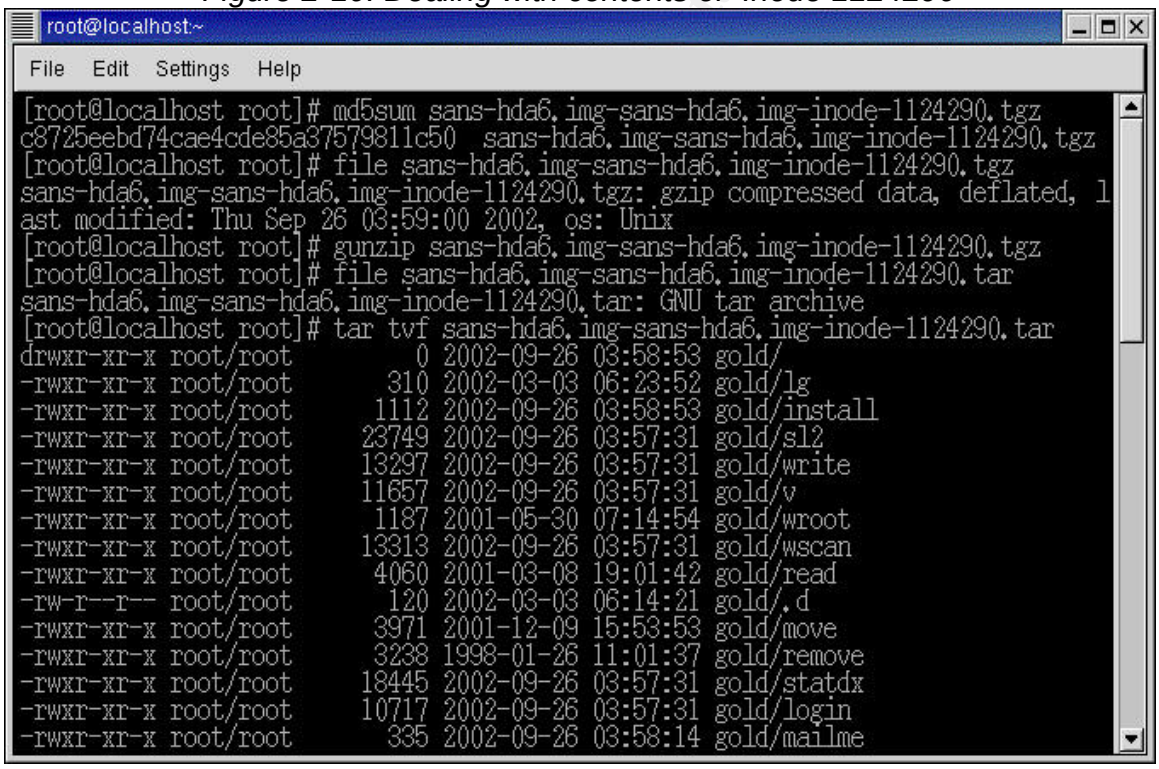
I parsed the timeline file using `/usr/bin/less`. The beginning of the timeline showed the date and time of the installation of the OS which were consistent with both what the user had told me and what was found in `/tmp/install.log`. The first instances of suspicious behavior in the time line which weren’t related to the OS install ,user or `cron` are shown in Figure 2-9.

Figure 2-9: First Instance of suspicious behavior in Timeline



According to the timeline, at 19:12:23 on Oct 7th inode 276914 was modified, and at 19:12:39 `ncftpget` was executed by root, which created `/root/.ncftp` It also shows that the file `/etc/rc.d/rc6.d/K83ypbind` was deleted as was the file `/var/log/gold.tgz` a few moments later.
Inode 1124290 is the same size as the deleted `gold.tgz` file, so using the inode browsing feature of Autopsy, I viewed the contents of the inode to determine if it contained the data of the deleted `gold.tgz` file. Autopsy identified the contents of inode 1124290 as "gzip compressed data, from UNIX". I then extracted the contents of inode 1124290 out to a file on the forensics box using the "Export Contents" functionality in Autopsy. After verifying that the MD5 checksum of the inode contents as reported by Autopsy matched the output of `md5sum` on the file that was extracted thus proving the file hadn't been modified in the extraction process, I proceeded to uncompress the file, which resulted in a "tar" file. I checked the contents and paths of the "tar" file using the `tvf` flags of the `/usr/bin/tar` command and then "untarred" it using the `xvf` flags so I could analyze the contents.

Figure 2-10: Dealing with contents of Inode 1124290



Analyzing the Rootkit

"Un-tarring" the "tar" file resulted in a directory named `gold` being created with several files in it. It appeared to be a rootkit containing tools that were potentially used by the intruder. I ran `/usr/bin/md5sum` on all the files found in the rootkit to fingerprint them. I then used `/usr/bin/file` to determine the type of each

of the files. `/usr/bin/strings` provided me some insight into the binary files and I used `/use/bin/less` to view the text and shell script files.
Figure 1-10 contains the names of the files from the rootkit and a brief explanation of the files' purposes based on the information I gathered using `strings` and `less`.

Figure 2-11: Contents of Root kit:

File Name	Description
.a	a zero length file
.c	a file containing Netblocks and domains
.d	file containing process names
.inetd.conf.swp	a VIM editor swap file - It's contents indicate it was editing /usr/X11R6/lib/X11/fonts/misc/ /dan/inetd.conf
.p	contains a list of the files in the rootkit
.sdc	sshd config file
.shk	sshd private key file
.x.tgz	gzipped tar file containing ADORE LKM source code (untarred into ".x" directory)
7350wurm	7350wurm - x86/linux wuftpd <= 2.6.1 remote root (version 0.2.2) - remote root exploiter of wu-ftpd
chattr	strings comparison shows it looks like stock chattr (used to change file attributes)
check	shell script to unpack .x.tgz in /usr/X11R6/lib/X11/fonts/misc/" "/,compile it and run ./start
cl	is "sauber" log cleaner
clean	shell script which runs "cl" and passes (yahoo.com, sshd, 208.158.209.235, rotind) as args.
dir	a version of dir
du	a version of du
encrypt	SOLcrypt 1.0 by sensei - to encrypt/decrypt files
fix	may try to modify checksums
ifconfig	a version of ifconfig - a strings comparison revealed it wouldn't show the PROMISC flag set.
init	shell script to run `[dan1,2] -p 3200 -q` from /usr/X11R6/lib/X11/fonts/misc/" "/ directory Runs /usr/X11R6/lib/X11/fonts/misc/" "/ .x/start Runs ./ava i `/sbin/pidof initd` >>/dev/null
install	shell script to install root kit
killall	a version of killall
lg	shell script to copy trojan version of login to /bin/login
Libproc.so.2.0.6	shared library used by trojaned lsof
login	trojaned version of login with string **cocacola**, references /bin/sh,/dev/mount (from login trojaning script "lg")
logn	trojaned version of login with string "bebito", references /bin/sh,and /usr/lib/.x
ls	a version of ls - references /tmp/extfsRNV23z
lsof	a version of lsof
mailme	a shell script which emails "smoke@cacanar.com" info about the compromised host
Md5sum	a version of md5sum - references /tmp/extfsRNV23z
move	shell script to install trojan sshd and others (see timeline)

File Name	Description
netstat	a version of netstat
patch	shell script to patch sshd version (see timeline)
ps	a version of ps - references /tmp/extfsRNV23z
pstree	a version of pstree (references /tmp/extfsRNV23z)
read	perl script to parse output of "LinSniffer"
remove	shell script which installs some trojan binaries (see timeline)
sc	scanning program
scan	statd scanner (looking for vulnerable 111 ports) runs "sc $1 111 $2 $3
sl2	modifed version of "slice" flooder from Knark. "strings" shows: anti-foonet by blizzard - based on sl2 - Usage: %s srcaddr dstaddr low high. If srcaddr is 0, random addresses will be used
ssh_host_key	host key for trojaned version of ssh
ssh_random_seed	random seed file for trojaned ssh
sshd	sshd trojan - references /usr/lib/.sdc, /usr/lib/.shk
Sshd_config	sshd_config file which references /usr/lib/.shk as Hostkey and /usr/lib/.srs as Random Seed
startfile	shell script to modify /etc/rc.d/rc.sysinit, rc.local, boot.local
statdx	scanner to exploit vulnerable versions of statd
top	a version of top, references /tmp/extfsRNV23z
v	Vadim v.Ibeta by Luciffer – udp flooder
vdir	version of vdir, references /tmp/extfsRNV23z
write	ethernet sniffer - writes to tcp.log
wroot	shell script to start wu ftpd vulnerability scanner
wscan	wu-ftpd vulnerability scanner
wted	strings shows "utzap" to remove login entries from wtmp

Included in the rootkit was a gzipped tar file `.x.tgz`. Unpacking the tarball and analyzing the files that were extracted from it into the directory `.x` revealed it was potentially the Adore Loadable Kernel Module (LKM).

I looked in all files in the .x direcotory for a software version number so I could download the Adore LKM source code from the Internet and compare it. The `Makefile` had the following entry `CFLAGS+=-DCURRENT_ADORE=42` and the `Changes` file listed the latest version as 0.42. Both of these files indicated that it was version 0.42
A quick Google search showed I could download the adore-0.42 source code from http://packetstormsecurity.nl/groups/teso/adore-0.42.tgz

I downloded the Adore LKM source, and compared md5 sums of files found in rootkit and files downloaded from net. Figures 2-12 and 2-13 are a comparison of the files in the .x directory and the source files downloaded from the Internet.

Figure 2-12: Adore LKM files from rootkit

md5sum	Filename
60a6b90f32d8387457c0357ffe33605e	.x/Changelog
8b35274c9f833c760738cd5765a5c1ba	.x/LICENSE
2d0c11e5237baac55759118567901e72	.x/Makefile
e4346f1a3a5fed10786e49b65fab7e6c	.x/Makefile.gen
9d626bf8f6874e63a64403ff24757b9d	.x/README
13d8ca70a0ca77b62c44c903c7d961d4	.x/TODO
9c1b9c8551e4ccfdfe2eb66a88588f69	.x/adore.c
7ae6abeb0db8e2ac4cb8f7b46613c8cf	.x/adore.h
a8af09fd53d76d218b3fadeb70d1fc09	.x/ava.c
3cb6c54561a78dd9c555cc3cbbf95ebc	.x/cleaner.c
03e0e705646ba77d7a399d952f15d6a6	.x/configure
ca37049245b51319ddc068f23882c3f9	.x/dummy.c
26e38f23062df4037a287303ea021484	.x/libinvisible.c
8af11813c20a544a60d2ba2d9f8f3f67	.x/libinvisible.h
158e51f5f2ceb287a4658257c9895f40	.x/rename.c
3de6dd6e7688f525e21d951fdf300e80	.x/start

Figure 2-13: Adore source from 'net download

md5sum	Filename
60a6b90f32d8387457c0357ffe33605e	adore/Changelog
8b35274c9f833c760738cd5765a5c1ba	adore/LICENSE
e4346f1a3a5fed10786e49b65fab7e6c	adore/Makefile.gen
9d626bf8f6874e63a64403ff24757b9d	adore/README
13d8ca70a0ca77b62c44c903c7d961d4	adore/TODO
4ae10ffd24d3038d555bbcd068e4db5b	adore/adore.c
b3b405ae9d97d68234208cda2f4a195b	adore/adore.h
a8af09fd53d76d218b3fadeb70d1fc09	adore/ava.c
3cb6c54561a78dd9c555cc3cbbf95ebc	adore/cleaner.c
55dbe55097ec9cbda701de95c084eec2	adore/configure
ca37049245b51319ddc068f23882c3f9	adore/dummy.c
26e38f23062df4037a287303ea021484	adore/libinvisible.c
8af11813c20a544a60d2ba2d9f8f3f67	adore/libinvisible.h
158e51f5f2ceb287a4658257c9895f40	adore/rename.c
92a334f54cf6f2ea67c3ac2c134ccef9	adore/startadore

The MD5 checksums of the files highlighted differ between the source code and the rootkit indicating that the source code of the kit on the compromised host had been modified. By using `/usr/bin/diff` of the files in question it was determined that the files had indeed been tailored to this particular install.

Figure 2-14: Diffs of adore.c

```
# diff adore.c ../.x/adore.c
543c543
<       if (strcmp(current->comm, "netstat") == 0 ) {
---
>       if ((strcmp(current->comm, "netstat") == 0 ) ||
(strcmp(current->comm, "lsof") == 0)){
```

For this rootkit installation: line 543 of adore.c has been modified from the original to check if the process to hide matches `netstat` or `lsof`.

Figure 2-15: Diffs of adore.h

```
# diff adore.h ../.x/adore.h
83c83
<       {":hell", ":2222", NULL};
---
>       {":initd", ":25330", ":48744", ":xbnc", ":write", NULL};
```

Line 83 of adore.h found with the rootkit has also been modified to account for processes and ports specific to this rootkit (`initd,xbnc,write` and ports 25330, 48744).

Figure2-16: Diffs of configure

```
# diff configure ../.x/configure
18a19
> $pass = "electricreality";
24,30c25
< print "\n\nSince version 0.33 Adore requires 'authentication' or\n".
< "its services. You will be prompted for a password now and this\n".
< "password will be compiled into 'adore' and 'ava' so no further
   actions\n".
< "by you are required.\nThis procedure will save adore from
   scanners.\n".
< "Try to choose a unique name that won't clash with normal calls to
  mkdir(2).\n";
<
<  print "Password (echoed):"; my $s = <STDIN>;
---
>  print "Password (echoed):"; my $s = "electricreality";
```

In the configure script found with the rootkit, a print statement with instructions was removed and the password compiled into `adore` and `ava` was statically defined as "*electricreality*", This was most likely done so that the compile process could be automated and would not produce output or require user intervention.

Figure 2-17: Diffs of start/startadore

```
# diff startadore ../.x/start
5d4
< # insmod adore without $0 but then its visible.
7,9d5
< insmod adore.o
< insmod cleaner.o
< rmmod cleaner
10a7,21
> if [ -f adore.o ] ;then
>
> mv adore.o xC.o
> fi
>
> if [ -f xC.o ] && [ -f cleaner.o ];then
>
> /sbin/insmod xC.o
> /sbin/insmod cleaner.o
> /sbin/rmmod cleaner
> ./ava i `/sbin/pidof initd`
> ./ava i `/sbin/pidof write`
> fi
```

The `startup` file in the rootkit was not only renamed from `startadore` to `start`, it was modified to rename the `adore.o` loadable module to `xC.o` and also automate the starting of the `ava` program to hide the `initd` and `write` processes.

Timeline Analysis: Tracing the path of the intruder

At this point in the investigation, I had a relative idea of when the intruder got access to the system and identified the tools that were downloaded on to the system. The next step in the investigation was to piece together specifically how and when the intruder gained access, and determine exactly what the intruder modified on the system.
Figure 2-18 contains a complete timeline of significant events and modifications that occurred on the system from October 6th – 9th. I pieced together this timeline by analyzing the timeline file I generated with Autopsy, tracing through the install script included with the rootkit and correlating these sources with each other and log file entries I retrieved from the system. The Time column is the date and time the event occurred, the Timeline Entry was taken from the timeline generated by Autopsy, and the Description column is my interpretation of the specific event. (See page 7 and Figure 2-6 for an explanation of Timeline Entry syntax)

Figure 2-18: Complete Timeline of significant events

Time	Timeline Entry	Description
Sunday October 6th 2002		
10:55:51 till 11:18:21	• 0 mac ---------- root/boby root 1 <sans-hda6.img-alive-1> • 16384 m.c d/drwxr-xr-x root/boby root 11 /lost+found • 4096 mac d/drwxr-xr-x root/boby root 32577 /proc • 9829 .a. -/-rw-r--r-- root/boby root 162882 /tmp/install.log	Operating System installed by user (too many files accessed/created /modified to list – I've just listed the first few from the timeline here)
17:00:13	• 46300 .a. -/-rwxr-xr-x root/boby root 537524 /sbin/depmod • 1331 .a. -/-rw-r--r-- root/boby root 211812 /etc/sysconfig/harddisks	User boots system. User logs in and X windows starts (too many files accessed to list – I've just included the first few from the timeline here.)
18:52:49	• 3077 .a. -/-rw-r--r-- root/boby root 260941 /usr/lib/linuxconf/help.eng/notices/10-welcome.help	User starts linuxconf application (too many files accessed/changed to list them all)
18:53:57	• 4096 m.c d/drwx------ jimmy jimmy 33001/home/jimmy	User creates valid user account "jimmy" using Linuxconf application
Monday October 7th 2002		
19:12:39	• 111548 .a. -/-rwxr-xr-x root/boby root 668586 /usr/bin/ncftpget • 4096 m.c d/drwxr-xr-x root/boby root 81855 /root/.ncftp	Intruder gains access to the system. /usr/bin/ncftp is executed to download the rootkit.
19:17:28 till 19:17:49	• 826817 ..c -/-rw-r--r-- root/boby root 1124290 /var/ftp/gold.tgz (deleted)	rootkit tarball: /var/ftp/gold.tgz deleted
19:18:51	• 4096 m.c d/drwxr-xr-x root/boby root 537505 /sbin	install script from rootkit run from /var/ftp/gold directory. Install runs ./remove which modified /sbin
	• 89601 .ac -/-rwxr-xr-x root/boby root 1954978 /usr/sbin/lsof	remove script replaces lsof
	• 37984 ..c -/-rwxr-xr-x root/boby root 1954988 /lib/libproc.so.2.0.6	remove script replaces libproc.so.2.0.6
	• 38425 .ac -/-rwxr-xr-x root/boby root 1954987 /usr/bin/md5sum	remove script replaces md5sum
	• 38477 ..c -/-rwxr-xr-x root/boby root 1954960 /sbin/ifconfig	remove script replaces ifconfig

Time	Timeline Entry	Description
	• 61125 m.c -/-rwxr-xr-x root/boby root 1954959 /bin/netstat	remove script replaces netstat
	• 69893 m.c -/-rwxr-xr-x root/boby root 1954957 /bin/ps	remove script replaces ps
	• 40965 .ac -/-rwxr-xr-x root/boby root 1954958 /usr/bin/top	remove script replaces top
	• 19313 .ac -/-rwxr-xr-x root/boby root 1954956 /usr/bin/pstree	remove script replaces pstree
	• 46669 .ac -/-rwxr-xr-x root/boby root 1954974 /usr/bin/dir	remove script replaces dir
	• 162437 .ac -/-rwxr-xr-x root/boby root 1954971 /usr/bin/vdir	remove script replaces vdir
	• 28279 ..c -/-rwxr-xr-x root/boby root 1954972 /usr/bin/killall	remove script replaces killall
	• 121821 .ac -/-rwxr-xr-x root/boby root 1954975 /usr/bin/du	remove script replaces du
	• 162435 m.c -/-rwxr-xr-x root/boby root 1954973 /bin/ls	remove script replaces ls
	• 25624 .a. -/-rwxr-xr-x root/boby root 537508 /sbin/chkconfig • 17 ..c l/lrwxrwxrwx root/boby root 2313169 /etc/rc.d/rc4.d/S13portmap -> ../init.d/portmap (deleted) • 17 ..c l/lrwxrwxrwx root/boby root 2182802 /etc/rc.d/rc2.d/K87portmap -> ../init.d/portmap (deleted) • 17 ..c l/lrwxrwxrwx root/boby root 2085239 /etc/rc.d/rc0.d/K87portmap -> ../init.d/portmap (deleted) • 17 ma. l/lrwxrwxrwx root/boby root 49243 /etc/rc.d/rc5.d/S13portmap -> ../init.d/portmap (deleted) • 17 ..c l/lrwxrwxrwx root/boby root 2247965 /etc/rc.d/rc3.d/S13 portmap -> ../init.d/portmap (deleted) • 17 4096 m.c l/drwxr-xr-x 30 root 81663 /etc/rc.d/rc6.d/K87portmap (deleted-realloc)	remove script executes /sbin/chkconfig --del portmap to shutdown portmap daemon
19:18:52	• 250 ..c -/-rw-r--r-- root/boby root 1954969 /usr/include/file.h	remove script copies .p to /usr/includefile.h
	• 161 ..c -/-rw-r--r-- root/boby root 1954967 /usr/include/hosts.h	remove script copies .c to /usr/include/hosts.h
	• 120 ..c -/-rw-r--r-- root/boby root 1954950 /usr/include/proc.h	remove script copies .d /usr/include/proc.h
	• 93 m.c -/-rw------- root/boby root 195923 /etc/ftpusers	move script (called from install) executes: echo anonymous >> /etc/ftpusers echo ftp >> /etc/ftpusers
	• 19464 ..c -/-rwxr-xr-x root/boby root 293292 /usr/lib/.x	/usr/lib/.x created by "lg" script called from "move" script
	• 10717 .ac -/-rwxr-xr-x root/boby root 1954954 /bin/login	"lg" script (called from "move") replaces /bin/login
	• 6 .ac -/-rw-r--r-- jimmy jimmy 765952 /usr/X11R6/lib/X11/fonts/misc/ /.x/CVS/Repository	install script executes: mkdir -p /usr/X11R6/lib/X11/fonts/misc/" "/
	• 1345 ..c -/-rwxr-xr-x root/boby root 1954966 /usr/X11R6/lib/X11/fonts/misc/ /cl • 13297 ..c -/-rwxr-xr-x root/boby root 1954970 /usr/X11R6/lib/X11/fonts/misc/ /wted • 14796 ..c -/-rw-r--r-- root/boby root 1954989 /usr/X11R6/lib/X11/fonts/misc/ /.x.tgz	install script executes: • mv -f wted cl .x.tgz /usr/X11R6/lib/X11/fonts/misc/" "/

Time	Timeline Entry	Description
	• 18445 ..c -/-rwxr-xr-x root/boby root 1954953 /usr/X11R6/lib/X11/fonts/misc/ /statdx • 13297 ..c -/-rwxr-xr-x root/boby root 1954945 /usr/X11R6/lib/X11/fonts/misc/ /write • 982 ..c -/-rwxr-xr-x root/boby root 1954962 /usr/X11R6/lib/X11/ fonts/misc/ /scan • 11657 ..c -/-rwxr-xr-x root/boby root 1954946 /usr/X11R6/lib/X11/ fonts/misc/ /v • 4060 ..c -/-rwxr-xr-x root/boby root 1954949 /usr/X11R6/lib/X11/ fonts/misc/ /read • 13505 ..c -/-rwxr-xr-x root/boby root 1954963 /usr/X11R6/lib/X11/ fonts/misc/ /sc • 1187 ..c -/-rwxr-xr-x root/boby root 1954947 /usr/X11R6/lib/X11/ fonts/misc/ /wroot • 13313 ..c -/-rwxr-xr-x root/boby root 1954948 /usr/X11R6/lib/X11/ fonts/misc/ /wscan • 23749 ..c -/-rwxr-xr-x root/boby root 1954944 /usr/X11R6/lib/X11/ fonts/misc/ /sl2	• mv -f statdx write scan sc sl2 wroot wscan v read /usr/X11R6/lib/X11/fonts/misc/ " "/
	• 716993 m.c -/-rwxr-xr-x root/boby root 2117771 /usr/X11R6/lib/X11/fonts/misc/ /dan1	install script executes: cp -f sshd /usr/X11R6/lib/X11/fonts/misc/" "/dan1
	• 716993 m.c -/-rwxr-xr-x root/boby root 2117772 /usr/X11R6/lib/X11/fonts/misc/ /dan2	install script executes: cp -f sshd /usr/X11R6/lib/X11/fonts/misc/" "/dan2
	• 998 ..c -/-rw-r--r-- root/boby root 1954983 /usr/lib/.sdc • 541 .ac -/-rw------- root/boby root 1954981 /usr/lib/.shk	Install script executes: mv -f .sdc .shk /usr/lib/
	• 523 m.c -/-rw------- root/boby root 749927 /usr/lib/.shk2	Install script executes: cp -f ssh_host_key /usr/lib/.shk2
	• 512 mac -/-rw------- root/boby root 749928 /usr/lib/.srs	Install script executes: cp -f ssh_random_seed /usr/lib/.srs
	• 121180 .a. -/-rwxr-xr-x root/boby root 668500 /usr/bin/make • 307 .a. -/-rwxr-xr-x root/boby root 81873 /usr/X11R6/lib/X11/fonts/misc/ /.x/start	The install script executes "check" script which unzips .x.tgz, and compiles it. There's too many files created and accessed to list here. I've shown make being invoked and the "start" script to load Adore LKM from the check script: make >> /dev/null ./start >> /dev/null The make process takes until 19:19:11 when the Adore LKM “start” script is run
19:18:57	• 306 ..c -/-rwxr-xr-x root/boby root 1954976 /etc/rc.d/init.d/init	install script calls “startfile” script to copy init file from rootkit to /etc/rc.d/init.d/init to activate LKM and start Trojan ssh daemons at boot

Time	Timeline Entry	Description
19:19:01	• 401748 .a. -/-r-sr-xr-x root/boby root 1140314 /usr/sbin/sendmail	"mailme" script is executed by "install" It gathers information about the system from ifconfig, hostname,w, /proc/meminfo, route -n ,/proc/cpuinfo and emails it to smoke@cacanar.com
	• 3777 mac -/-rw------- root/boby root 1661574 /var/spool/mail/root	The email sent from the "mailme" script is bounced back to root since sendmail is not configured correctly
	• 945 m.. -/-rw-r--r-- root/boby root 2117778 /var/log/maillog	/var/log/mail.log is updated to report bounced mail
19:19:09	• 38651 .a. -/-rw-r--r-- root/boby root 2117775 /var/log/cron • 2880 .a. -/-rw-r--r-- root/boby root 2117786 /var/log/secure • 3394 .ac -/-rw-r--r-- root/boby root 2117776 /var/log/dmesg • 4941 .a. -/-rw-r--r-- root/boby root 2117774 /var/log/boot.log • 0 .ac -/-rw-r--r-- root/boby root 2117781 /var/log/htmlaccess.log • 945 .ac -/-rw-r--r-- root/boby root 2117778 /var/log/maillog • 81072 .a. -/-rw-r--r-- root/boby root 2117783 /var/log/messages • 282 .ac -/-rw-r--r-- root/boby root 2117784 /var/log/netconf.log • 0 .ac -/-rw-r--r-- root/boby root 2117791 /var/log/statistics • 0 .ac -/-rw-r--r-- root/boby root 2117787 /var/log/spooler • 0 .ac -/-rw-r--r-- root/boby root 2117790 /var/log/xferlog	install calls "clean" shell script which runs "cl" with args to attempt to clean log files in /var/log
19:19:10	• 40028 .a. -/-rwxr-xr-x root/boby root 537567 /sbin/ipchains	install calls "clean" which shuts down sshd and executes ipchains
19:19:11	• 716993 mac -/-rwxr-xr-x root/boby root 1140266 /usr/sbin/sshd	install calls "clean" which replaces the sshd and restarts the new daemon
	• 0 mac d/drwxr-xr-x root/boby root 1954941 /var/ftp/gold (deleted)	install script executes: rm -rf gold*
19:20:12	• 4096 m.c d/drwxr-xr-x root/boby root 374625 /home • 4096 m.c d/drwx------ bobby bobby 146960 /home/bobby	User bobby created by intruder
	• 4096 m.c d/drwx------ bobby bobby 146960 /var/log/sa (deleted -realloc)	/var/log/sa which was deleted by "patch" script gets reallocated to /home/bobby
19:20:50	• 688 m.c -/-rw-r--r-- root/boby root 521616 /root/.emacs • 24 m.c -/-rw-r--r-- root/boby root 521424 /root/.bash_logout	user boby created with uid 0 and /root as home dir
19:20:56	• 13536 .a. -/-r-s--x--x root/boby root 668868 /usr/bin/passwd	/usr/bin/password executed to set password for user bobby
19:27:57	• (see /var/log/messages file)	eth0 put in promiscuous mode
19:46:49	• (see /var/log/messages) • 146584 m.c -/-rw-r--r-- root/boby root 276898 /var/log/lastlog	user root logs in from webmaster.de.psu.edu
19:47:59	• 64604 .a. -/-rwxr-xr-x root/boby root 668167 /usr/bin/ftp	/usr/bin/ftp executed to download psyBNC source code
19:50:36	• 224 .ac -/-rw-rw-r-- root/boby root 1987406 /usr/X11R6/lib/X11/fonts/misc/ / /xbnc/	/usr/X11R6/lib/X11/fonts/misc/ / /xbnc/ directory created when psybnc kit is untarred into this directory - too many files created to list here.

Time	Timeline Entry	Description
19:51:16	• 524596 .a. -/-rwxr-xr-x root/boby root 1726721 /usr/X11R6/lib/X11/ fonts/misc/ / /xbnc/xbnc • 524596 .a. -/-rwxr-xr-x root/boby root 1726721 /usr/X11R6/lib/X11/ fonts/misc/ / /xbnc/psybnc (deleted-realloc)	psybnc program renamed to xbnc
	• 2075 .a. -/-rw------- root/boby root 2199405 /usr/X11R6/lib/X11/ fonts/misc/ / /xbnc/log/psybnc.log • 6 mac -/-rw------- root/boby root 1726723 /usr/X11R6/lib/X11/ fonts/misc/ / /xbnc/psybnc.pid	xbnc program started creating psybnc.pid and psybnc.log
19:52:24	• 4096 m.c d/drwxrwxr-x root/boby root 2199403 /usr/X11R6/lib/X11/ fonts/misc/ / /xbnc/log	Intruder logs into IRC as “andrei” from 81.196.65.132
19:55:41	• 37884 .a. -/-rwxr-xr-x root/boby root 1140325 /usr/sbin/in.telnetd	telnet connection attempted to host
20:07:42	• (see psybnc.log)	User andrei logged off IRCfrom 81.196.65.132
Tuesday October 8th 2002		
2:52:59	• 4096 m.c d/drwxrwxr-x root/boby root 1319728 /usr/X11R6/lib/X11/ fonts/misc/ / /xbnc/motd	/usr/X11R6/lib/X11/fonts/misc/ / /xbnc/motd/USER1.MOTD changed by IRC user.
4:20:13	• 13297 .a. -/-rwxr-xr-x root/boby root 1954945 /usr/X11R6/lib/X11/fonts/misc/ /write	write program (sniffer) is accessed
05:46:25	• (see psybnc.log)	Failed Authentification for ravens from host 62.231.98.76 via IRC
5:50:22	• 3651 .a. -/-rw-r--r-- bobby bobby 147118 /home/bobby/.screenrc • 230 .a. -/-rw-r--r-- bobby bobby 147115 /home/bobby/.bash_profile • 124 .a. -/-rw-r--r-- bobby bobby 147116 /home/bobby/.bashrc • 24 .a. -/-rw-r--r-- bobby bobby 147114 /home/bobby/.bash_logout	Intruder logs in as bobby
Wednesday October 9th 2002		
13:25:43	• 70216 .a. -/-rwxr-xr-x root/boby root 668310 /usr/bin/gnome-linuxconf	User - who is logged in ran linuxconf – sees user “bobby”, “boby” – suspicious someone has hacked in
16:10:11	• 14460 .a. -/-rwxr-xr-x root/boby root 537556 /sbin/shutdown	User initiated shutdown

In the process of compiling the complete timeline shown in Figure 2-18, I came across additional evidence related to the intruder’s activities on the system. Specifically, I discovered a bounced email message and that the user was using an IRC bouncer.

Using Autopsy’s “File Browsing” function, I recovered the email that the `mailme` script (called from the rootkit `install` script) attempted to send. This email, shown in Figure 2-19 was intended to provide the recipient smoke@cacanar.com information about the system that had been compromised.

The email, however, was never sent because `sendmail` wasn’t configured on the system. Instead, the it bounced back to root’s inbox `/var/spool/mail/root` from which I extracted it using Autopsy.

Figure 2-19: Contents of Email sent by intruder (full email headers not shown)

```
From: root <root>
Message-Id: <200210072319.g97NJ1i19847@localhost.localdomain>
To: smoke@cacanar.com
Subject: root:cpu MHz           : 501.143:localhost.localdomain

          inet addr:128.119.x.xx  Bcast:128.119.x.xx  Mask:255.255.255.x
          inet addr:127.0.0.1  Mask:255.0.0.0
localhost.localdomain
Linux localhost.localdomain 2.2.16-22 #1 Tue Aug 22 16:49:06 EDT 2000
i686 unknown
  7:18pm  up 1 day,  2:19,  2 users,  load average: 0.39, 0.08, 0.03
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                Sun 6pm 25:05m  1.11s  0.03s  sh
/usr/X11R6/b
root     pts/0    :0               Sun 6pm 24:24m  0.04s  0.04s  bash
        total:    used:    free:  shared: buffers:  cached:
Mem:  263716864 133496832 130220032 157261824 35594240 42483712
Swap: 271392768        0 271392768
MemTotal:    257536 kB
MemFree:     127168 kB
MemShared:   153576 kB
Buffers:      34760 kB
Cached:       41488 kB
BigTotal:         0 kB
BigFree:          0 kB
SwapTotal:   265032 kB
SwapFree:    265032 kB
PING yahoo.com (66.218.71.198) from 128.119.x.xx : 56(84) bytes of data.
64 bytes from w1.rc.vip.scd.yahoo.com (66.218.71.198): icmp_seq=0 ttl=244
time=84.765 msec
64 bytes from w1.rc.vip.scd.yahoo.com (66.218.71.198): icmp_seq=1 ttl=244
time=84.673 msec
--- yahoo.com ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/mdev = 84.673/84.719/84.765/0.046 ms
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use
Iface
128.119.xx.xx   0.0.0.0         255.255.255.0   U     0      0        0
eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0
lo
0.0.0.0         128.119.x.xx    0.0.0.0         UG    0      0        0
eth0

--g97NJ1h19875.1034032741/localhost.localdomain--
```

I also discovered that at 19:47:59 on Oct 7th the intruder had run `/usr/bin/ftp` on the system to download the psyBNC IRC bouncer into the directory `/usr/X11R6/lib/X11/fonts/misc  /  /psybnc`

I concluded by looking at the directory `/usr/X11R6/lib/X11/fonts/misc / /` with Autopsy that the `psybnc` directory had been renamed to `xbnc`. The `psybnc` directory was marked as (deleted-realloc) and it shared the same inode as the `xbnc` directory. In addition, the binary `psybnc` was also marked as (deleted-realloc), was the same size (524596 bytes) and shared the same inode (1726721) and the same MD5 checksum as the `xbnc` binary indicating the program itself had been renamed from `psybnc` to `xbnc`.
The intruder likely renamed it because `xbnc` is one of the processes that appeared in `.x/adore.h` for the Adore LKM to hide.
After parsing several of the source code files in the `xbnc` directory, I determined it was the psyBNC IRC bouncer version 2.3 source code. Using Google, I was able to find and download the source code to psyBNC v2.3 from http://www.psychoid.lam3rz.de/psyBNC2.3.tar.gz and compared it to the source code the intruder left behind. Goolge also referred me to http://www.netknowledgebase.com/tutorials/psybnc.html , which is a great tutorial on psyBNC. Essentially psyBNC allows IRC users to "bounce" through a host system (in this case the compromised box) and hide their real IP address from other IRC users.
To gather more clues about the IRC habits of the user I extracted the `psybnc.conf` file, which configures the IRC Bouncer, from `/usr/X11R6/lib/X11/fonts/misc / /`

Using http://www.netknowledgebase.com/tutorials/psybnc.html as a reference for interpreting psybnc.conf, I was able to determine information about the IRC configuration outlined in Figure 2-20.

Figure 2-20: Contents of psybnc.conf (My notes included with arrows)

```
PSYBNC.SYSTEM.PORT1=40401            ------------→ xbnc process operates
PSYBNC.SYSTEM.HOST1=*                               on port 40401
PSYBNC.HOSTALLOWS.ENTRY0=*;*
USER1.USER.LOGIN=andrei              ------------→ intruder's IRC name is
USER1.USER.USER=Mr.BIG                              "andrei"
USER1.USER.PASS==0z0t`Q`K0v0e'4'Y`h  -----------→ Encrypted password for user
USER1.USER.RIGHTS=1
USER1.USER.VLINK=0
USER1.USER.PPORT=0
USER1.USER.PARENT=0
USER1.USER.QUITTED=0
USER1.USER.DCCENABLED=1
USER1.USER.AUTOGETDCC=0
USER1.USER.AIDLE=0
USER1.USER.LEAVEQUIT=0
USER1.USER.AUTOREJOIN=1              ------------→ If the user is kicked off
USER1.USER.SYSMSG=1                                 the IRC channel, it will
USER1.USER.LASTLOG=0                                log back in automatically
USER1.USER.NICK=andrei
USER1.SERVERS.PORT1=6667             ------------→ port 6667 on IRC server
USER1.SERVERS.SERVER1=Oslo1.NO.EU.undernet.org --→ IRC server
USER1.SERVERS.SPASS1=6667            ------------→ Password to join channels
USER1.CHANNELS.ENTRY1=#Slick         ----|
USER1.CHANNELS.ENTRY0=#gold          ----|--------→ Channels intruder joins
```

The logfile `psybnc.log` , shown in Figure 2-21 that I extracted from `/usr/X11R6/lib/X11/fonts/misc / /xbnc/log/` contains a history of the intruder's IRC activity. This confirms the date and time the xbnc process was started, which I had already seen in the timeline generated by Autopsy. This also showed that the intruder "andrei" was logging in via IRC from 81.xxx.65.132 (and several failed attempts by user "ravens" from 62.xxx.98.76). I'm unable to tell from this whether those IPs are other bounce point the intruder is using, or their originating IP addresses. psyBNC does have the ability to be one of many hops in an IRC bounce.

Figure 2-21: Contents of psybnc.log

```
Mon Oct  7 19:51:16 :Listener created :0.0.0.0 port 40401
Mon Oct  7 19:51:16 :Error Creating Socket
Mon Oct  7 19:51:16 :Can't create listening sock on host * port 40401
Mon Oct  7 19:51:16 :Loading all Users..
Mon Oct  7 19:51:16 :No Users found.
Mon Oct  7 19:51:16 :psyBNC2.3-cBtITLdDMSNp started (PID :20963)
Mon Oct  7 19:52:24 :connect from 81.xxx.65.132
Mon Oct  7 19:52:25 :New User:andrei (andrei) added by andrei
Mon Oct  7 19:52:34 :User andrei () has no server added
Mon Oct  7 19:54:22 :User andrei () has no server added
Mon Oct  7 19:54:40 :User andrei () trying Oslo1.NO.EU.undernet.org port 6667 ().
Mon Oct  7 19:54:41 :User andrei () connected to Oslo1.NO.EU.undernet.org:6667 ()
Mon Oct  7 19:56:30 :Hop requested by andrei. Quitting.
Mon Oct  7 19:56:30 :User andrei got disconnected from server.
Mon Oct  7 19:56:45 :User andrei () trying Oslo1.NO.EU.undernet.org port 6667 ().
Mon Oct  7 19:56:45 :User andrei () connected to Oslo1.NO.EU.undernet.org:6667 ()
Mon Oct  7 20:07:42 :User andrei quitted (from 81.xxx.65.132)
Tue Oct  8 02:52:52 :User andrei () got disconnected (from Oslo1.NO.EU.undernet.
org) Reason: Closing Link: dick-66 by Oslo1.NO.EU.undernet.org (Ping timeout)
Tue Oct  8 02:52:55 :User andrei () trying Oslo1.NO.EU.undernet.org port 6667().
Tue Oct  8 02:52:56 :User andrei () connected to Oslo1.NO.EU.undernet.org:6667 ().
Tue Oct  8 05:46:25 :connect from 62.xxx.98.76
Tue Oct  8 05:46:25 :Failed Authentification for ravens from host 62.xxx.98.76
Tue Oct  8 05:46:25 :Lost Connection from 62.xxx.98.76 (ravens)
Tue Oct  8 05:46:28 :connect from 62.xxx.98.76
Tue Oct  8 05:46:29 :Failed Authentification for ravens from host 62.xxx.98.76
Tue Oct  8 05:46:29 :Lost Connection from 62.xxx.98.76 (ravens)
Tue Oct  8 05:46:32 :connect from 62.xxx.98.76
Tue Oct  8 05:46:32 :Failed Authentification for ravens from host 62.xxx.98.76
Tue Oct  8 05:46:32 :Lost Connection from 62.xxx.98.76 (ravens)
Wed Oct  9 16:10:28 :Program Context : src/p_socket.c/socketdriver Line 1229
Wed Oct  9 16:10:28 :Received TERMINATE signal from terminal
```

Throughout the course of the investigation I had recovered several deleted files with Autopsy which were significant pieces of evidence. Using the “Show All deleted” function on Autopsy I was able to generate a list of all the deleted files found on the evidence image. At this point in the investigation I had already found and recovered all the significant deleted data. The only other deleted files that Autopsy turned up when I listed them all were files deleted and reallocated by cron jobs. As stated earlier the user performed a clean shutdown before notifying me the system was compromised. As a result, the /proc filesystem was cleared of all information (the /proc filesystem exists only in memory, not on disk).

Strings Search

The swap partition on the disk contains the all that remains of what was in the memory of the system. The swap space contained on disk contains data that was

paged out from memory, and the data is not in any logical order. Parsing the image with strings command is the most efficient way to view its contents. The file is about 265 MB so I used the command `fgrep –f GREPFILE` to match specific patterns, contained in `GREPFILE`, which appeared in other evidence I had gathered from the system.
`GREPFILE` contains the strings:
`bobby,boby,psybnc,andrei,xbnc,ftp,gold,xC,ava,cleaner,initd,write,PROMISC,webmaster.de.xxx.edu,62.xxx.98.76,81.xxx.65.132,ravens`

Figure 2-22: strings of swap image

```
# strings sans-hdd5-swap.img | fgrep –f GREPFILE | less
```

This strings search matched some data that was on the swap partition, but nothing that was related to the intrusion, or the trojaned processes running on the system. I ran strings again without piping to the pattern matching "fgrep", so that I could parse the output by hand. This also didn't turn up any additional info related to the intrusion.

Autopsy has a "String Search" functionality build in that allowed me to enter some search criteria and a regular expression to match on the Linux filesystem image. I was able to search in both the allocated files and un-allocated files (deleted files) using the same patterns I entered in GREPFILE. The Autopsy "String Search" generates the output such that you can see the pattern matched and the inode/disk block that it appeared in. I reviewed the contents of all the inodes and disk blocks returned by Autopsy and didn't find any additional information.

Conclusion:

I was able to determine that the host had been broken into on Oct 7th 2002 at 19:12:24 from webmaster.de.xxx.edu. The intruder most likely used the wu-ftpd file globbing heap corruption vulnerability exploit to gain entry and leverage root privileges. I arrived at this conclusion based on the fact that the system was running a version of wu-ftpd (2.6.1) that was vulnerable only to the file globbing exploit, and the first sign of attack was an anonymous FTP login with root privileges. Once the intruder gained root access to the system via FTP, he/she immediately downloaded a rootkit into `/var/ftp`, which is the FTP user's home directory.
The rootkit installed by the attacker contained several trojaned versions of system binaries and daemons, programs to gather information about the current system, utilities to hide their presence on the system and tools to attack other systems. The attacker also installed the psyBNC IRC bouncer for communicating via IRC.

I can only speculate as to the intention of the intruder. The installed rootkit included tools to scan and compromise other systems via statd or wu-ftpd exploits, and perform Denial of Service attacks. Though there is no evidence that these tools were actually used on this system, it is likely that the intruder intended on using them. The intruder achieved minimum success by gaining access to this system though his ultimate goals were most likely halted by the quick response of the user who found the intrusion early on and removed the system from the network.

I reported my findings to the owners of the IP addresses from which the intruder accessed this system and asked that they secure their systems. I notified the user that his system had indeed been compromised and recommended that he format the hard drive, re-install a newer version of the the OS, apply the most recent security patches and disable unnecessary services (such as FTP) before connecting it to the University network again.

References

Card, Rémy. Ts'o, Theodore. Tweedie, Stephen. “Design and Implementation of the Second Extended Filesystem” http://e2fsprogs.sourceforge.net/ext2intro.html Dec 13 2002

Computer Privacy and Security (CoPS) Lab at the University of North Texas. "Analysis for the reverse engineering Challenge." 23 Oct 2002 http://www.honeynet.org/reverse/results/sol/sol-21/analysis.html

Dittrich, Dave
http://staff.washington.edu/dittrich/misc/largefiles.txt

Google. 02 Dec. 2002
http://www.google.com

grugq <grugq@lokmail.net>, scut <scut@team-teso.net>. “Armouring the ELF: Binary encryption on the UNIX platform” Phrack 59. 28 Nov. 2002
http://www.phrack.com/show.php?p=58&a=5

Guidance on New Authorities that Relate to Computer Crime and Electronic Evidence Enacted in the USA Patriot Act of 2001 (October 2001) U.S. Department of Justice 30 Nov 2002.
http://www.usdoj.gov/criminal/cybercrime/usapatriot_redline.htm

IANA. “Port Numbers” 15 Nov 2002
http://www.iana.org/assignments/port-numbers

“Interception and Disclosure of Wire, Oral or Electronic Communications Prohibited” 18 U.S.C. §2511 30 Nov 2002.
http://www.usdoj.gov/criminal/cybercrime/usc2511.htm

Jestrix. “Introduction to psyBNC” 18 Nov 2002.
http://www.netknowledgebase.com/tutorials/psybnc.html

Libpcap Homepage. 03 Nov 2002
http://freshmeat.net/projects/libpcap/?topic_id=809

Miller, Toby. “Detecting Loadable Kernel Modules.” Incident-response.org 30 Nov 2002
http://www.incident-response.org/LKM.htm

Miller, Toby. “Analysis of Knark Rootkit”
http://online.securityfocus.com/guest/4871 Mar 12 2001 6:00PM GMT

Moolenaar, Bram “VIM REFERENCE MANUAL” 27 Oct 2002

http://www.polarhome.com/vim/manual/v58/recover.html

Pesch, Roland H., Osier, Jeffery M. and Cygnus Support "The gnu Binary Utilities" 29 Oct 2002
http://www.skyfree.org/linux/references/binutils.pdf

Psychoid "the most psychoid" 25 Oct 2002
http://www.psychoid.lam3rz.de/

Rivest R., "RFC 1321" Apr. 1992
http://www.ietf.org/rfc/rfc1321.txt

tcpdump homepage Tcpdump Group, The. 22 Oct 2002
http://www.tcpdump.org

Tool Interface Standards, Portable Formats Specification, Ver 1.1 "Executable and Linking Format (ELF)" 30 Oct 2002
http://www.skyfree.org/linux/references/ELF_Format.pdf

Venema, Wietse., Farmer, Dan. "The Coroner's Toolkit (TCT)" 10 Nov. 2002
http://www.porcupine.org/forensics/tct.html

Vonck, Tjerk "IRC FAQ" 07 Nov. 2002
http://www.mirc.co.uk/ircintro.html

"Wu-ftpd File Globbing Heap Corruption Vulnerability" Feb 14, 2002
http://online.securityfocus.com/bid/3581

Legal Issues of Incident Handling

SANS GCFA Practical Assignment v.1.2 Part 3

Jacob Cunningham

Part 3: Legal Implications

Recently a law enforcement officer contacted me and informed me that an account on a system in my administrative domain was used to hack into a government computer. On the phone he asked me to verify the activity by reviewing the log files on the system and determine if the logs indicate the activity was initiated by the user on my system or from an upstream provider. I was able to determine from the log files that a valid user had logged in a dialup account during the period of suspicious activity. I have a pre-existing relationship with the law enforcement officer, so I have already verified his credentials and I'm sure that this is not an instance of social engineering.

When given this situation, I would immediately seek the advise of the company's legal counsel to protect my rights, the company's, and the rights of the subscriber. Without having done so, I've stated my opinion below of how I would handle the situation based on research that I have done.

There are several state and federal laws that apply to fraudulent and illegal activities on computers and computer networks. Most unauthorized "cracking" activities on computers and networks tend to be in violation of Federal Law under the Computer Fraud and Abuse Act (Title 18 U.S.C. §1030), the Wiretap Act (18 U.S.C. §2511), or the Electronic Communications Privacy Act (18 U.S.C. §2701).

The Electronic Communications Privacy Act (ECPA) 18 U.S.C §2702(a) outlines among other things, the privacy rights for customers of Internet service providers and dictates what information and under what circumstances ISPs can turn over information to law enforcement.

The ECPA distinguishes between two types of Internet service providers, public and private. The following definition of public and non-public providers as outlined in the ECPA is provided in Section III B of the Department of Justice document "Searching and Seizing Computers and Obtaining Electronic Evidence in Criminal Investigations":

"Services are available to the public if they are available to any member of the general population who complies with the requisite procedures and pays any requisite fees. For example, America Online is a provider to the public: anyone can obtain an AOL account.
(It may seem odd at first that a service can charge a fee but still be considered available "to the public," but this mirrors commercial relationships in the physical world. For example, movie theaters are open "to the public" because anyone can buy a ticket and see a show, even though tickets are not free.) In contrast, providers whose services are open only to those with a special relationship with the

provider are not available to the public. For example, employers may offer network accounts only to employees. See Andersen Consulting LLP v. UOP, 991 F. Supp. 1041, 1043 (N.D. Ill. 1998) (interpreting the "providing . . . to the public" clause in § 2702(a) to exclude an internal e-mail system that was made available to a hired contractor but was not available to "any member of the community at large").

There are different limitations around what public and private providers can disclose to law enforcement. The restrictions the ECPA places on the disclosure of information to law enforcement for the most part do not apply to non-public providers. The company that I work for is a public Internet service provider therefore certain statutory exceptions from 18 U.S.C §2702(b) must be met before I can disclose certain information to law enforcement officials.

The ECPA also defines several different types of information that Internet Service providers may possess and law enforcement may request relating to customers or subscribers.

- Transactional data from §2703(c)(1): “record or other information pertaining to a subscriber”. This has been interpreted to include "a log identifying the date, time, user, and detailed internet address of sites accessed" in H.R. Rep. No. 103-827, at 10, 17, 31 (1994), reprinted in 1994 U.S.C.C.A.N. 3489, 3490, 3497, 3511; United States v. Allen, 53 M.J. 402, 409 (C.A.A.F. 2000)
- Subscriber, billing and session specific information from §2703(c)(2): “name, address…telephone number or other subscriber number or identity and length of service of a subscriber to customer of such service and the types of services the subscriber or customer utilized”
- Stored content of communications that is defined in 18 U.S.C § 2510(8)

In my initial conversations with the law enforcement officer, I am able to confirm that my employer is a public Internet Service provider and disclose information about the company’s operational practices such as what sort of transactional and system logging is performed, and how long the logfiles are kept. In order for me to legally disclose subscriber information or transactional data contained in logfiles to the law enforcement officer about a customer or subscriber, one of the following exceptions stated in 18 U.S.C §2702(c) or 18 U.S.C §2702(b) must apply to the situation. 18 U.S.C §2702(b) provides exceptions for the disclosure of contents and 18 U.S.C §2702 (c) provides exceptions for the disclosure of other non-content customer records.

- Under §2702(c)(2) records can be disclosed: “with lawful consent of the customer or subscriber.” Login banners and user agreements have been considered legitimate consent by the user.

- Under §2702(c)(3) records can be disclosed for: "the protection of the rights or property of the provider of that service"
- Under §2702(c)(4) records can be disclosed: "if the provider reasonably believes that an emergency involving immediate danger of death or serious physical injury to any person justifies disclosure of the information"

In this situation, none of the exceptions allowed in §2702(b-c) apply. The law enforcement officer did not feel that there is the threat of death or serious injury, my company's user policy didn't provide me the authority to disclose their information, and the log files did not indicate that there was any threat to my company. The law enforcement officer must get the required legal authority before I can provide him with subscriber information or log files. If the officer is unable to get the documents for legal authority in a timely manner, during the phone call or subsequent phone calls the law enforcement officer can request me to "freeze" stored records and communications as outlined in 18 U.S.C §2703(f)(1) which states:
"A provider of wire or electronic communication service or a remote computing service, upon the request of a government entity, shall take all necessary steps to preserve records and other evidence in its possession pending the issuance of a court order or other process"
Section 2703 (f)(2) mandates that when the law enforcement officer issues a 2703f freeze order, the records he requested "shall be retained for a period of 90 days, which shall be extended for an additional 90-day period upon a renewed request by the government entity."

There are no provisions governing how effectively the ISP logs user activities or how effectively the ISP complies with the §2703(f) order. One limitation of the §2703(f) request discussed in the document "Searching and Seizing Computers and Obtaining Electronic Evidence in Criminal Investigations" is that it "cannot order providers to preserve records not yet made. If agents want providers to record information about future electronic communications, they must comply with electronic surveillance statutes".

As stated above, in order for me to send the law enforcement officer my secured log files related to this particular incident, the officer must get legal authority requesting the information as mandated in 18 U.S.C §2703(d):
"A court order for disclosure under subsection (b) or (c) may be issued by any court that is a court of competent jurisdiction described in section 3127(2)(A) and shall issue only if the government entity offers specific and articulable facts showing that there are reasonable grounds to believe that the contents of the wire or electronic communication, or the records or other information sought, are relevant and material to an ongoing criminal investigation."

The document “Searching and Seizing Computers and Obtaining Electronic Evidence in Criminal Investigations” list the following options for legal authority to request information:

- Subpoena:
  “Investigators can subpoena basic subscriber information”. The scope of “basic subscriber information” is listed in §2703(c)(2).

- Subpoena with prior notice to the subscriber or customer
  “Agents who obtain a subpoena, and either give prior notice to the subscriber or comply with the delayed notice provisions of § 2705(a), may obtain:
  1) everything that can be obtained using a subpoena without notice;
  2) "the contents of any wire or electronic communication" held by a provider of remote computing service "on behalf of . . . a subscriber or customer of such remote computing service." 18 U.S.C. § 2703(b)(1)(B)(i), § 2703(b)(2); and
  3) "the contents of a wire or electronic communication that has been in electronic storage in an electronic communications system for more than one hundred and eighty days." 18 U.S.C. § 2703(a).

- §2703(d) court order
  Law enforcement agents who possess an order under 18 U.S.C §2703(d) can get access to:
  “1) anything that can be obtained using a subpoena without notice; and 2) all "record[s] or other information pertaining to a subscriber to or customer of such service (not including the contents of communications [held by providers of electronic communications service and remote computing service])." 18 U.S.C. § 2703(c)(1).”

- §2703(d) Order with prior notice to the subscriber or customer:
  “Agents who obtain a court order under 18 U.S.C. § 2703(d), and either give prior notice to the subscriber or else comply with the delayed notice provisions of § 2705(a), may obtain:
  1) everything that can be obtained using a § 2703(d) court order without notice;
  2) "the contents of any wire or electronic communication" held by a provider of remote computing service "on behalf of . . . a subscriber or customer of such remote computing service," 18 U.S.C. § 2703(b)(1)(B)(ii), § 2703(b)(2); and
  3) "the contents of a wire or electronic communication that has been in electronic storage in an electronic communications system for more than one hundred and eighty days." 18 U.S.C. § 2703(a). “”

- Search warrant:

"Investigators can obtain the full contents of an account with a search warrant. ECPA does not require the government to notify the customer or subscriber when it obtains information from a provider using a search warrant."

After the law enforcement officer contacted me and notified me that a customer or subscriber may have been involved in criminal activity, I reviewed the relevant logs on the system to determine it was a valid user who logged in via a dialup account. Upon reviewing the log files, I did not find any evidence that the user was doing anything wrong. Because there was no evidence that the user was doing anything wrong, in accordance with the law, I as the system administrator cannot monitor the activities of that user to find proof of wrongdoing based solely on suspicion or curiosity. The user's rights pertaining to being monitored in real-time are protected by the Wiretap Act (18 U.S.C. §2511), which prohibits, among other things, the interception and monitoring of computer communications unless a specific exception applies to the situation. There are several exceptions stated in §2511 that permit the interception and monitoring of computer communications. The exceptions that pertain most to system administrators are:

- §2511(2)(a)(i) is referred to as the "provider exception":
  "It shall not be unlawful under this chapter for an operator of a switchboard, or an officer, employee, or agent of a provider of wire or electronic communication service, whose facilities are used in the transmission of a wire or electronic communication, to intercept, disclose, or use that communication in the normal course of his employment while engaged in any activity which is a necessary incident to the rendition of his service or to the protection of the rights or property of the provider of that service, except that a provider of wire communication service to the public shall not utilize service observing or random monitoring except for mechanical or service quality control checks."

- §2511(2)(c) is referred to as the "consent exception":

"It shall not be unlawful under this chapter for a person acting under color of law to intercept a wire, oral, or electronic communication, where such person is a party to the communication or one of the parties to the communication has given prior consent to such interception."

- §2511(2)(i) is referred to as the "computer trespasser exception":

If the logs disclosed that the hacker had gained unauthorized access to my system, created an account and used that unauthorized account to hack into the government computer, then a few of the exceptions apply and I would legally be able to monitor the activities of the user. The provider exception §2511(2)(a)(i) allows the system administrator to investigate the matter to protect the provider's

assets and prevent “theft-of-service” The D.O.J document “Searching and Seizing Computers and Obtaining Electronic Evidence in Criminal Investigations” states: “…system administrators can track hackers within their networks in order to prevent further damage. Cf. Mullins, 992 F.2d at 1478 (concluding that need to monitor misuse of computer system justified interception of electronic communications pursuant to § 2511(2)(a)(i)) …..United Sates vs. McLaren, 957 F. Supp. 215, 219 (M.D. Fla. 1997) determined there must be a “substantial nexus” between the monitoring and the threat to the provider’s rights and property.”

Although I as the system administrator can monitor to protect my company’s rights and assets, I cannot be asked to monitor by law enforcement for the purpose of collecting evidence for a case without the law enforcement officer having legal consent to do so. Legal consent can be in the form of the court documents listed previously, or consent may be granted through the “computer trespasser exception” in 18 U.S.C §2511(2)(i) which states: “Computer trespassers have no reasonable expectation of privacy on the systems”. 18 U.S.C §2510(21) states that a computer trespasser “does not include anyone known to the provider to have an existing relationship with the provider.” In this case, it is not known whether or not the user is know to have an existing relationship with my employer. Therefore the individual who set up the unauthorized account is considered a trespasser and can be monitored by the system administrator in conjunction with law enforcement under the trespasser exception as long as the following criteria are met:

- The owner or operator computer must authorize the monitoring of the trespasser's communications as stated in 18 U.S.C. § 2511(2)(i)(I)
- The person who intercepts the communications must be "lawfully engaged in an investigation." as stated in 18 U.S.C. § 2511(2)(i)(II).
- The person who intercepts the communications must have "reasonable grounds to believe that the contents of the computer trespasser's communications will be relevant to the investigation." as stated in 18 U.S.C. § 2511(2)(i)(III).
- The monitoring should only intercept communications transmitted to or from the computer trespasser as stated in 18 U.S.C. § 2511(2)(i)(IV).

The law enforcement officer and I would be able to meet all the above criteria and therefore could monitor the user under the computer trespasser exception of the Wiretap Act and any evidence gathered could be used in a court of law.

My research was focused around the Federal laws that govern the various aspects of dealing with the given situation, however there is one state law that also applies. Massachusetts General Law 272 §99 governs the “Interception of wire and oral communications.” It is modeled after the federal Wiretap Act, and contains similar language for prohibiting and allowing the interception of computer communications.

The notable difference between the exceptions listed in M.G.L 272 §90 D and ECPA 2511(2)(c) is that in Massachusetts, in order for the "consent exception" to apply, both parties of the communication must consent to the interception of information. Also, in M.G.L. 272 §90(D)(c) it clearly states that intercepting communications is permitted when the investigator is in compliance with the federal wiretap laws: "for investigative and law enforcement officers of the United States of America to violate the provisions of this section if acting pursuant to authority of the laws of the United States and within the scope of their authority."

In Summary, the Electronic Communications Privacy Act (18 U.S.C. §2701) regulates what information and under what circumstances ISPs can turn over information to law enforcement. I, as an employee of a public provider, cannot disclose subscriber or customer information to law enforcement unless one or more exceptions allowed in 18 U.S.C §2702(b-c) apply. In this case, the law enforcement official would have to issue a §2703(f) freeze order on the log files and obtain a court document (subpoena, search warrant) to gain access to the customer or subscribers information and log files etc.
In order for me the system administrator or the law enforcement officer, to conduct real-time monitoring of communications, they are bound by the provisions outlined in the Wiretap Act (18 U.S.C. §2511). To legally intercept real-time computer communications, one of the many exceptions listed in §2511(2) must apply. Several exceptions are made which would allow me, the system administrator to monitor intruders on the system to protect the provider's rights and property. However when I, the system administrator am working in conjunction with law enforcement (under the color of law), I must adhere to the guidelines mandated in §2511(2)(i), which relate to monitoring computer trespassers.

References:

18 U.S.C §1030
http://www4.law.cornell.edu/uscode/18/1030.html

18 U.S.C §2510
http://www4.law.cornell.edu/uscode/18/2510.html

18 U.S.C §2511
http://www4.law.cornell.edu/uscode/18/2511.html

18 U.S.C §2702
http://www4.law.cornell.edu/uscode/18/2702.html

18 U.S.C §2703
http://www4.law.cornell.edu/uscode/18/2703.html

SEARCH, The National Consortium for Justice Information and Statistics. "Investigation of Computer Crime" 1998

Schwartz, Kurt N. Assistant Attorney General Deputy Chief, Criminal Bureau "Obtaining Transactional Records and Stored Communications from Electronic Communications Services (Telephone Companies, Paging Services and Internet Service Providers)" March 1,2000

US D.O.J – July 2002
Searching and Seizing Computers and Obtaining Electronic Evidence in Criminal Investigations.
http://www.cybercrime.gov/s&smanual2002.htm

M.G.L Chapter 272, Section 99
"Interception of wire and oral communications"
http://www.state.ma.us/legis/laws/mgl/272-99.htm