Global Information Assurance Certification Paper

GIAC Practical Version 1.2
Forensic Studies in the Digital World

Mark J. de Jong
March, 28 2003

Abstract

This paper goes over some of the many issues and processes regarding a forensic investigation of a compromised computer system. It is split up into three parts. Part one discusses the identification and capabilities of an unknown binary found on a compromised host. It details how the binary can be utilized and what measures can be used to safeguard against its use. Methods of detection for this particular binary and the legal ramifications towards the user who installs such a binary are also discussed.

Part two discusses a system compromise and the forensic process used to identify the actions of the hacker who compromised the host. Modified system binaries relating to a rootkit are identified along with newly created system processes installed by the hacker. Installation scripts used during the compromise are analyzed as well as the hacker tools installed on the machine. A timeline of all the activity on the compromised host has been created and is thoroughly discussed.

Part three goes over the legal issues involved when dealing with law enforcement regarding the compromise of a government computer system. It discusses the legalese regarding the rights of an ISP caught in the middle of a computer crime investigation. Discussions revolve around both the rights of the ISP as a thoroughfare for crime and the ISP being the main target of the investigation.

Part 1: Analysis of an Unknown Binary

On January 27, 2003 I acquired a compressed archive containing an unknown binary. The only information given about the binary was that it was extracted from a compromised host. No additional clues concerning the binaries function, its capabilities or its purpose were conveyed. My task was to analyze the binary to determine its capabilities and how it could be utilized on a compromised machine.

Upon receiving the zip file I immediately transferred it to my forensic computer for its initial analysis. I first analyzed the zip file to gather information about what the archive contained. Using the unzip utility with the *-lv* flags I was able to get some quick facts pertaining to the files inside. The *-l* flag lists the contents of the archive and the –v flag increases the verbosity of the unzip utility. The zip file was named binary_v1.2.zip and it contained two files, one was the binary named *atd* and the other was a file containing the binaries md5 hash.

After extracting the binary from the archive, I first performed an md5sum on the file and compared it to the md5 hash provided in the zip archive. Figure 1.1 shows the comparison between the two md5 hashes revealing that the file extracted from the zip archive is identical to the original file.

Figure 1.1 – md5sum comparison

After performing the md5 comparison I gathered ownership and mactime information as well as other file attributes with the program *stat*. The output from the stat program can be seen in figure 1.2. All of the file attributes can be seen in figure 1.3.

Figure 1.2 – stat output

```
[user@hostname user]$ stat atd
  File: "atd"
  Size: 15348        Filetype: Regular File
  Mode: (0666/-rw-rw-rw-)         Uid: (  534/ user)  Gid: (  534/
user)
Device: 48,11  Inode: 2663374   Links: 1
Access: Thu Aug 22 14:57:54 2002(00165.04:25:39)
Modify: Thu Aug 22 14:57:54 2002(00165.04:25:39)
Change: Mon Feb  3 18:23:20 2003(00000.00:00:13)
```

Figure 1.3 – atd file attributes	
Name of binary	*atd* (name of binary on compromised system) *lokid* (actual binary name)
MACTime information (gathered with stat)	Access: Thu Aug 22 14:57:54 2002(00165.04:25:39) Modify: Thu Aug 22 14:57:54 2002(00165.04:25:39) Change: Mon Feb 3 18:23:20 2003(00000.00:00:13)
File Owners	Original owner and group unknown
File Size	15348 bytes
MD5 Hash	48e8e8ed3052cbf637e638fa82bdc566
Key words associate with program	LOKI2 route [(c) 1997 guild corporation worldwide]

I found that the original ownership attributes of the file were lost, possibly during the archival process. I tested this by extracting the files as root and as an unprivileged user. Both times I used the command *unzip -X binary_v1.2.zip.* The "-X" flag allows you to extract the files leaving the ownership attributes intact. In both cases the files inherited the ownership of the current user. There was still the possibility that the files were owned by root and that the unzip utility was defaulting the ownership permissions to the unprivileged user as a security precaution.

I tested this possibility be creating a separate zip archive and setting the ownership attributes to a user other than the ones I had been using to extract the files, user and group "news." When extracting the files as root, the owner and group attributes remained intact. When I extracted the files as the unprivileged user, I got an error explaining that the UID and GID attributes of the file could not be set thus proving that the ownership attributes of the files contained within the binary_v1.2.zip file had been lost. Finally I did a strings analysis on the binary to gather any information that might lead to the true identity of the malicious code.

Program Description

While performing a strings analysis of the binary *atd,* which was found on the compromised system, I discovered some very convincing evidence that the binary was not what its name would lead you to believe. The actual program *atd* is a daemon that schedules the execution of programs that have been spooled, by a user, in the daemons queue. This is much like a print server, which has a print queue to spool print jobs until the printer is ready to accept the job. In the case of *atd*, the queue contains commands and waits to execute those commands until its scheduled execution time has been reached. While *atd* and the suspect code masquerading as *atd* are relatively similar in file size, the contents of the two binaries are very different.

While browsing through the strings collected from the true *atd* binary, I noticed that many strings referred to files and processes associated with the scheduling

of jobs by the daemon. Keywords and phrases that I found relevant to the *atd* process were “nice”, “chdir”, “time”, “/var/spool/at/spool”, “trying to execute job %.100s twice” and many more. These keywords refer to either system calls or file locations used in scheduling the execution of programs by *atd*. Having been enlightened by my findings, I decided to investigate the malicious *atd* binary.

While searching through the strings contained in the malicious binary, I found a great deal of references to “lokid” and network socket calls. The *atd* binary from the compromised machine was in no means comparable to the true *atd* binary I had analyzed earlier. The very first string, /lib/ld-linux.so.1, gave a good indication that the binary was compiled on a Linux platform. The line refers to a shared library located on Linux machines. I also found the very descriptive string, “route [(c) 1997 guild corporation worldwide” that was later found in the programs source code.

I went to google.com and typed in “lokid” in the search field to find no shortage of relevant links. Most of the articles listed by Google had the common idea that *lokid* was known to be associated with exploited *NIX machines as a means to gain access or run commands once the box had been compromised. *Lokid* is the server side of a two-sided application that uses ICMP packets to transfer data back and forth between it and its client. Its payload is concealed within the ICMP packets data field, which is in most cases left empty. There are some occasions where the data field is used for sending timing information or testing the packets integrity, but in most common day scenarios this is rare.

ICMP stands for Internet Control Message Protocol. The protocol was developed to aide in troubleshooting network connectivity issues on an existing network. You can think of the ICMP protocol as the Internet’s Swiss army knife that can be used to test DNS name resolutions, routing problems, host connectivity and much more. It is a tool that can be used to help work through network issues ranging from the simplest to some of the more complicated events.

ICMP packets start with a 64 bit header followed by a variable length data field. Each 64 bit header consists of five different fields. The most important being the first field, which defines the ICMP packet as being one of a possible 256 different types, all of which can traverse the Internet. Every one has a unique purpose intended either for gathering or providing information concerning a given host. One of the most utilized ICMP packet type is type 8 and 0, the Ping echo and reply packet respectively.

Ping is used to elicit an echo-response from a remote host. It allows a troubleshooter to discover if the target host is connected to, and can communicate on, a given network. Ping initiates the dialog with a type 8, echo-request packet, and if the target host is online and is able to respond, ping receives a response from that host as a type 0, echo-reply packet.

Many administrators allow their firewalls, routers and gateways open to pass echo-request and reply packets. This is usually permitted as a means to test their networks from locations other than their internal infrastructure. Unbeknownst to them, they are making their networks more susceptible to hacker attacks. In many scenarios ping is initially used as a form of reconnaissance before an attack to determine if a host is "live." No matter what precautions an administrator might take, a pingable host is a vulnerable host, and the fact that it's sending echo-replies make it all the more appealing to a hacker. But the ICMP echo-request and reply packets cannot only be used as a method of reconnaissance, it can also be used as a method of client-server communication. This brings us to the concepts used by daemon9, the one behind the Loki project. His client-server application would use ICMP packets to transfer data between two hosts. And back in 1996 that's all it was – a concept.

Looking back at Norse mythology, Loki was regarded as the god of deceit and trickery. According to Phrack magazine, Loki was "well known for his subversive behavior" and "Inversion and reversal of all sorts was typical of him." All of those traits gave daemon9, the author of the August, 1996 Phrack magazine article, the idea to name his project "Project Loki." [1]

In 1996, Project Loki was a paper written to describe an ability to use common ICMP packets to mask communication channels between two hosts. ICMP packets are a common occurrence throughout the Internet and most administrators leaving this otherwise harmless protocol unmonitored and free to traverse to and from their networks. This being the case, any communication piggybacking on this protocol would, in many cases, be left undetected. This type of communication between hosts is referred to as a covert channel, skillfully hidden from the untrained eye.

A covert channel is a means for passing traffic between two hosts using a path that is not typically used towards that end. In the case of the Loki project, the 1996 paper described the ability to transmit data between client and server using the data portion of the common ping echo-request and reply packets which are in most cases left empty. As stated earlier ping packets are also referred to as type 8 and type 0 ICMP packets respectively. This would effectively circumvent detection and allow the traffic to easily pass through firewalls and scurry around older IDS systems without detection.

The idea behind the Loki Project was to illustrate the possibility of devising a method to gain access to a remote system undetected. This could allow a perpetrator to access a remote shell on the compromised system or send commands to the target machine. It could also be used to grab information such as documents, logs, database entries or any other type of electronic data. The main focus of the project was to make an ongoing session between two hosts look like ordinary ping data. In September of 1997, the idea became reality and was shared with the community.

The supplement code to the August 1996 Phrack article was released with the intention of showing the world that the theory of the covert channel via ICMP ping packets was possible. Its primary purpose was to educate the reader in terms of its implementation and use. The author makes it clear that the Loki program is a representation of an old idea and that the methods discussed have been used with other protocols for years.

The code included in the article can be built on four different platforms including Linux, the platform thought to be the host for the previously unknown binary. I did a quick search for some key words that I found during a strings analysis of the binary and found that same strings within the source code. Figure 1.4 shows the keywords that I searched for along with the segment of code that it was found in. The most convincing keywords linking the binary to the source code was the string mentioned earlier, "route [(c) 1997 guild corporation worldwide]," which was found a number of times within the code.

Figure 1.4 - Searched keywords and code segments

"route (c) 1997 guild corporation worldwide"

```
loki.h:#define L_MSG_BANNER    "\nLOKI2\troute [(c) 1997 guild
corporation
Makefile:# route (c) 1997 Guild Corporation, Worldwide
```

"inactive client"

```
client_db.c:                if (verbose) fprintf(stderr, "\nlokid:
inactive client
```

"Client database full"

```
client_db.c:        if (verbose) fprintf(stderr, "\nlokid: Client
database full");
```

"lokid –p (l|u) [-v (0|1)]"

```
loki.h:#define S_MSG_USAGE     "\nlokid -p (i|u) [ -v (0|1) ]\n"
```

I downloaded and extracted the source code from the original Phrack article using the code extraction program provided by Phrack. I attempted to build the binary on my forensic workstation, which is running Linux 2.4.18, but the original code was written for a much earlier version of Linux, version 2.0.x. I downloaded Redhat 5.2, which was the earliest version of Redhat for which an ISO was available. This version was released sometime in late 1998, one year after the release of the Loki source code. The Linux kernel version shipped with Redhat 5.2 was 2.0.36 meeting the requirements for the Loki code. However, I again was unable to compile the code.

Earlier versions on Redhat are available and can be downloaded. I attempted to install Redhat version 4.2 since it was released the same year the Loki code was released. Redhat 4.2 comes with Linux kernel version 2.0.30 meeting the requirements of the Loki source code. However, most of the installation methods

for Redhat require some sort of network connectivity and the network drivers provided with older distributions are relatively archaic. Every attempt at achieving a network installation failed. I attempted to create distribution CDs of Redhat 4.2 but was unable to get the install disks to recognize the CDs. I was about to give up hope of getting an old distribution installed when I ran into a friend who just happened to have an original Redhat 4.2 CD. I took this opportunity to try again and finally experienced success. I installed all of the development libraries and tools and attempted to get a network connection so that I could capture any packets that *loki* or *lokid* might throw out onto the network. But my attempts at getting the network connection up and running were futile so I decided to proceed without it.

I transferred the Loki source code over to the Redhat 4.2 box using the old trusted sneaker-net, otherwise known as the diskette. After extracting the code on the machine, I switched over to the Loki source directory and typed in "make linux." My first attempt at compiling the code was successful. However, when performing an md5sum on the *lokid* binary, I found that it did not match the hash of the unknown binary. Granted there are a number of methods available for compiling Loki.

The Loki makefile can be configured to conform to the users specific needs concerning cryptography of the covert stream, locations of certain cryptographic libraries, type of the child process handler to be used for spawning new child processes and other options pertaining to the functionality of Loki. The configurable portion of the Makefile can be seen in Figure 1.5. Each variation of the selected items will alter the md5sum of the resulting binary. I compiled the Loki code with every possible variation of options achievable on my Linux installation. I decided not to compile the binary with strong crypto support primarily due to lack of resources. Figure 1.6 shows the md5sums associated with the different variations of the compiled binary.

Figure 1.5 – LOKI2 Makefile

```
# Makefile for LOKI2 Sun Jul 27 21:29:28 PDT 1997
# route (c) 1997 Guild Corporation, Worldwide


######
#    Choose a cryptography type
#


#CRYPTO_TYPE            =   WEAK_CRYPTO         # XOR
CRYPTO_TYPE             =   NO_CRYPTO           # Plaintext
#CRYPTO_TYPE            =   STRONG_CRYPTO       # Blowfish and DH
```

Figure 1.5 – LOKI2 Makefile

```
######
#    If you want STRONG_CRYPTO, uncomment the following (and make sure
you have
#    SSLeay)

#LIB_CRYPTO_PATH        =    /usr/local/ssl/lib/
#CLIB                   =    -L$(LIB_CRYPTO_PATH) -lcrypto
#MD5_OBJ                =    md5/md5c.o

######
#    Choose a child process handler type
#
SPAWN_TYPE              =    POPEN
#SPAWN_TYPE             =    PTY

######
#    Addedum
#
NET3                    =    -DNET3
SEND_PAUSE              =    SEND_PAUSE=100
DEBUG                   =    -DDEBUG
#-------------------------------------------------------------------------
------#
```

Figure 1.6 – md5sums of lokid builds

Crypto?	Spawn type?	Debug?	Md5 hash of resulting binary	
unknown	unknown	unknown	48e8e8ed3052cbf637e638fa82bdc566 (Original)	atd
None	popen	yes	3da4181074cbe289a334af58a1828231	lokid
Xor	Popen	yes	06a53afee0953b0f910dbc3824d882d2	lokid
Xor	pty	yes	ccb653305fcc2148d3a33951ae2582cf	lokid
None	Pty	yes	574ebb9730f30780d4042a61f13d7aad	lokid
None	Popen	no	113add9aa4e4548af021a6b01ab1be87	lokid
Xor	Popen	no	64a01f369cc5ea4da573476da6055e88	lokid
Xor	pty	no	3ac9fc444b327cd0666f80d658bcac0c	lokid
None	Pty	no	257371a36f2f14abe10a6e6311054ce4	lokid

After compiling all the different *lokid* binaries I decided to choose one to perform any further testing with. However, it was still unclear which new *lokid* binary was most comparable to the *atd* binary. I could think of only one method of making a reasonably reliable decision and that was to compare string results from the

newly compiled binaries with the string results of the malicious *atd* file. I proceeded by running strings on every *lokid* binary and placing the results in separate text files. I then used the diff utility to report the differences between the strings results of the *lokid* binaries and the *atd* binary. I was pleasantly surprised when the results of one comparison were completely empty. The strings from both binaries were exactly the same.

While performing the string comparisons I discovered a few import things about what shows up in the strings results in relation to how the makefile is configured. The stings “pty” and “popen” are clearly visible when chosen as the child process handler type. The word “none” can be found in binaries with no encryption set and a good deal of what seem to be verbosity strings can be seen in binaries with the -DDEBUG option set. Referring back to the makefiles used to compile my copies of *lokid*, I determined that the malicious *atd* binary was compiled with CRYPTO_TYPE set to WEAK_CRYPTO, the child process handle type was set to POPEN and -DDEBUG was not set.

After deciding which binary most closely resembled the malicious code I proceeded to check the results of some common commands to see what they revealed about the malicious *atd* binary and compared the results to those found when running the same commands with a newly compiled copy of *lokid*.

ps (Process Listing)

I checked to see if the process was running using *ps* and it reported that *atd* was running as process ID 187. Given that the true *atd* application does not come standard with RH4.2, the authenticity of this claim could not be questioned. The process list can be seen in figure 1.7.

Figure 1.7 – System Process List (ps)

```
  PID TTY STAT  TIME COMMAND
    1  ?  S    0:03 init [3]
    2  ?  SW   0:00 (kflushd)
    3  ?  SW<  0:00 (kswapd)
   19  ?  S    0:00 /sbin/kerneld
   93  ?  S    0:00 syslogd
  102  ?  S    0:00 klogd
  113  ?  S    0:00 crond
  125  ?  S    0:00 inetd
  136  ?  S    0:00 lpd
  149  ?  S    0:00 sendmail: accepting connections on port 25
  164   1 S    0:00 /bin/login -- root
  165   2 S    0:00 /sbin/mingetty tty2
  166   3 S    0:00 /sbin/mingetty tty3
  167   4 S    0:00 /sbin/mingetty tty4
  168   5 S    0:00 /sbin/mingetty tty5
  169   6 S    0:00 /sbin/mingetty tty6
  171  ?  S    0:00 update (bdflush)
  172   1 S    0:00 -bash
  187  ?  S    0:00 ./atd
```

Figure 1.7 – System Process List (ps)

```
  199   1 R    0:00 ps -ax
```

netstat (net status)

Netstat reported a number of open raw sockets when the malicious *atd* binary was running. The more *atd* processes were running, the more raw sockets were open. When the *atd* processes were stopped, the open sockets were all closed. The same results were witnessed while running the newly compiled *lokid* binary. The *netstat* output can be seen in figure 1.8.

Figure 1.8 – netstat output

```
Active Internet connections (including servers)
Proto Recv-Q Send-Q Local Address           Foreign Address
State
tcp        0      0 0.0.0.0:21              0.0.0.0:*
LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:70              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:514             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:513             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:109             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:110             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:143             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:79              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:37              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:113             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:515             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:25              0.0.0.0:*               LISTEN
udp        0      0 0.0.0.0:514             0.0.0.0:*
udp        0      0 0.0.0.0:517             0.0.0.0:*
udp        0      0 0.0.0.0:518             0.0.0.0:*
udp        0      0 0.0.0.0:37              0.0.0.0:*
raw        0      0 0.0.0.0:1               0.0.0.0:*
raw        0      0 0.0.0.0:1               0.0.0.0:*
raw        0      0 0.0.0.0:255             0.0.0.0:*
```

gdb (Gnu Debugger)

I ran both the *atd* binary and the *lokid* binary through the Gnu-debugger program and found nothing of interest concerning either binary.

strace

Strace is a great utility that records all the system calls and signals produced and received by the monitored program. It can be set to follow child processes that may be spawned from the parent process. The results from *strace* show that the following takes place when the malicious *atd* program is executed. The process is listed in order of execution.

1. get file status of, and open /etc/ld.so.cache
2. open /lib/libc.so.5.3.12
3. get the user and group ID running the process.
4. Attempts to gather locality information
5. open up raw ICMP socket
6. set ICMP socket options
7. get process ID
8. echo " route [(c) 1997 guild corporation worldwide"
9. The process forks and the parent process dies
10. The child process opens a read write session to /dev/tty
11. The child process changes to the /tmp directory
12. The child process sets the umask to 022
13. The child process seems to be set to read input
14. For every command sent via the *loki* client a new read event is recorded by *strace*

Loki usage

I attempted to connect to the malicious *atd* (*lokid*) process using the *loki* client that had been compiled. Given the limited resources I had to work with I could only run the *loki* client on the same machine that the *lokid* process was running on. I ran the command "loki -d 127.0.0.1" which brought me to a "loki>" prompt, but when sending commands to the running *lokid* server (*atd*), the server failed to return any results.

Legal Implications in the State of Virginia

Every action has an equal and opposite reaction, and in the case of hacking into a system and altering its contents the equal an opposite reaction can throw the perpetrator right through the courtroom and into a jail cell. The legal ramifications of gaining access to a workstation or server without the prior consent of the owner and then placing a malicious binary on that system can be quite severe. Every state has their own laws concerning computer crimes. In the case of Virginia the act of logging into a system and uploading malicious code can put someone behind bars for a minimum of one year and a fine of up to $2500.

According to article 7.1, section 18.2-152.4 in the Code of Virginia, the alteration or erasure of any computer data, programs or computer software is punishable as a class one misdemeanor. As stated in title 18.2-11 the punishment for a class one misdemeanor is confinement in jail for no more than 12 months and a fine of no more than $2500, either or both. If the offender happens to perform his acts maliciously the damage equals or exceeds $2500, and the person can face a Class 6 felony charge, which would allow a court and jury to decide his or her fate. In this case the offender could find himself in prison for as long as five years with the addition of a fine up to $2500. Article 7.1, section 18.2-152.4 also covers the case of a hacker temporarily or permanently removing any computer data

from the target system, or causing the system to malfunction. This is important because in many cases the advent of a program such as *loki* is usually accompanied by the installation of a rootkit. A rootkit will usually overwrite key binaries of the operating system and could, in some cases, cause the system to malfunction.

In the event that the *lokid* binary is used for computer fraud, the hacker could face a class one misdemeanor charge if the services obtained by the offender are valued at $200 or less. This is referenced in article 7.1, section 18.2-152.3 in the Code of Virginia. Services worth greater than $200 can result in a class 5 felony charge. In this case the offender can face up to ten years in prison and up to $2500 in fines. In the case of an ISP, services are in many cases measured by bandwidth. The *loki* server, with its corresponding client, consumes bandwidth just like any other service that utilizes the Internet for its means of transport. This could be viewed as stolen bandwidth. The article covering computer fraud also covers embezzlement and larceny. Loki can be used for more than just sending commands and receiving feedback from that command. It can also be used to stream data - personal, top-secret or private.

If the hacker happens to be invading an individual's privacy, then charges would fall under article 7.1, section 18.2-152.5, computer invasion of privacy laws. It states that any person obtaining employment, salary, credit, or other financial or personal information will be punished under the terms of a class 3 misdemeanor. Granted it must be proven that the hacker examined the personal information after he or she understood that they didn't have the authority to do so, it still bares the penalty of up to 20 years in prison and a stiff fine of up to $100,000. As stated earlier, Loki could be used to transfer any form of information including an individual's private information.

Implications within the Corporate Environment

Many occurrences of computer cracking take place in the workplace. This makes sense for the opportunistic individual who doesn't want to spend the time attempting to subvert network defenses such as firewalls or IDS systems. Many companies put those types of measures in place to guard against aggressors trying to break in from the outside. However, in an effort to save money or just due to lack of foresight, most companies forgo implementation of a network security infrastructure on their private LAN. This gives an inside hacker all the time he or she needs to try out that latest exploit hyped on the Internet.

Some companies have taken the time to implement an Acceptable Use Policy clearly outlining what employees may or may not do within the confines of the corporate network. This policy can be as simple as stating that users may not browse websites on the Internet or deny the right to install certain software on the individuals computer. Others can become quite complex and explain pages

upon pages the rights and wrongs or do's and don'ts on the company's internal network.

Looking back at the Acceptable Use Policy written up by the company that I work for, I wanted to know what rules an individual would have broken if he or she decided to break into a computer system while on the job. We happen to have one of those complex policies so it took some time to find what I was looking for. I happened to find a few sections in the document that most likely could hold ground in the case of a computer hack. And if the employee were found guilty of the act, the penalty would surely be dismissal.

According to our acceptable use policy there are three sections that could be used against an employee who hacks into a machine and places a binary to be used as a covert-channel to that machine. The policies don't clearly state the incident as an attack or a break in and the policy doesn't use terms such as "covert-channel" or "binary" but the policies are clear and refer to such offenses in the most round about way.

The company's communication policy states that "no e-mail or other electronic communications may be sent that hides the identity of the sender or represents the sender as someone else." In the case of the *lokid* binary, its purpose is to hide an entire communication stream, effectively hiding the identity of the sender all-together. A case could be further corroborated if the individual had placed the binary on a machine outside of the companies network. Given that our internal network resides on private IP space all traffic traversing beyond our firewall and onto the Internet is represented by a single IP address. This may leave the impression that the communication is originating from our company as an entity, not from the employee's individual workstation. Thus, the employee would be representing the company through his or her communication stream.

The second policy, the Acceptable Software policy, would have been breached if the software was found on a company workstation or server. The Acceptable Software policy states that "Only software that is required to perform job functions is supported by the company's Information Systems department. The installation of unlicensed or non-business related software will not be tolerated and is grounds for dismissal." I have personally bared witness to the legal implications of installing unlicensed software and had to deal with a great number of reckless software installs by mindless users. The fallout was unpleasant and affected many people. The installation of an application such as *lokid* would just put another checkmark on my list.

The Remote Access policy is the last policy that could cause repercussions to an employee that installed the *lokid* binary on a company computer. Loki is capable of many things including the capability of gaining a root shell from a remote location. Our remote access policy clearly states that "no remote access is available, or shall be granted to employees for reasons of security." The

existence of the Loki binary anywhere on the corporate network would be a clear violation of the Remote Access policy and could lead to strict disciplinary action. But the binary's mere existence would probably not be enough to inflict any harsh disciplinary action upon the employee. There must be substantial proof that the application was actually executed and used in a manner that goes against the corporate policy.

Proof of Execution

There are a number of methods by which an experienced administrator could gather proof that the *lokid* binary had indeed been executed. And if the administrator has access to some additional resources, he could easily discover some important facts about the hacker and what his or her intentions really were. The first step, however, is discovering that something is even amiss.

In most cases the first signs of the implantation of a rogue binary are discovered by an IDS. IDS stands for Intrusion Detection System. An IDS can be implemented either as a system that monitors traffic on the network or to monitor changes on a specific host.

A NIDS, short for network intrusion detection system, is a network sniffer that looks for specific signatures or abnormal traffic sent over the wire. This traffic is, in many cases regarded as hostile and is often transmitted by exploits or applications that are known to be affiliated with hacker attacks. Once a signature or an anomaly is recognized the NIDS will, in most cases, send an alert to the administrator responsible for the monitored network in order for the administrator to take action against the intrusion.

There are some NIDS that will go beyond notification and attempt to block the intrusion using a number of possible methods. These methods include the interaction with the sites firewalls, instructing them to take the proper action. Another method is to incorporate the use of RST packets, packets intended to indicate the rejection of the transmission, which are sent to the supposed source of the attack and are mangled to look like they originated from the victims IP address.

A HIDS, short for Host Intrusion Detection System is an IDS that monitors a specific host. A HIDS is installed directly on the monitored host. There are two types of HIDS systems. One that acts like a NIDS but only monitors the traffic to and from the host it's installed on, or one that monitors the file system for any changes that might take place. This is called a File Integrity Scanner and the most popular example of this is a product called Tripwire. There are some downsides to using a HIDS such as tripwire, however.

One problem concerning a HIDS is that, in most real world situations, only certain parts of the system will be monitored. Take the case of a web server for instance.

On a busy web server, the frequency of file system changes due to web page uploads and modifications could be quite high. For this reason many administrators choose not to monitor those files and directory trees. This inevitably leaves the system open to compromise. Monitoring those directories for executable files is always a possibility but would very likely initiate a good number of false positives.

A File Integrity Scanner scans the target system for files changes by comparing the files MD5 sum and MAC times to those recorded in the scanners database. The scanner also maintains a record of what files should be located in the directories that are scanned. If there are any noticeable differences within the file system the Integrity Scanner will send out an alert to the administrator of the machine so that he or she can take the proper action. The downside of the type of IDS is that once a change has been made to the system, the damage has already been done and in most cases the proper remedy is to rebuild the machine. Loki would be recognized using either of these types of IDS systems. In the case of the NIDS, however, its detection could be somewhat difficult.

Most current NIDS rely on fixed signatures. These signatures are very specific and must match the packet exactly or else the packet will cruise past the NIDS undetected. Loki uses the ICMP data field to transmit its payload. Unfortunately, the data is variable and would not match a specific signature. The clearest sign that the Loki backdoor exists and has been executed would be the high frequency of ICMP packets and most IDS systems, and firewalls for that matter, can be configured to detect and report a high volume of ICMP packets, which could inevitably lead to the detection of the Loki binary.

There is a tool available, called IMON that could be used to validate the existence of the Loki binary. It was written by a person known as Stealth and is a tool to monitor and analyze an ICMP packet stream and provides a means of detection of the Loki backdoor. The author's website is no longer available but the binary is available at http://packetstormsecurity.org/UNIX/IDS/icmp-0.9.tar.gz.

Of course the most substantial proof that the Loki binary exists on one of your servers is actually locating and identifying it on the suspect machine. This is where a HIDS comes in handy. If the Loki daemon has been installed in a location of the server that is being monitored by the HIDS, its existence will eventually be noted. Even if *lokid* has been copied over an existing binary on the system, the fact that the MD5 sum or MAC times have been changed is a good indication that something is wrong. Once the *lokid* binary has been located, a simple analysis of the binary will uncover its true nature.

Once the binary has been discovered, proof of execution could be obtained via its MAC time that could indicate the last time the binary was accessed. With some luck, the binary might already be running. If the hacker didn't take the time to cover their tracks, doing a simple "ps -ax" could reveal its existence. If you

then follow its process ID you can look it up on the /proc directory and grab the binary from its associated directory. In Linux, it would be identified as “exe” within the numbered process directory and would be symlinked to its related binary. In Solaris the memory resident binary can be found in /proc/$PID/object/a.out, where $PID is the process ID number.

During execution *lokid* does not create or modify any files on the system. It utilizes portions of the file system that are often used by other processes. In the case of execution on Linux, it accesses the /etc/ld.so.cache file to help locate certain shared library files on the system. It then accesses the /lib/libc.so library and finally opens /dev/tty and switches directories to /tmp. All of these are used by other processes on the system and would provide little aid in discovering that the *lokid* binary had actually been executed.

Besides using the information gained from this type of experience to better protect your network and your hosts, it can do little to prove who actually committed the crime. But if there is some reasonable indication as to who might have initiated it, there are some questions that can be asked of the subject to further prove his or her guilt beyond a reasonable doubt.

Interview Questions

It could prove difficult to get somebody to confess to an act when there is the slightest indication of unpleasant repercussions. As an investigator it's important to make the subject feel comfortable with his or her answers to the questions that may be asked. Of course every incident needs to be handled in its own unique way. In the case of a *lokid* install the range of repercussions could be quite broad. It all depends on what the binary was actually used for.

In this specific case, I'll portray the user as someone who was experimenting with different methods to remotely access company servers via different types of backdoors. There was evidence found that the user was playing around with multiple types of backdoor programs, and the one he finally got to work was *lokid*. Further evidence showed that the user used the Loki programs to transfer some files from the server to his own workstation. There was plenty of evidence, but not enough to lead to a conviction. We needed a confession, so we set up an interview with the subject.

Before the interview I would attempt to make small talk and ask the question “Have you ever given your password to someone else?” This would allow the user to be candid and most likely honest. If asked that same question after leading into an interview, the user is sure to think of someone he had given his password to. I would later ask him to join me in a relatively quiet and comfortable office or conference room and start the interview. Here is how I would open up:

> "I've noticed some interesting activity on the network lately and it baffled me at first – lots of ping packets. It's funny because we monitor the network all day and this had been going on for almost a week before I realized that it wasn't our host monitor but your workstation. Granted ping is rather benign so I really don't care but why are you pinging the file server so often?"

At this point I have given the impression that I don't know the full story. Hopefully the user would give a rather straightforward answer eluding the actual reason why he would be doing this. His knowledge of the ping packets would be enough to follow through with a scenario in which I would keep the conversation friendly and say:

> "That's cool, but I was wondering if you knew what all this code was at the end of each packet? Normally, ping packets don't look like this."

Now speculating of course, but the user will most likely deny any knowledge of it but realize that he might be in some sort of trouble. I would find this to be the most opportune time to begin bringing up the real reason as to why I called him into the office. I would proceed like this:

> "Did you know that your workstation has a copy of the *loki* client installed?"

If he denies any knowledge of the Loki client, which would most likely be the case, I would begin to try and relate to him in the most sincere manner:

> "I understand the need to learn new things and I understand that the company promotes this. I also understand that you're interested in network security and that it's important to keep up with current methods and trends. I do this kind of stuff every day so you can be honest."

The user would hopefully take you up on the offer of being honest and begin the spill his guts as to why he's experimenting with Loki. If not, I would say:

> "You have already stated that you're responsible for transmitting all of those ping packets. I'm not here to torment you, I'm trying to help you out. Nobody knows about this except for me and your supervisor. I really wouldn't want to make this a bigger deal than it really is. We just want to get to the bottom of this. I know that those packets came from Loki because of the way the packet is formatted. They're not normal ping packets. How about you help yourself out and let us know what's going on."

Let's say at this point the user finally confesses that he has used the Loki client. We still need to find out how he got the Loki daemon on the server. Keeping the conversation friendly and seeming really interested I would simply ask “So how did you get the Loki daemon on the server?”

Hopefully with all the preceding questions and the friendly demeanor, the user will confess to installing the *lokid* binary on the server. This, of course, is not a fail save method of interviewing and anything that the interviewer might do that's out of the ordinary could throw the entire interview process in limbo. As stated earlier, every incident requires a unique method of interviewing and every interviewer has their own style in which they perform an interview. It's up to the interviewer to provide the right atmosphere and tone.

References

Daemon9. “Project Loki.” Phrack Magazine, Volume 7, Issue 49. August 1996. http://www.phrack.com/show.php?p=49&a=6

Daemon9. “LOKI2 (the implementation).” Phrack Magazine, Volume 7, Issue 51. September 1, 1997. http://www.phrack.com/show.php?p=51&a=6

Low, Christopher. “ICMP Attacks Illustrated.” SANS Info Sec Reading Room. December 11, 2001. http://www.sans.org/rr/threats/ICMP_attacks.php

Partsenidis, Chris. “Internet Control Message Protocol.” http://www.firewall.cx/index.php?c=icmp-intro

Smith, J. “Covert Shells.” November 12, 2000. http://www.s0ftpj.org/docs/covert_shells.htm

“Title 18.2. Article 7.1 Computer Crimes.” Code of Virginia. http://nsi.org/Library/Compsec/computerlaw/Virginia.txt

Part 2: Forensic Analysis of a System

The Setup

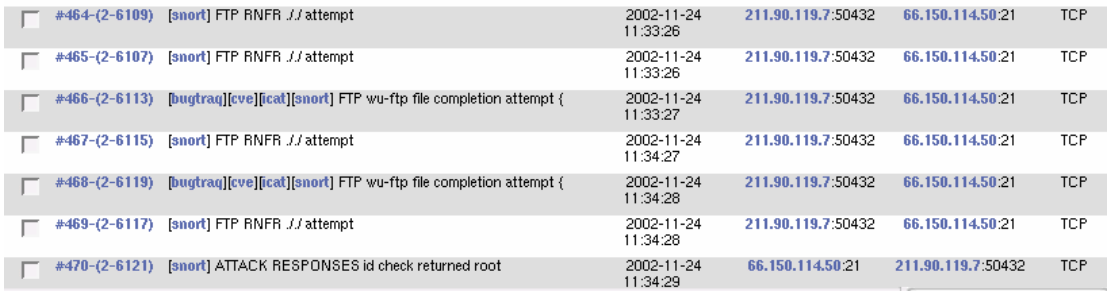
On Wednesday November 20th, 2002 a computer system was connected to the Internet and left unprotected to the world. It was setup with the intention that it be used solely as a honeypot and that it would be analyzed with or without proof of intrusion after a period of two weeks from the date of installation. It's operating system and configuration was kept secret with the exception that it was installed from the distribution CD's and no additional patches or packages were installed. Once the system went live, all monitoring of the workstation took place via the Snort Intrusion Detection System (IDS).

The monitoring system was a Linux workstation with a clean installation of RedHat 8 installed. All of the latest errata were applied, all ports were closed with the exception of SSH, and iptables was configured not to accept or transmit any packets to any foreign hosts. This was to assure that the monitoring workstation was free of vulnerabilities and less likely to get exploited. Snort version 1.90 was installed and set up to monitor only the honeypot and capture every packet that traversed to and from the workstation. Ethereal was installed as a method to view the TCP/IP streams collected by Snort. It captured the first signs of a system compromise where an intruder gained root access a mere 4 days after the installation date of the honeypot.

Signs

The Snort logs looked reasonably routine on the following Monday, uncovering network traffic also seen on most other days. There were a great number of alert signatures mostly consisting of port scan attempts. There were a few open proxy scans and a great deal of FTP attack signatures associated with a common FTP buffer overflow vulnerability. Upon further scrutiny I noticed that at 11:34 AM on November 24th, 2002 Snort delivered the alert "ATTACK RESPONSES id check returned root" as seen in figure 2.1.

Figure 2.1 – Snort log: id check returned root



#464-(2-6109)	[snort] FTP RNFR ./.// attempt	2002-11-24 11:33:26	211.90.119.7:50432	66.150.114.50:21	TCP
#465-(2-6107)	[snort] FTP RNFR ./.// attempt	2002-11-24 11:33:26	211.90.119.7:50432	66.150.114.50:21	TCP
#466-(2-6113)	[bugtraq][cve][icat][snort] FTP wu-ftp file completion attempt {	2002-11-24 11:33:27	211.90.119.7:50432	66.150.114.50:21	TCP
#467-(2-6115)	[snort] FTP RNFR ./.// attempt	2002-11-24 11:34:27	211.90.119.7:50432	66.150.114.50:21	TCP
#468-(2-6119)	[bugtraq][cve][icat][snort] FTP wu-ftp file completion attempt {	2002-11-24 11:34:28	211.90.119.7:50432	66.150.114.50:21	TCP
#469-(2-6117)	[snort] FTP RNFR ./.// attempt	2002-11-24 11:34:28	211.90.119.7:50432	66.150.114.50:21	TCP
#470-(2-6121)	[snort] ATTACK RESPONSES id check returned root	2002-11-24 11:34:29	66.150.114.50:21	211.90.119.7:50432	TCP

The attack signatures that led up to this event referred to a common WU-FTPD file globbing heap corruption vulnerability. WU-FTPD is an FTP (File Transfer Protocol) daemon developed and maintained at Washington University. File globbing is the ability for a "user to specify multiple file names and locations

using typical shell notation." [3] For example, if a user were to type 'get file[0-9].zip' at the FTP prompt, he would initiate the transfer of file0.zip, file1.zip, file2.zip and so forth up to and including file9.zip all using just one single globbing command. This vulnerability focuses on a known exploit based on WU-FTPD's built in file globbing capabilities which can enable an attacker to run arbitrary code on the victim computer with the same privileges as WU-FTPD.

I took this opportunity to review the full FTP stream using Ethereal. The FTP stream can be seen in figure 2.2. Timestamps were added to show how much time had passed during the exploit and labels were added to identify what each side of the connection sent during that time period. The stream shows that after the ftp vulnerability was successfully exploited, the connection timed out and the attacker took no other action.

Figure 2.2 – Ethereal: FTP stream

```
11:30:08 Victim:  220 scoop.domain.com FTP server ready
11:30:08 Intruder:      USER ftp
11:30:08 Victim:  331 Guest login ok, send your complete
11:30:08 Intruder:      PASS mozilla@
11:30:08 Victim:  230 Guest login ok, access restrictions apply
11:30:09 Intruder:      RNFR ./.
11:30:09 Victim:  350 File exists, ready for destination name
......
11:33:26 Intruder:      PWD
11:33:26 Victim:  257 "/" is current directory.
11:33:26 Intruder:      CWD 000000000000000000000000000000000000000550
000000000000000000000000000000000000000
11:33:27 Intruder:      CWD ~/{.,.,.,.}
11:33:27 Victim:  550 /home/ftp/.: No such file or directory
11:34:27 Intruder:      CWD .
11:34:27 Victim:  250 CWD command successful.
11:34:27 Intruder:      RNFR ./.
```



```
11:34:27 Victim:  350 File exists, ready for destination name
11:34:27 Intruder:      CWD 735073
11:34:27 Victim:  550 735073: No such file or directory.
11:34:27 Intruder:      CWD 73507
11:34:27 Victim:  550 73507: No such file or directory.
11:34:28 Intruder:      RNFR .
11:34:29 Victim:  350 File exists, ready for destination name
11:34:29 Intruder:      RNFR ./.
11:34:29 Victim:  350 File exists, ready for destination name
11:34:29 Intruder:      CWD ~{
11:34:29 Intruder:      unset HISTFILE;id;uname -a;
11:34:29 Victim:  uid=0(root) gid=0(root) egid=50(ftp) groups=0(root)
11:34:29 Victim:  Linux scoop.domain.com 2.0.36 #1 Tue Oct 1
11:48:04 Victim:  421 Timout (900 seconds): closing control
```

As I reviewed the Snort logs further I found that the same WU-FTP globbing exploit was executed a number of hours after the first attempt. It had the same signature as the exploit that was executed at 11:30 AM, however, after the second attempt Snort revealed an incorrect login attempt via telnet at 16:43 as seen in figure 2.3.

Figure 2.3 – Snort log: TELNET login incorrect

#826-(2-6485)	[bugtraq][cve][icat][snort] FTP wu-ftp file completion attempt {	2002-11-24 16:40:44	208.177.157.114:4812	66.150.114.50:21	TCP
#827-(2-6493)	[bugtraq][snort] FTP command overflow attempt	2002-11-24 16:40:48	208.177.157.114:4812	66.150.114.50:21	TCP
#828-(2-6491)	[bugtraq][cve][icat][snort] FTP wu-ftp file completion attempt {	2002-11-24 16:40:48	208.177.157.114:4812	66.150.114.50:21	TCP
#829-(2-6489)	[snort] FTP RNFR ./../ attempt	2002-11-24 16:40:48	208.177.157.114:4812	66.150.114.50:21	TCP
#830-(2-6487)	[snort] FTP RNFR ./../ attempt	2002-11-24 16:40:48	208.177.157.114:4812	66.150.114.50:21	TCP
#831-(2-6498)	[arachnids][snort] TELNET login incorrect	2002-11-24 16:43:39	66.150.114.50:23	213.233.106.180:1910	TCP
#832-(2-6497)	[arachnids][snort] TELNET login incorrect	2002-11-24 16:43:39	66.150.114.50:23	213.233.106.180:1910	TCP

The attack signature prior to the incorrect login attempt was identical to that which returned 'root' at 11:30 on the same day. This led me to believe that the machine had been compromised and the hacker was now attempting to login using a newly created user account. At this point I decided to listen to the traffic that was currently going to and from the suspect workstation, hoping to get additional evidence that the system had indeed been compromised. The traffic showed that it was running an IRC daemon, which was previously not identified as running on that machine. TCP traffic that was previously collected from the system revealed that the first IRC packets began transferring to and from the server at 17:13 as seen in figure 2.4, a half hour after the incorrect telnet login attempt. Now convinced that the box had been compromised I disconnected the switch, that the system was connected to, from the network and decided to proceed with a forensic audit of the system.

Figure 2.4 – Snort log: First IRC packets

355237	17:13:02.260522	66.150.115.15	66.150.114.50	DNS	Standard query response[Short Frame]
355238	17:13:02.261007	66.150.114.50	195.54.102.4	TCP	1183 > ircd [SYN] Seq=788181081 Ack=0 Win=512 Len=0 MSS=1460
355239	17:13:02.261256	66.150.114.50	213.233.106.180	TCP	40401 > 1927 [PSH, ACK] Seq=1515431292 Ack=1160807050 Win=32
355240	17:13:02.373394	195.54.102.4	66.150.114.50	TCP	ircd > 1183 [SYN, ACK] Seq=961370104 Ack=788181082 Win=2048 L
355241	17:13:02.373575	66.150.114.50	195.54.102.4	TCP	1183 > ircd [ACK] Seq=788181082 Ack=961370105 Win=32120 Len=(
355242	17:13:02.476555	195.54.102.4	66.150.114.50	IRC	Response
355243	17:13:02.477038	195.54.102.4	66.150.114.50	TCP	4730 > auth [SYN] Seq=3519508982 Ack=0 Win=65535 Len=0 MSS=14
355244	17:13:02.477253	66.150.114.50	195.54.102.4	TCP	auth > 4730 [SYN, ACK] Seq=2450817294 Ack=3519508983 Win=327
355245	17:13:02.492641	66.150.114.50	195.54.102.4	TCP	1183 > ircd [ACK] Seq=788181082 Ack=961370148 Win=32120 Len=(
355246	17:13:02.581126	195.54.102.4	66.150.114.50	TCP	4730 > auth [ACK] Seq=3519508983 Ack=2450817295 Win=65535 Ler
355247	17:13:02.581434	195.54.102.4	66.150.114.50	TCP	4730 > auth [PSH, ACK] Seq=3519508983 Ack=2450817295 Win=655
355248	17:13:02.585497	66.150.114.50	195.54.102.4	TCP	auth > 4730 [FIN, ACK] Seq=2450817295 Ack=3519508996 Win=327
355249	17:13:02.597353	195.54.102.4	66.150.114.50	IRC	Response
355250	17:13:02.612652	66.150.114.50	195.54.102.4	TCP	1183 > ircd [ACK] Seq=788181082 Ack=961370181 Win=32120 Len=(
355251	17:13:02.646433	213.233.106.180	66.150.114.50	TCP	1927 > 40401 [ACK] Seq=1160807050 Ack=1515431420 Win=8400 Ler

The Forensic Audit

Upon discovering that the workstation was compromised, I decided that it might be beneficial to conduct a forensic audit. A forensic audit can bring some additional facts to light that would otherwise not have been discovered upon immediately pulling the systems power supply. The knowledge gained from a forensic audit centers around the volatile evidence that is ordinarily lost once a system is shut down. This evidence consists of information gained from the computers network connections, swap space and, most importantly, memory. Some of the benefits include discovering what processes are running on the box, what ports are open and what processes are listening on those ports. Another benefit of a forensic audit is that you have the ability to take a snapshot of the

systems memory, potentially revealing information pertaining to the significance of running processes in relation to the hack or to the hacker.

The audit was initiated on November 25th at 15:30 by connecting a keyboard and monitor to the compromised Linux machine. Upon turning on the monitor it was found that the previous user had not logged off of the terminal. Since the password to the machine was unknown, the fact that someone had already logged in allowed me to swiftly move forward with my investigation without the need to retrieve the root password from the administrator of the machine. The fact that someone had already logged in as root also removed the need to use the systems 'login' binary and associated libraries. This minimized the loss of potential evidence associated with those files. Once on the system the cdrom containing the forensic toolkit was mounted. Mounting is the process of gaining access to a file system, allowing one to peruse through its file structure. It is accomplished by using the mount command on UNIX hosts. New environment and library paths were set up to make sure that only binary and library files from the cdrom would be accessed by the commands that were to be executed. The environment and library paths were changed so that files and libraries on the compromised system would not be accessed or modified in any way. This is to ensure that the results obtained from running the commands are genuine and that the evidence on the system remains untainted. A laptop was connected to the network switch in preparation to receive a number of files transmitted from the compromised system for future analysis.

The laptop was configured to listen on port 2222 using *netcat* to capture any data transmitted from the compromised host. The data transferred was saved to an appropriately named file that corresponded to the type of data and the date it was received. An md5 sum was created and recorded for each of the files created.

An attempt was made to retain as much evidence as possible. The systems memory, which is the most volatile information on a computer, was the first piece of evidence collected and transferred to the laptop. I accomplished this by using the program *dd,* which is a program used to make bit-by-bit images of files, hard drives or even system memory.

After successfully dumping the systems memory onto the remote system, I proceeded by getting a list of what processes were running and what network ports they were utilizing by using the *lsof* utility*. Lsof* is a utility that lists open files on a system. It is also capable of providing information concerning what network connections are related to those open files. After retrieving this information I then got a listing of current or recently disconnected network connections between the compromised system and any foreign hosts using *netstat*. The commands used to gather all this evidence can be seen in figure 2.5.

Barring all temptation to pursue any further investigation I took the final step of pulling the power plug from the wall. Even though pulling the plug could cause

harm to the physical hardware of a computer, the benefits far outweigh the disadvantages during a forensic investigation. Pulling the plug, rather than performing a clean shutdown, leaves much of the data on the hard drive as it was when the power was on. Many things get written, or unwritten from the hard drive when you perform a clean shutdown. In the case of a forensic analysis, it is best to minimize the number or reads and writes to the drive.

Figure 2.5 Commands used during Forensic Audit

The following commands were used to gather system specific information concerning system resources, network connections and other important information. All commands were issued while another system was listening on port 2222 using netcat to receive the output of the commands.

Creation of systems memory image

```
dd if=/dev/mem | nc 192.168.0.10 2222
```

Creation of lsof output (open file listing)

```
lsof | nc 192.168.0.10 2222
```

Creation of lsof net output (open files related to open network connections)

```
lsof –I | nc 192.168.0.10 2222
```

Creation of netstat output (Open or listening network ports)

```
netstat –an | nc 192.168.0.10 2222
```

Gathering Physical Evidence

Following the forensic audit the next step was to gather the systems physical evidence. I began by recording the computer make, model and serial number of the compromised system and labeled it with an identification number of A0001. The location of the compromised system within the office building was recorded along with the building address. The compromised system identified as A0001 was promptly removed from the building and transferred to the lab for further analysis.

After the system was transferred to the forensic lab the machine was opened and the hard drive was removed from the machine. All of the pertinent information concerning the drive was recorded on an evidence tag. This information included the make, model, serial number and disk size. The drive was then placed in the forensic system so that an identical image of the drives partitions could be created. An md5 sum of each of the partitions was created and recorded on the evidence tag. Once the images were complete I removed the hard drive from the forensic system, placed it in an evidence bag along with the evidence tag and labeled the bag with the tag number A0002. The evidence tag information can be seen in figure 2.6.

Figure 2.6 Evidence tag information		
Tag #	Description	
A0001	**Item**	Computer System
	Make	Micron
	Model	SE440BX2
	Serial Number	1569550-0002
	Location	6315 XXXXXXX Drive Suite 250 Alexandria, Va 22312 Office o12
	Notes	Micron Millenium, Pentium 3 500 Mhz
A0002	**Item**	Hard Drive
	Make	IBM
	Model	DTTA-371440
	Serial Number	WK0R7383
	Location	6315 XXXXXXX Drive Suite 250 Alexandria, Va 22312 Office o12
	Notes	IBM IDE 14.4GB Hard drive The following are md5sums of the contained hard drive. The device names (/dev/xxx) were recorded from the forensic workstation. They might be different on other platforms. mount point:/boot (/dev/hdd1) 595fff228bd2cd01c1f31d16770320f8 mount point:/usr (/dev/hdd5) b8b889549758f57e8da9eebb41855882 mount point:/var (/dev/hdd6) 2627e315da141eaec18e2689863504e3 mount point:/home (/dev/hdd7) 0bb7471dabd998efdd1f0b9b786efa15 mount point:/tmp (/dev/hdd8) 6ce0a513935b48c5924a9ac6b6cb0bd9 mount point:/ (/dev/hdd10) 78b13c5f94a9e02ad0a08ca0805fe2a9

The Forensic Workstation

The forensic workstation that was used for the forensic analysis was a Pentium III computer with a clean version of Redhat 8 installed. All of the latest updates

from Redhat were applied. The programs specific to forensic analysis that had been installed were the @stake Sleuth Kit (Task) 1.52 and Autopsy 1.62. Task is a suite of programs specifically written to aide in the investigation of file systems associated with compromised computer systems. The Task tools provide a means for an investigator to glimpse at the file system in ways not possible using other conventional methods. The tools can access both allocated and unallocated space within the file system allowing an investigator to peruse the given file structure on the media as well as deleted data on that media. Task can also create a timeline of the activity on the media concerning both live and deleted files. Autopsy utilizes all of the tools provided by Task but packages their features in an easy to use graphical web-based environment.

Task was installed so that a thorough timeline of file activity could be created and for the ability to recover deleted inodes. Autopsy was installed for its ease of use. However, most of the investigation was conducted using the simple tool *find* to search for evidence pertaining to the system compromise. *Find* is a program that can search through the operating system for files with attributes that you specify during the execution of the tool. These attributes include file type, owner, permissions and many more. If used correctly, find can be a very powerful tool.

An additional package that was installed on the forensic workstation, which does not come packaged in the standard install of Redhat 8 is the tool *stat. Stat* is a utility that is used to reveal information about a specified file. It is capable of gathering a files MAC times, which are the timestamps of when a file got modified (M), accessed (A), or created (C). In addition to this it also gathers other useful information and displays it in a simple and easy to read format.

Image Media

Images of the hard drive partitions (tag # A0002) were created on the forensic workstation. The program *dd* was used to create the images. The images were named according to the device name that the image was collected from and the date the image was created. For example, if the image was collected from device /dev/hdd1 on December 15, 2002, then the file was named `hdd1-12152002.img`. After all the images were collected an md5 hash was generated against the original hard drive partitions and their respective images. The hashes of the partitions were then compared to those of the corresponding images to make certain that they were identical. This process verifies that the images are an exact replica. Figure 2.7 shows a side-by-side comparison of the images and partition md5sums. As demonstrated below the md5 sums of the images are identical to their respective partition.

Figure 2.7 MD5Sum Comparison	
Image md5sum	*Partition md5sum*

`595fff228bd2cd01c1f31d16770320f8 hdd1-12152002.img`
 	
`b8b889549758f57e8da9eebb41855882`
	`595fff228bd2cd01c1f31d16770320f8 /dev/hdd1`
	
	`b8b889549758f57e8da9eebb41855882`

Figure 2.7 MD5Sum Comparison	
hdd5-12152002.img 2627e315da141eaec18e2689863504e3 hdd6-12152002.img 0bb7471dabd998efdd1f0b9b786efa15 hdd7-12152002.img 6ce0a513935b48c5924a9ac6b6cb0bd9 hdd8-12152002.img 78b13c5f94a9e02ad0a08ca0805fe2a9 hdd10-12152002.img	/dev/hdd5 2627e315da141eaec18e2689863504e3 /dev/hdd6 0bb7471dabd998efdd1f0b9b786efa15 /dev/hdd7 6ce0a513935b48c5924a9ac6b6cb0bd9 /dev/hdd8 78b13c5f94a9e02ad0a08ca0805fe2a9 /dev/hdd10

Media Analysis of the Compromised System

After creating the images of the partitions from the hard drive, the hard drive was removed from the machine and placed it in a safe and secure location along with its evidence tag. Copies of the images were then made and checked to make sure they were identical to the original images by comparing their md5 sums. The images were then mounted so that the forensic analysis could begin.

The images were mounted in a way that would assure that they would not be modified during the forensic process. This is done with the use of some options assigned to the mount command when mounting the images. The command that was used was:

```
mount -o ro,loop,noexec,nodev,noatime /path.to.img
/mnt/hack
```

The command argument "-o" is used to provide the options to the mount command. The option "ro" stands for "read-only." This ensures that nothing will be written to the mounted image. The "loop" option provides a means of mounting an image file that resides on an already mounted file system. "noexec" denies the execution of any binaries on the image. "nodev" is used in order to not interpret special character and block devices which are usually located in the /dev directory. And "noatime" disables the updating of access times to the files being accessed. All of these options used together ensure that the image will not be modified in anyway during the process of forensic analysis.

Commonly Modified files

After all images were mounted in the `/mnt/hack` directory on the forensic workstation, and it looked to be a complete file system, the forensic analysis begun. The first thing that was examined was the type and version of the operating system. This was done by checking the file `/etc/issue`. Its contents revealed the following:

```
Red Hat Linux release 5.2 (Apollo)
Kernel 2.0.26 on an i686
```

I Next looked in the `/etc/password` file to see if any user accounts were created that seemed out of place. I was looking for any entries where a user, other than root, had the user id of 0. Having a user id of 0 gives a user super-user (root) privileges and allows that user to perform almost any function on the machine. It was discovered that a new user with user ID 0 had been created and given the username 'user'. Figure 2.8 shows a portion of the `/etc/password` file.

Figure 2.8 - /etc/passwd snippet

```
gopher:*:13:30:gopher:/usr/lib/gopher-data:
ftp:*:14:50:FTP User:/home/ftp:
nobody:*:99:99:Nobody:/:
user:gR35MgykaNK3g:0:0::/root:/bin/bash
jmk:SaFHJG3Xrqmf.:500:500::/home/jmk:/bin/bash
```

The user accounts highlighted in red are the two accounts created by the cracker, the person who compromised the system, 'user' being the super-user account and 'jmk' being a low privileged account. I then looked for `.bash_history` files in all of the associated user directories, in this case being `/root` and `/home/jmk`. `.bash_history` files are files that contain all of the commands executed by the user. These files are sometimes overlooked by inexperienced crackers and can shed light on exactly what happened on the machine while the user was logged on. Unfortunately, the `.bash_history` files were either erased or disabled so no information could be gathered from them.

I went on to check the log files located in `/var/log`. These files are utilized by most of the processes that run on a Linux machine. The log files are a great resource and can be used to discover a great deal about the processes that log to it. An example would be the file transfer protocol daemon (*ftpd*). Every time a user logs on via FTP, *ftpd* logs the username and the location that the connection originated from in `/var/log/messages`. Every connection is logged in `/var/log/secure`. Even ones that have failed or have been denied are logged. In the case of a compromise, log files can begin to illustrate what actually took place on the machine and how it took place. In many cases, however, the cracker is aware of this and takes the time to delete all or part of the log files. In the case of this forensic investigation, there was no initial evidence insinuating that the logs had been tampered with.

I began by performing a simple search for the username 'jmk' against the logs in `/var/log`. The search showed that the user 'jmk' had been created at 16:49:35 and shortly after, the users password had been changed. About 30 seconds later, user 'jmk' logged on from 106dial180.xnet.ro, and nine minutes after that, an e-mail was delivered to the address jmk@emoka.ro. I performed an identical search for the username 'user' and found that the user had a failed logon attempt at 16:40:09 from 106dial180.xnet.ro showing that 'user' and 'jmk' are the same

person since the connection originated from the same location. Figure 2.9 shows the search results. Further analysis of the log files was performed and is explained in detail in the timeline analysis of this document.

Figure 2.9 - /var/log keyword search

```
>grep jmk *
maillog:Nov 24 16:59:20 scoop sendmail[1894]: QAA01813: to=jmk@emoka.ro,
ctladdr=root (0/0), delay=00:00:09, xdelay=00:00:09, mailer=esmtp,
relay=mx.emoka.ro. [217.156.96.21], stat=Sent (ok 1038179091 qp 23905)
messages:Nov 24 16:49:35 scoop adduser[1432]: new group: name=jmk,
gid=500
messages:Nov 24 16:49:35 scoop adduser[1432]: new user: name=jmk,
uid=500, gid=500, home=/home/jmk, shell=/bin/bash
messages:Nov 24 16:49:40 scoop PAM_pwdb[1433]: password for (jmk/500)
changed by ((null)/0)
messages:Nov 24 16:50:22 scoop PAM_pwdb[1436]: password for (jmk/500)
changed by ((null)/0)
messages:Nov 24 16:50:52 scoop PAM_pwdb[1438]: (login) session opened
for user jmk by (uid=0)
messages:Nov 24 16:50:52 scoop login[1438]: LOGIN ON ttyp0 BY jmk FROM
106dial180.xnet.ro
messages:Nov 24 16:50:52 scoop PAM_pwdb[1438]: (login) session closed
for user jmk

>grep user *
Nov 24 16:40:09 scoop login: FAILED LOGIN 1 FROM 106dial180.xnet.ro FOR
user, Authentication failure
```

SUID and GUID files

After reviewing the log files the next step was to analyze the file system and review its activity. I began by searching for SUID or GUID files that seemed out of place. SUID and GUID permissions allow any user executing the command to raise their privileges to those of the files owner or group. This can be very dangerous especially if the 'root' user owns the file. This has the potential to allow anyone who executes the file to gain root privileges. To locate SUID or GUID files I executed the following command in the `/mnt/hack` directory:

```
find . \( -perm -004000 -o -perm -002000 \) -type f -ls
```

Everything seemed normal except for the entry

```
28809   16 -rws--x--x   1 root     root        15284 Oct 14
1998 ./usr/lib/.x
```

Upon doing a strings analysis on the binary `.x`, I found it to be the login utility usually located in `/bin/login`. A strings analysis is when you take a binary file, which is mostly composed of unreadable characters, and extract just the readable strings from that binary. This will, in many cases, reveal clues as to what the binary might be used for. In the case `/usr/bin/.x` the stings analysis clearly showed that the file was the *login* binary. Utilizing `/usr/lib/.x` with

SUID privileges would allow anyone to gain super-user access to the machine without explicitly login on as root.

I went on to look for directories that begin with a period(.). Beginning a directory or a file name with a period deems that object hidden. In directories other than a users home directory, the use of a period as the first character of a directory name is very rare. In most cases these directories will only exist if someone is trying to hide something as in the case of a system compromise. The command to find these types of directories looks like this:

```
find /dir -name ".*" -type d -printf "%Tc %k %h/%f\n"
```

Upon running this command I found a short yet substantial list, which can be seen in figure 2.10.

Figure 2.10 Directories beginning with a period (.)

```
Sun 24 Nov 2002 05:02:38 PM EST     11/24/02    17:03:40
      ./home/jmk/..
Sun 24 Nov 2002 04:59:08 PM EST     11/25/02    04:02:32
      ./usr/X11R6/lib/X11/fonts/misc/   /.x
```

I looked in `/home/jmk/..\ / (the „\ „ signifies an escaped space)` and found what appeared to be an web daemon directory named httpd. After further investigation I found the application psyBNC inside the httpd directory. PsyBNC is described as an IRC bouncer or proxy that can provide anonymous connections to IRC servers. The README file inside the httpd directory had clear instructions as to how to build in configure the psyBNC daemon. The actual psyBNC executable was named *httpd*, most likely in an attempt to make it less noticeable by the system administrator. A strings analysis demonstrated that the file was indeed psyBNC. Figure 2.11 shows a snippet of the strings analysis. More information about psyBNC can be obtained from http://www.psychoid.lam3rz.de.

Figure 2.11 – PsyBNC strings analysis snippet

```
(%s)!psyBNC@lam3rz.de PRIVMSG %s :%s
startdialogues
stopdialogues
senddialoguequery
*!*@*
CHANNEL
CHANTOPIC
CHANMODE
CHANKEY
CHANLIMIT
CHANUSER%d
%s!%s@%s|%c
#&+!
Undef.
:-psyBNC!psyBNC@lam3rz.de PRIVMSG %s :BHELP User defined Aliases:
:-psyBNC!psyBNC@lam3rz.de PRIVMSG %s :BHELP   %-15s - %s
```

```
:%s(%s!DCCChat@%s PRIVMSG %s :[%d] %s
```

In the directory `/usr/X11R6/lib/X11/fonts/misc/\ \ \ /.x` I found the program adore, which is a Linux kernel module (lkm) that has the ability to hide files, directories and processes from view. This program is not to be confused with the Red Worm, which went by the same name. It came nicely wrapped with installation instructions that clearly state that compilation produces an lkm. There was no evidence that compilation of the binary was successful.

`The /usr/X11R6/lib/X11/fonts/misc/\ \ \ /` directory revealed its own archive of tools installed by the cracker. A number of executables were found in this directory and will be explained in more detail later in this document.

Files newer than /tmp/install.log

A search for executable files owned by the root user and newer than the install.log came up with the following results:

```
Sun 24 Nov 2002 04:59:08 PM EST     11/24/02    16:59:08
      ./usr/X11R6/lib/X11/fonts/misc/   /dan1
Sun 24 Nov 2002 04:59:08 PM EST     11/24/02    16:59:08
      ./usr/X11R6/lib/X11/fonts/misc/   /dan2
Sun 24 Nov 2002 04:59:09 PM EST     11/24/02    16:59:09
      ./etc/rc.d/rc.sysinit
```

These results were obtained by executing the following command from the `/mnt/hack` directory:

```
find . -newer /tmp/install.log -type f -user root \
-perm +111 -printf "%Tc %k %h/%f\n" |sort
```

The files `dan1` and `dan2` within the `/usr/X11R6/lib/X11/fonts/misc/\ \ \ /` directory are identical to one another. A string search revealed that they are both *sshd* binaries and reference some odd files in the `/usr/lib` directory, `.sdc` and `.shk`. The `/usr/lib/.sdc` file turned out to be *sshd*'s server configuration file and `/usr/lib/.shk` is a ssh private key file. The *sshd* configuration file references the daemon's host key at `/usr/lib/.shk2` which also exists. I decided to do a search for other hidden files in `/usr/lib` and discovered one additional file, `.srs`, which could not immediately be identified. The following is a listing of all the files beginning with a period (.) in `/usr/lib`.

```
[root@forensics01 lib]# find -name ".?*" -type f -maxdepth 1 -ls
 28809   16 -rws--x--x   1 root     root        15284 Oct 14  1998 ./.x
 28810    1 -rw-r--r--   1 root     root          998 Mar  2  2002
./.sdc
 28811    1 -rw-------   1 root     root          541 Dec 11  2001
./.shk
 28812    1 -rw-------   1 root     ftp           523 Nov 24 16:59
```

```
./.shk2
 28813    1 -rw-------   1 root     ftp           512 Nov 24 16:59
./.srs
```

Modified Executables
Next I ran the following command from the `/mnt/hack` directory:

```
find . -type f -user root -perm +111 -printf "%T@ %k
%h/%f\n" |sort
```

This command prints out the modification dates along with its associated executable. Figure 2.12 shows a snippet of the results of this command.

Figure 2.12 – Modified Executables

```
984096102   03/08/01    19:01:42    ./usr/X11R6/lib/X11/fonts/misc/
/read
988497316   04/28/01    18:35:16    ./usr/X11R6/lib/X11/fonts/misc/
/cl
991221294   05/30/01    07:14:54    ./usr/X11R6/lib/X11/fonts/misc/
/wroot
1003240720  10/16/01    09:58:40    ./usr/X11R6/lib/X11/fonts/misc/
/scan
1006869671  11/27/01    09:01:11    ./home/ftp/raven/chattr
1007931233  12/09/01    15:53:53    ./home/ftp/raven/move
1008532089  12/16/01    14:48:09    ./usr/X11R6/lib/X11/fonts/misc/
/sc
1008532101  12/16/01    14:48:21    ./usr/X11R6/lib/X11/fonts/misc/
/statdx
1008532104  12/16/01    14:48:24    ./usr/X11R6/lib/X11/fonts/misc/   /v
1008532107  12/16/01    14:48:27    ./usr/X11R6/lib/X11/fonts/misc/
/write
1008532111  12/16/01    14:48:31    ./usr/X11R6/lib/X11/fonts/misc/
/wscan
1008532115  12/16/01    14:48:35    ./usr/X11R6/lib/X11/fonts/misc/
/wted
1008532131  12/16/01    14:48:51    ./home/ftp/raven/fix
1008534111  12/16/01    15:21:51    ./home/ftp/raven/encrypt
1008534111  12/16/01    15:21:51    ./usr/bin/md5sum
1008534111  12/16/01    15:21:51    ./usr/sbin/lsof
1008535262  12/16/01    15:41:02    ./lib/libproc.so.2.0.6
1008537314  12/16/01    16:15:14    ./bin/login
1010924156  01/13/02    07:15:56    ./home/ftp/raven/startfile
1012384774  01/30/02    04:59:34    ./usr/X11R6/lib/X11/fonts/misc/
/sl2
1014202746  02/20/02    05:59:06    ./home/ftp/raven/7350wurm
1014383422  02/22/02    08:10:22    ./home/ftp/raven/clean
1015153751  03/03/02    06:09:11    ./home/ftp/raven/patch
1015154632  03/03/02    06:23:52    ./home/ftp/raven/lg
1023197225  06/04/02    09:27:05    ./etc/rc.d/init.d/init
1023197312  06/04/02    09:28:32    ./home/ftp/raven/install
1026864962  07/16/02    20:16:02    ./home/ftp/raven/mailme
1038175148  11/24/02    16:59:08    ./usr/X11R6/lib/X11/fonts/misc/
/dan1
1038175148  11/24/02    16:59:08    ./usr/X11R6/lib/X11/fonts/misc/
```

Figure 2.12 – Modified Executables

```
/dan2
1038175149  11/24/02    16:59:09    ./etc/rc.d/rc.sysinit
```

Even though not all of the modification dates seem to represent recent changes to the system, they are still important. The only files whose modification date represents the day the hack occurred are the last three files on the list, `dan1`, `dan2` and `rc.sysinit`. All of the other files listed give the impression that they were modified well before the system was compromised. However, all of files listed are suspicious. Again, the `/usr/X11R6/lib/X11/fonts/misc/\ \ \ /` directory is found along with a number of different files within that directory. The list also contains a number of system files that have the same modification date as many of the files within that rogue directory. The following are system files that have been modified from its original version:

```
/bin/login
/usr/sbin/lsof
/usr/bin/md5sum
/lib/libproc.so.2.0.6
```

This implicates the use of a rootkit, a group or system files meant to act as the original system files, but modified to hide the fact that the system has been compromised.

The search for modified files also shows that the directory `/home/ftp` had been used during the time of the incident. Upon looking in that directory I found the file `strike.tgz`. I copied `strike.tgz` to my working directory for future analysis. The analysis of `strike.tgz` will be discussed later in this document.

Newly create file listing

A search for newly created files produced quite a long list. The ones that stood out the most are listed in figure 2.13. It uncovered more evidence that a root kit was indeed installed. The following command was executed in the `/mnt/hack` directory to create the listing:

```
find . -printf "%C@ \t%h/%f\n" |sort
```

Some of the items found with this command were system files that were overwritten by the rootkit and previously not listed. These files are:

```
/bin/netstat
/bin/ps
/sbin/ifconfig
/bin/ls
/usr/bin/dir
/usr/bin/du
/usr/bin/killall
/usr/bin/pstree
```

```
/usr/bin/top
/usr/bin/vdir
```

A strings analysis of some of the files shows that they reference newly created files in `/usr/include`. The binary `/bin/ls` references `/usr/include/file.h`. The binaries `/usr/bin/pstree` and `/usr/bin/killall` reference `/usr/include/proc.h`.

Figure 2.13 – Newly created files

```
1038174058  11/24/02    16:40:58    ./var/lib/rpm/conflictsindex.rpm
1038174058  11/24/02    16:40:58    ./var/lib/rpm/fileindex.rpm
1038174058  11/24/02    16:40:58    ./var/lib/rpm/groupindex.rpm
1038174058  11/24/02    16:40:58    ./var/lib/rpm/nameindex.rpm
1038174058  11/24/02    16:40:58    ./var/lib/rpm/providesindex.rpm
1038174058  11/24/02    16:40:58    ./var/lib/rpm/requiredby.rpm
1038174058  11/24/02    16:40:58    ./var/lib/rpm/triggerindex.rpm
...
1038175147  11/24/02    16:59:07    ./bin/netstat
1038175147  11/24/02    16:59:07    ./bin/ps
1038175147  11/24/02    16:59:07    ./lib
1038175147  11/24/02    16:59:07    ./lib/libproc.so.2.0.6
1038175147  11/24/02    16:59:07    ./sbin
1038175147  11/24/02    16:59:07    ./sbin/ifconfig
1038175147  11/24/02    16:59:07    ./usr/bin/md5sum
1038175147  11/24/02    16:59:07    ./usr/sbin
1038175147  11/24/02    16:59:07    ./usr/sbin/lsof
1038175148  11/24/02    16:59:08    ./bin
1038175148  11/24/02    16:59:08    ./bin/login
1038175148  11/24/02    16:59:08    ./bin/ls
1038175148  11/24/02    16:59:08    ./etc/ftpusers
1038175148  11/24/02    16:59:08    ./etc/rc.d/rc0.d
1038175148  11/24/02    16:59:08    ./etc/rc.d/rc1.d
1038175148  11/24/02    16:59:08    ./etc/rc.d/rc2.d
1038175148  11/24/02    16:59:08    ./etc/rc.d/rc3.d
1038175148  11/24/02    16:59:08    ./etc/rc.d/rc4.d
1038175148  11/24/02    16:59:08    ./etc/rc.d/rc5.d
1038175148  11/24/02    16:59:08    ./etc/rc.d/rc6.d
1038175148  11/24/02    16:59:08    ./usr/bin
1038175148  11/24/02    16:59:08    ./usr/bin/dir
1038175148  11/24/02    16:59:08    ./usr/bin/du
1038175148  11/24/02    16:59:08    ./usr/bin/killall
1038175148  11/24/02    16:59:08    ./usr/bin/pstree
1038175148  11/24/02    16:59:08    ./usr/bin/top
1038175148  11/24/02    16:59:08    ./usr/bin/vdir
1038175148  11/24/02    16:59:08    ./usr/include
1038175148  11/24/02    16:59:08    ./usr/include/file.h
1038175148  11/24/02    16:59:08    ./usr/include/hosts.h
1038175148  11/24/02    16:59:08    ./usr/include/proc.h
1038175148  11/24/02    16:59:08    ./usr/lib
1038175148  11/24/02    16:59:08    ./usr/lib/.sdc
1038175148  11/24/02    16:59:08    ./usr/lib/.shk
1038175148  11/24/02    16:59:08    ./usr/lib/.shk2
1038175148  11/24/02    16:59:08    ./usr/lib/.srs
1038175148  11/24/02    16:59:08    ./usr/lib/.x
```

Figure 2.13 – Newly created files

...

The three files created in `/usr/include` are associated with the rootkit. The files `file.h`, `hosts.h` and `proc.h` all seem to contain lists of objects that the cracker wants to hide from the systems administrator and, as stated earlier, are referenced by some of the system binaries that were overwritten. The lists contained within those files can be seen in figure 2.14.

Files in `/var/lib/rpm` are files associated with the *rpm* database and have been modified. Their modification might signify that the cracker has installed RPM's. Files in `/etc/rc.d` had also been modified. The files within that directory are responsible for starting services during system boot. Modification of those files leads one to conclude that the cracker is starting his/her own services upon a system reboot.

Figure 2.14 – Contents of file.h, hosts.h and proc.h

file.h		Hosts.h	proc.h
.shk2	move	1 217.156	2 sl2
7350wu	lg	1 217.10	2 b
startwu	v	1 213.233	2 startstatd
startstat	write	1 xlogic.ca	2 startwu
d	.x	1 limp-bizkit.ro	2 awu
awu	.x.tgz	2 217.156	2 arpc
arpc	init	2 217.10	2 xbnc
essh.tgz	initd	2 213.233	3 dan1
.sdc	wroot	2 microrom.ro	3 dan2
.shk	statdx	3 25330	2 scan
targets	scan	3 6696	2 write
wu	read	3 20202	2 assh
sc	hosts.h	4 25330	2 sssh
wted	proc.h	4 6667	2 xmech
wscan	file.h	4 6666	2 start
x2	cl	4 6696	2 xlx2
assh	xC.o	4 20202	
sssh	cleaner.o		
start	freedom.t		
sl2	gz		
srd0	eggdrop		
remove	dan1		
	dan2		

Command usage timeline

Listing files by access time (atime) can give you an idea of what commands were executed and in what order. The command that will produce this list is:

```
find /dir -type f -perm +111 -printf "%A@ \t%h/%f\n" |sort
```

Figure 2.15 is an abridged list of the file access times during the time of the system compromise. In this list we see that `/bin/rpm` was the first binary accessed at 16:40:55. This shows that an rpm might have been installed to patch the vulnerable version of WU-FTPD. But that is just an assumption. It could simply mean that the RPM database was queried to find out information concerning an installed RPM package.

Looking at the command usage timeline you will notice that the between 16:59:03 and 16:59:11, a lot of files were accessed during this short time span. This is most likely because a script was run to perform a number of tasks, presumably the installation of the rootkit and other hacker tools like *linsniffer* and *7350wurm* *7350wurm* was written by team teso (http://www.team-teso.net) and is a WU-FTPD server exploit. This exploit is most likely the same exploit used to compromise this particular system. At 16:59:03 you can see that the file *7350wurm* was accessed, probably to be copied to its final location. The source for 7350wurm can be downloaded from Packetstorm Security (http://209.100.212.5/cgi-bin/search/search.cgi?searchvalue=7350wurm&type=archives&%5Bsearch%5D.x=0&%5Bsearch%5D.y=0) At 16:59:08 `/etc/rc.d/init.d/portmap` was accessed either to start or stop the portmap process. Upon running *stat* against the *portmap* file it was noted that its modification time appeared to be unchanged. At 17:00:46 ftp was accessed to download a file and shortly after at 17:02:28, the downloaded file was untarred in `/home/jmk/..\ /httpd.` It was then compiled at 17:03:29. As stated earlier the `/home/jmk/..\ /httpd` directory contained psyBNC, the IRC bouncer program.

Figure 2.15 – Command usage timeline (abridged)

```
1038174055  11/24/02    16:40:55    ./bin/rpm
1038174575  11/24/02    16:49:35    ./usr/sbin/useradd
1038174619  11/24/02    16:50:19    ./usr/bin/passwd
1038174644  11/24/02    16:50:44    ./usr/lib/.x
1038174932  11/24/02    16:55:32    ./usr/bin/locate
1038175143  11/24/02    16:59:03    ./home/ftp/raven/7350wurm
1038175143  11/24/02    16:59:03    ./home/ftp/raven/chattr
1038175143  11/24/02    16:59:03    ./home/ftp/raven/fix
1038175143  11/24/02    16:59:03    ./home/ftp/raven/logn
1038175143  11/24/02    16:59:03    ./usr/bin/md5sum
1038175143  11/24/02    16:59:03    ./usr/sbin/lsof
1038175143  11/24/02    16:59:03    ./usr/X11R6/lib/X11/fonts/misc/
/read
1038175143  11/24/02    16:59:03    ./usr/X11R6/lib/X11/fonts/misc/
/sc
1038175143  11/24/02    16:59:03    ./usr/X11R6/lib/X11/fonts/misc/
/scan
1038175143  11/24/02    16:59:03    ./usr/X11R6/lib/X11/fonts/misc/
/sl2
1038175143  11/24/02    16:59:03    ./usr/X11R6/lib/X11/fonts/misc/
/statdx
```

Figure 2.15 – Command usage timeline (abridged)

```
1038175143  11/24/02    16:59:03    ./usr/X11R6/lib/X11/fonts/misc/   /v
1038175143  11/24/02    16:59:03    ./usr/X11R6/lib/X11/fonts/misc/
/write
1038175143  11/24/02    16:59:03    ./usr/X11R6/lib/X11/fonts/misc/
/wroot
1038175143  11/24/02    16:59:03    ./usr/X11R6/lib/X11/fonts/misc/
/wscan
1038175143  11/24/02    16:59:03    ./usr/X11R6/lib/X11/fonts/misc/
/wted
1038175147  11/24/02    16:59:07    ./bin/netstat
1038175147  11/24/02    16:59:07    ./home/ftp/raven/encrypt
1038175147  11/24/02    16:59:07    ./usr/bin/dir
1038175147  11/24/02    16:59:07    ./usr/bin/top
1038175147  11/24/02    16:59:07    ./usr/bin/vdir
1038175148  11/24/02    16:59:08    ./bin/ps
1038175148  11/24/02    16:59:08    ./bin/pwd
1038175148  11/24/02    16:59:08    ./bin/usleep
1038175148  11/24/02    16:59:08    ./etc/rc.d/init.d/functions
1038175148  11/24/02    16:59:08    ./etc/rc.d/init.d/portmap
1038175148  11/24/02    16:59:08    ./home/ftp/raven/lg
1038175148  11/24/02    16:59:08    ./home/ftp/raven/move
1038175148  11/24/02    16:59:08    ./home/ftp/raven/remove
1038175148  11/24/02    16:59:08    ./lib/libproc.so.2.0.6
1038175148  11/24/02    16:59:08    ./sbin/chkconfig
1038175148  11/24/02    16:59:08    ./usr/bin/perl
1038175148  11/24/02    16:59:08    ./usr/X11R6/lib/X11/fonts/misc/
/.x/configure
1038175149  11/24/02    16:59:09    ./home/ftp/raven/check
1038175149  11/24/02    16:59:09    ./home/ftp/raven/startfile
1038175149  11/24/02    16:59:09    ./sbin/ifconfig
1038175149  11/24/02    16:59:09    ./usr/bin/chattr
1038175150  11/24/02    16:59:10    ./bin/ping
1038175151  11/24/02    16:59:11    ./etc/rc.d/init.d/init
1038175151  11/24/02    16:59:11    ./home/ftp/raven/clean
1038175151  11/24/02    16:59:11    ./home/ftp/raven/install
1038175151  11/24/02    16:59:11    ./home/ftp/raven/mailme
1038175151  11/24/02    16:59:11    ./home/ftp/raven/patch
1038175151  11/24/02    16:59:11    ./sbin/killall5
1038175151  11/24/02    16:59:11    ./sbin/route
1038175151  11/24/02    16:59:11    ./usr/bin/killall
1038175151  11/24/02    16:59:11    ./usr/X11R6/lib/X11/fonts/misc/
/cl
1038175151  11/24/02    16:59:11    ./usr/X11R6/lib/X11/fonts/misc/
/dan1
1038175151  11/24/02    16:59:11    ./usr/X11R6/lib/X11/fonts/misc/
/dan2
1038175151  11/24/02    16:59:11    ./usr/X11R6/lib/X11/fonts/misc/
/.x/start
1038175197  11/24/02    16:59:57    ./usr/bin/w
1038175210  11/24/02    17:00:10    ./bin/mkdir
1038175246  11/24/02    17:00:46    ./usr/bin/ftp
1038175348  11/24/02    17:02:28    ./bin/tar
1038175348  11/24/02    17:02:28    ./home/jmk/.. /httpd/psybncchk
1038175419  11/24/02    17:03:39    ./bin/uname
1038175419  11/24/02    17:03:39    ./home/jmk/.. /httpd/tools/autoconf
```

Figure 2.15 – Command usage timeline (abridged)

```
1038175419  11/24/02    17:03:39    ./home/jmk/.. /httpd/tools/chkenv
1038175419  11/24/02    17:03:39    ./home/jmk/.. /httpd/tools/chksock
1038175419  11/24/02    17:03:39    ./home/jmk/.. /httpd/tools/convconf
1038175420  11/24/02    17:03:40    ./bin/echo
1038175420  11/24/02    17:03:40    ./home/jmk/.. /httpd/makesalt
1038175420  11/24/02    17:03:40    ./home/jmk/.. /httpd/tools/chkresolv
1038175420  11/24/02    17:03:40    ./home/jmk/.. /httpd/tools/chktime
1038175420  11/24/02    17:03:40    ./usr/bin/as
1038175420  11/24/02    17:03:40    ./usr/bin/gcc
1038175420  11/24/02    17:03:40    ./usr/bin/i386-redhat-linux-gcc
1038175420  11/24/02    17:03:40    ./usr/bin/ld
1038175420  11/24/02    17:03:40    ./usr/bin/make
1038175420  11/24/02    17:03:40    ./usr/bin/strip
1038175561  11/24/02    17:06:01    ./usr/bin/pico
1038175707  11/24/02    17:08:27    ./home/jmk/.. /httpd/httpd
1038181584  11/24/02    18:46:24    ./usr/sbin/in.telnetd
1038181586  11/24/02    18:46:26    ./bin/login
```

Non-block and non-character mode devices

A search for non-block and non-character mode devices in the `/dev` directory did not uncover anything out of the ordinary.

Analysis of strike.tgz

The tar archive `strike.tgz` was created relatively early during the system compromise. Even though the archive is owned by root, the file was not found by any of the file searches performed towards the beginning of the forensic process. This is due to the fact that the file searches were configured to look for files with at least one execute bit set. Since tar files are not executables, they will usually not have the execution bit enabled. Fortunately it was found through other methods and its contents explain a great deal of what actually happened on the machine and the order in which it was processed.

The tar file was copied to the `/root` directory of the forensic workstation and extracted using the following command:

```
tar -xvzf strike.tgz
```

This created a directory called `raven` that contained most of the files already found on the system that were installed during the system compromise. The directory listing can be seen in figure 2.16. The first file that I examined was the file called 'install'. It is a shell script that describes exactly how most of the files were installed and in the order in which they were installed.

Figure 2.16 – Directory listing of raven

```
[root@forensics01 raven]# ls -al
total 2200
drwxr-xr-x    3 root     root         4096 Mar  7 13:23 .
drwxr-xr-x    3 root     root         4096 Jan  3 19:02 ..
-rwxr-xr-x    1 root     root       382072 Feb 20  2002 7350wurm
```

Figure 2.16 – Directory listing of raven

```
-rw-r--r--    1 root     root            0 Oct 16  2001 .a
-rw-r--r--    1 root     root          161 Mar  3  2002 .c
-rwxr-xr-x    1 root     root         7144 Nov 27  2001 chattr
-rwxr-xr-x    1 root     root          302 Jan 26  1998 check
-rwxr-xr-x    1 root     root         1345 Apr 28  2001 cl
-rwxr-xr-x    1 root     root          197 Feb 22  2002 clean
-rw-r--r--    1 root     root          120 Mar  3  2002 .d
-rwxr-xr-x    1 root     root        39696 Dec 16  2001 dir
-rwxr-xr-x    1 root     root       114848 Dec 16  2001 du
-rwxr-xr-x    1 root     root        14808 Dec 16  2001 encrypt
-rwxr-xr-x    1 root     root         6648 Dec 16  2001 fix
-rwxr-xr-x    1 root     root        31504 Dec 16  2001 ifconfig
-rw-------    1 root     root        12288 Mar  3  2002 .inetd.conf.swp
-rwxr-xr-x    1 root     root          306 Jun  4  2002 init
-rwxr-xr-x    1 root     root         1120 Jun  4  2002 install
-rwxr-xr-x    1 root     root        21306 Dec 16  2001 killall
-rwxr-xr-x    1 root     root          310 Mar  3  2002 lg
-rwxr-xr-x    1 root     root        37984 Dec 16  2001 libproc.so.2.0.6
-rwxr-xr-x    1 root     root         3744 Dec 16  2001 login
-rwxr-xr-x    1 root     root        14410 Jan 26  1998 logn
-rwxr-xr-x    1 root     root       155462 Dec 16  2001 ls
-rwxr-xr-x    1 root     root        82628 Dec 16  2001 lsof
-rwxr-xr-x    1 root     root          329 Jul 16  2002 mailme
-rwxr-xr-x    1 root     root        31452 Dec 16  2001 md5sum
-rwxr-xr-x    1 root     root         3971 Dec  9  2001 move
-rwxr-xr-x    1 root     root        54152 Dec 16  2001 netstat
-rw-r--r--    1 root     root          250 Mar  3  2002 .p
-rwxr-xr-x    1 root     root         1292 Mar  3  2002 patch
-rwxr-xr-x    1 root     root        62920 Dec 16  2001 ps
-rwxr-xr-x    1 root     root        12340 Dec 16  2001 pstree
-rwxr-xr-x    1 root     root         4060 Mar  8  2001 read
-rwxr-xr-x    1 root     root         3238 Jan 26  1998 remove
-rwxr-xr-x    1 root     root         6532 Dec 16  2001 sc
-rwxr-xr-x    1 root     root          982 Oct 16  2001 scan
-rw-r--r--    1 root     root          998 Mar  2  2002 .sdc
-rw-------    1 root     root          541 Dec 11  2001 .shk
-rwxr-xr-x    1 root     root        16776 Jan 30  2002 sl2
-rwxr-xr-x    1 root     root       710020 Mar  3  2002 sshd
-rw-r--r--    1 root     root          612 Mar  3  2002 sshd_config
-rw-------    1 root     root          523 Jan 26  1998 ssh_host_key
-rw-------    1 root     root          512 Jan 26  1998 ssh_random_seed
-rwxr-xr-x    1 root     root         1244 Jan 13  2002 startfile
-rwxr-xr-x    1 root     root        11472 Dec 16  2001 statdx
-rwxr-xr-x    1 root     root        33992 Dec 16  2001 top
-rwxr-xr-x    1 root     root         4684 Dec 16  2001 v
-rwxr-xr-x    1 root     root       155464 Dec 16  2001 vdir
-rwxr-xr-x    1 root     root         6324 Dec 16  2001 write
-rwxr-xr-x    1 root     root         1187 May 30  2001 wroot
-rwxr-xr-x    1 root     root         6340 Dec 16  2001 wscan
-rwxr-xr-x    1 root     root         6324 Dec 16  2001 wted
drwxr-xr-x    3 30       root         4096 Jan 19  2002 .x
-rw-r--r--    1 root     root        14796 Jan 19  2002 .x.tgz
```


Analysis of the 'install' script

The 'install' script seems to be the heart of the compromise. It is responsible for installing the rootkit, adore, and a number of exploits on the machine. It also coordinates the patching of the machine and the installation of a system backdoor. I will go though the install file line by line, including the scripts referenced by this file, in hopes to account for every file already found on the system and perhaps discover others. The commands are referenced verbatim, followed by a description of the command.

'Install' script
`#!/bin/sh` Execute the script using /bin/sh

`unset HISTFILE`
`unset HISTSAVE`
 	
Disables the use of the .bash_history file, which is written in a users home directory. This file logs every command executed in the bash shell by the user that executed those commands. The .bash_history file is usually the first thing a cracker disables once he has gained access to the system.

`chown root.root *` Changes the ownership of all files in the current directory to user and group root.
`dir='pwd'` Sets the variable $dir to the current working directory, /home/ftp/raven.
`./remove` Execute the 'remove' script, which replaces key system files and installs a backdoor. Refer to 'Analysis of the remove script' for further details.
`./move` Execute the 'move' script. This script seems to remove files associated with previously installed hacks that may currently exist on the compromised machine. This would allow the cracker to have full control of the box and minimize the chances that the system will be taken over by somebody else.
`mkdir -p /usr/X11R6/lib/X11/fonts/misc/"  "/` Create the /usr/X11R6/lib/fonts/misc/" "/ directory.

'Install' script

```
mv -f wted cl .x.tgz \
/usr/X11R6/lib/X11/fonts/misc/"   "/
mv -f statdx write scan sc sl2 wroot wscan v read \
/usr/X11R6/lib/X11/fonts/misc/"   "/
```

Move hacker utilities to the newly created directory.

```
cp -f sshd /usr/X11R6/lib/X11/fonts/misc/"   "/dan1
cp -f sshd /usr/X11R6/lib/X11/fonts/misc/"   "/dan2
```

Copy *sshd* to the newly created directory. Presumably these will be used to open up a backdoor to the system.

```
if [ -f /usr/lib/.shk ];then
     chattr -saui /usr/lib/.shk
     rm -f /usr/lib/.shk
fi
```

As stated earlier .shk is a SSH private key file. This removes any previously installed ssh private key file with the same name from /usr/lib, just in case this exploit has been used before on this system.

```
if [ -f /usr/lib/.sdc ];then
     chattr -saui /usr/lib/.sdc
     rm -f /usr/lib/.sdc
fi
```

As stated earlier .sdc is a SSH server configuration file. This removes any previously installed ssh server configuration file with the same name from /usr/lib, just in case this exploit has been used before on this system.

```
if [ -f /usr/lib/.shk2 ];then
     chattr -ai /usr/lib/.shk2
     rm -rf /usr/lib/.shk2
fi
```

As stated earlier .shk2 is a SSH private key file. This removes any previously installed ssh private key with the same name from /usr/lib, just in case this exploit has been used previously on this system.

```
if [ -f /usr/lib/.srs ];then
     chattr -ai /usr/lib/.srs
     rm -rf /usr/lib/.srs
fi
```

This removes any previously installed .srs file from /usr/lib, just in case this

'Install' script
exploit has been used previously on this system. We find later in the 'install' script that the .srs file is the ssh_random_seed file.

`mv -f .sdc .shk  /usr/lib/`
`chattr +saui /usr/lib/.sdc /usr/lib/.shk`
 	
Copy new SSH server configuration file and SSH private key to the /usr/lib directory.

`cp -f ssh_host_key /usr/lib/.shk2`
`cp -f ssh_random_seed /usr/lib/.srs`
 	
Copy new the ssh_host_key to /usr/lib/.shk2 and ssh_random_seed file to /usr/lib/.sdc.

`touch /usr/X11R6/lib/X11/fonts/misc/"   "/tcp.log` Create the file tcp.log in the /usr/X11R6/lib/X11/fonts/misc/" "/ directory. This file is associated with linsniffer, which has been renamed to 'write'.
`./check` Execute the 'check' script. This script checks to see if /usr/bin/gcc, /usr/bin/make, and necessary include files are installed on the system. If it finds that they are all installed, the script then cd's to the /usr/X11R6/lib/X11/fonts/misc/" "/ directory, untars .x.tgz, cd's to the .x directory and compiles the binaries. As stated earlier, the .x directory contains the adore source code. The compiled binaries have not been found.
`./startfile` Execute the 'startfile' script. This script modifies the /etc/rc.d/rc.sysinit file. It appends the line /etc/rc.d/init.d/init to the end of this file. Refer to 'Analysis of /etc/rc.d/init.d/init for further details.
`./mailme` Executes the 'mailme' script. This script grabs a number of system attributes from the compromised host, places the information in the file /tmp/info and them mails that information to jmk@emoka.ro.
`./clean` Executes the 'clean' script. This script cd's to the /usr/X11R6/lib/X11/fonts/misc/" "/ directory and runs the 'cl' command with the following arguments: yahoo.com,

'Install' script
sshd, 208.158, 209.235, rotind. As stated earlier, the cl command is a log cleaner that goes through the logs in /var/log and removes logs containing the strings that were set as the commands argument.
`./patch` Execute the 'patch' script. This script patches the host from the SSHD 1.2.26-32 vulnerability. It replaces sshd binaries in both, /usr/sbin and in /usr/local/sbin. It also seems to remove certain binaries associated with previous exploits used on the compromised machine and adds an ipchains rule blocking port 32760 through 32769 on the compromised host.
`cd $dir` cd to /home/ftp/raven directory.
`cd ..` cd to /home/ftp directory.
`rm -rf cruel*` Removes any files preceded with the string cruel.
`cd /` cd to the root directory of the machine.
`/etc/rc.d/init.d/init` Run the script /etc/rc.d/init.d/init. Refer to 'Analysis of /etc/rc.d/init.d/init for further details.

Analysis of the 'remove' script

The remove script replaces system files with modified system files and backdoors. The first action the script takes is to grab the md5sum of the system files it is going to replace and places those md5 sums in a file named `.tkmd5`. It then encrypts the `.tkmd5` file and places the encrypted list in `/dev/srd0`. A strings analysis of `/usr/bin/md5sum` did not uncover a reference to the file `/dev/srd0` and the `/dev/srd0` file was not found. The remove script continues by replacing the following system files with modified versions:

`/sbin/ifconfig`

```
/bin/ps
/bin/ls
/bin/netstat
/usr/bin/find
/usr/bin/top
/usr/bin/vdir
/usr/bin/killall
/usr/bin/dir
/usr/bin/md5sum
/usr/sbin/lsof
/lib/libproc.so.2.0.6
/usr/bin/chattr
/usr/bin/pstree
/usr/bin/du
```

The remove script goes on and disables the portmap service, first stopping it and then removing it all together from the `/etc/rc.d` directories. It then goes through the file system and attempts to remove files associated with possible previous compromises. These files include:

```
/dev/caca
/dev/pisu
/dev/dsx
```

Finally, it installs `proc.h`, `hosts.h` and `file.h` in the `/usr/include` directory. As stated earlier, these files are associated with the modified system files and contain lists of objects that the cracker wants to hide from the systems administrator.

Analysis of the '/etc/rc.d/init.d/init' script

The 'init' script placed in `/etc/rc.d/init.d` is responsible for starting up certain services associated with the system compromise. The script can be seen in Figure 2.17. The first services started by the script is `dan1` and `dan2` in the `/usr/X11R6/lib/X11/fonts/misc/\ \ \ /` directory. Remember that earlier it was stated that `dan1` and `dan2` were actually the *sshd* process allowing a user to access the system remotely via secure shell. In this case these services were setup to listen on port 3200 with the use of the "–p" option. Referring back to the netstat log created just prior to shutting the system down, I found no evidence that either service had been started.

The script then goes on to check and see if eggdrop is installed. Eggdrop is what is known as an IRC bot, which is capable of running many different maintenance tasks in relation to the IRC network. If it exists, it starts the eggdrop daemon. In this particular case, eggdrop was not installed. You can find more information about eggdrop at http://www.eggheads.org.

The last portion of the script checks to see if the `.x` directory exists in the `/usr/X11R6/lib/X11/fonts/misc/\ \ \ /` directory. The `.x` directory contains the adore source code. If the `.x` directory is present it moves forward and starts the adore process.

Figure 2.17 - /etc/rc.d/init.d/init script

```
#!/bin/sh

x=`pwd`

cd /usr/X11R6/lib/X11/fonts/misc/"   "/

./dan1 -p 3200 -q
./dan2 -p 3200 -q

if [ -d eggdrop ];then
cd eggdrop
#./xbnc -m egg.conf >> logz
#rm -rf logz
cd ..
fi

if [ -d .x ];then
cd .x >> /dev/null
./start >> /dev/null
./ava i `/sbin/pidof initd` >>/dev/null

fi
cd $x > /dev/null
```

The files of /usr/X11R6/lib/X11/fonts/misc/\ \ \ /

As revealed earier in this document, the `/usr/X11R6/lib/X11/fonts/misc/\ \ \ /` directory contained a number of files installed by the cracker. All of them have something to do with a system hack. Some tools are use to remove evidence that a system has been compromised. Others are exploits that could potential compromise another system. Below is a list of all the files found in the directory with a brief explaination of what the tool does.

cl

Renamed from its original name “sauber” which is a utility that cleans log files by removing entries that contain a given keyword.

read

A perl script that acts as a linsniffer log sorter.

write
Renamed from its original name linsniffer, which is a utility whose main purpose is to capture usernames and passwords.

v
Renamed from its original name vadim, which is a udp distributed denial of service (DDOS) utility.

wroot
shell script, compiles wu-ftp portion of an exploit and runs wscan

wscan
This is a wu-ftpd scanner that looks for vulnerable wu-ftpd servers.

sl2
Renamed from its original name antifoo-net by blizzard, which is based on sl2. SL2 is a DoS tool able to spoof its source address and attack a range or ports on a specific host.

statdx
This is a remote root exploit that is able to compromise vulnerable rpc-statd services.

scan
This is a shell script that compiles statdx hack.

sc
This is a scanner that looks for vulnerable rpc-statd services and runs statdx on those hosts.

wted
This utility allows one to view entries in the /var/log/wtmp file and is able to delete entries by username or hostname.

Deleted File Recovery
I used the utilities *ils* and *icat* to recover deleted files from the compromised file system. I wrote a short shell script based on a code snippet of Thomas Roessler's scan of the month entry, which can be found at http://project.honeynet.org/scans/scan15/proj/t/ . The script can be seen in figure 2.18.

Figure 2.18 – Script to recover deleted inodes

```
#!/bin/bash
mkdir -p /tmp/deleted$1  #Create repository for recovered
```

Figure 2.18 – Script to recover deleted inodes

```
inodes
cd /usr/local/task/bin
./ils -rf linux-ext2 $1| \
awk -F '|' '($2=="f") {print $1}' | \
while read i; do
     /usr/local/task/bin/icat -hf linux-ext2 $1 $i > \
     /tmp/deleted$1/$i; \
     file /tmp/deleted$1/$i >> /tmp/deleted$1/filetypes
done
```

This script gets the inodes of deleted files from the referenced image file using the *ils* utility and creates a directory to act as a repository for the recovered files. It then recovers the inodes using the *icat* utility. Once the recovery is complete, the file types of all of the recovered files are gathered and placing in a file called '`filetypes`' in the associated directory.

There were three files that I was hoping to recover: `.tkmd5` - the file containing the md5sums of the overwritten system binaries, the mail message sent to jmk@emoka.ro and the supposed rpm that was installed. Unfortunately, I was unable to recover any of them. I contribute this to the fact that the scripts used to install the rootkit applied special file attributes to the files with the use of the *chattr* tool. *Chattr* is capable is setting file attributes that cause the file to be "zeroed" before the file is actually deleted. The command `'chattr -saui'` was used frequently during the install script and the scripts that it initiated. The -s option for *chattr* is key since that is the option that causes the zeroing of the files data before it is deleted. The evidence supporting this is that the recovered files from numerous partitions where quite large, but contained absolutely no data. Figure 2.19 shows this. The only files containing data are `16066`, `30123` and `file types`. The rest are completely empty. However notice the file sizes on all other files. File `14107(red)` even exceeds the size of `16066(green)` yet contains nothing. The only file that was recoverable was `psyBNC.3.1.tar`, `which` was at inode 16066. According to the timeline produced by Autopsy, the last reference regarding the inode 16066 (psyBNC.3.1.tar) shows that it was deleted November 24, 2002 at17:02:33 as seen in the following timeline entry:

```
Sun Nov 24 2002 17:02:33 \
312188 ..c -rw-rw-r-- 500      500      16066    <parthdd7.img-dead-
16066>
```

Data proving that an IRC server was running on the machine was found in inode 30123. It contents are listed in figure 2.20. The autopsy timeline shows that it was deleted on November 24, 2002 at 17:10:23 as seen in the following timeline entry:

```
Sun Nov 24 2002 17:10:23
```

```
1556 ma. -rw------- 500      500      30123    <parthdd7.img-dead-
30123>
```

Figure 2.19 – List of recovered inodes

```
[root@forensics01 parthdd7.img]# ls -al
total 2044
drwxr-xr-x    2 root     root         4096 Mar  9 16:17 .
drwxr-xr-x    8 root     root         4096 Mar  9 16:17 ..
-rw-r--r--    1 root     root        16776 Mar  9 16:17 14060
-rw-r--r--    1 root     root         6324 Mar  9 16:17 14061
-rw-r--r--    1 root     root         4684 Mar  9 16:17 14062
-rw-r--r--    1 root     root         1187 Mar  9 16:17 14063
-rw-r--r--    1 root     root         6340 Mar  9 16:17 14064
-rw-r--r--    1 root     root         4060 Mar  9 16:17 14065
-rw-r--r--    1 root     root          120 Mar  9 16:17 14066
-rw-r--r--    1 root     root        11472 Mar  9 16:17 14069
-rw-r--r--    1 root     root         3744 Mar  9 16:17 14070
-rw-r--r--    1 root     root        12340 Mar  9 16:17 14072
-rw-r--r--    1 root     root        62920 Mar  9 16:17 14073
-rw-r--r--    1 root     root        33992 Mar  9 16:17 14074
-rw-r--r--    1 root     root        54152 Mar  9 16:17 14075
-rw-r--r--    1 root     root        31504 Mar  9 16:17 14076
-rw-r--r--    1 root     root          982 Mar  9 16:17 14078
-rw-r--r--    1 root     root         6532 Mar  9 16:17 14079
-rw-r--r--    1 root     root         1345 Mar  9 16:17 14082
-rw-r--r--    1 root     root          161 Mar  9 16:17 14083
-rw-r--r--    1 root     root          250 Mar  9 16:17 14085
-rw-r--r--    1 root     root         6324 Mar  9 16:17 14086
-rw-r--r--    1 root     root       155464 Mar  9 16:17 14087
-rw-r--r--    1 root     root        21306 Mar  9 16:17 14088
-rw-r--r--    1 root     root       155462 Mar  9 16:17 14089
-rw-r--r--    1 root     root        39696 Mar  9 16:17 14090
-rw-r--r--    1 root     root       114848 Mar  9 16:17 14091
-rw-r--r--    1 root     root          306 Mar  9 16:17 14092
-rw-r--r--    1 root     root        82628 Mar  9 16:17 14094
-rw-r--r--    1 root     root          541 Mar  9 16:17 14097
-rw-r--r--    1 root     root          998 Mar  9 16:17 14099
-rw-r--r--    1 root     root          523 Mar  9 16:17 14100
-rw-r--r--    1 root     root          512 Mar  9 16:17 14101
-rw-r--r--    1 root     root        31452 Mar  9 16:17 14103
-rw-r--r--    1 root     root        37984 Mar  9 16:17 14104
-rw-r--r--    1 root     root        14796 Mar  9 16:17 14105
-rw-r--r--    1 root     root          612 Mar  9 16:17 14106
-rw-r--r--    1 root     root       710020 Mar  9 16:17 14107
-rw-r--r--    1 root     root          471 Mar  9 16:17 14110
-rw-r--r--    1 root     root       312188 Mar  9 16:17 16066
-rw-r--r--    1 root     root         1556 Mar  9 16:17 30123
-rw-r--r--    1 root     root         2108 Mar  9 16:17 filetypes
```

Figure 2.20 – IRC Server log

```
:Stockholm.SE.eu.Undernet.org 001 Emperor^ :Welcome to the Internet
Relay Network, Emperor^
:Stockholm.SE.eu.Undernet.org 002 Emperor^ :Your host is
Stockholm.SE.eu.Undernet.org, running version u2.10.11.02
:Stockholm.SE.eu.Undernet.org 003 Emperor^ :This server was created Thu
Oct 10 2002 at 14:25:47 CEST
```

Figure 2.20 – IRC Server log

```
:Stockholm.SE.eu.Undernet.org 004 Emperor^ Stockholm.SE.eu.Undernet.org
u2.10.11.02 dioswkgx biklmnopstvr bklov
:Stockholm.SE.eu.Undernet.org 005 Emperor^ WHOX WALLCHOPS USERIP
CPRIVMSG CNOTICE SILENCE=15 MODES=6 MAXCHANNELS=15 MAXBANS=30 NICKLEN=9
TOPICLEN=160 AWAYLEN=160 KICKLEN=160 :are supported by this server
:Stockholm.SE.eu.Undernet.org 005 Emperor^ CHANTYPES=#& PREFIX=(ov)@+
CHANMODES=b,k,l,imnpstr CASEMAPPING=rfc1459 NETWORK=UnderNet :are
supported by this server
:Stockholm.SE.eu.Undernet.org 251 Emperor^ :There are 49311 users and
69897 invisible on 35 servers
:Stockholm.SE.eu.Undernet.org 252 Emperor^ 75 :operator(s) online
:Stockholm.SE.eu.Undernet.org 253 Emperor^ 134 :unknown connection(s)
:Stockholm.SE.eu.Undernet.org 254 Emperor^ 42499 :channels formed
:Stockholm.SE.eu.Undernet.org 255 Emperor^ :I have 10375 clients and 1
servers
:Stockholm.SE.eu.Undernet.org 375 Emperor^ :-
Stockholm.SE.eu.Undernet.org Message of the Day -
:Stockholm.SE.eu.Undernet.org 372 Emperor^ :Type /MOTD to read the AUP
before continuing using this service.
:Stockholm.SE.eu.Undernet.org 372 Emperor^ :The message of the day was
last changed: 2002-10-5 17:30
:Stockholm.SE.eu.Undernet.org 376 Emperor^ :End of /MOTD command.
```

Timeline analysis

A timeline analysis consists of a step-by-step account of everything that happened on the machine during the system compromise and the time the actions of the cracker took place. This timeline analysis has been created by pooling information from a number of different sources. These sources include the logs found in `/var/log`, and the file activity timeline produced by autopsy. The analysis is formatted in a way that makes it easy to read and understand. Each record consists of three fields. The first two fields are the event identifier. It includes the events date and a short description of the event. The time within the date field is written in 24-hour notation. The field immediately underneath the event identifier shows the evidence that the event occurred.

Timeline Analysis

Nov. 20 09:34:50 Operating System Installed

stat /tmp/install.log:

```
File: "install.log"
  Size: 5620            Blocks: 12         IO Block: 4096   Regular
File
Device: 705h/1797d      Inode: 12          Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/
root)
Access: Wed Nov 20 09:34:50 2002
Modify: Wed Nov 20 09:39:45 2002
Change: Wed Nov 20 09:39:45 2002
```

Timeline Analysis	
Nov. 24 11:29:41	Signs of the first ftp connection
/var/log/messages: `Nov 24 11:29:41 scoop ftpd[27074]: ANONYMOUS FTP LOGIN FROM 211.90.119.7 [211.90.119.7], mozilla@`	
Nov 24 16:37:05	Signs of the second ftp connection
/var/log/messages: `Nov 24 16:37:05 scoop ftpd[1197]: ANONYMOUS FTP LOGIN FROM w114.z208177157.sjc-ca.dsl.cnc.net [208.177.157.114], mozilla@`	
Nov 24 16:39:26	Added user with the username 'user'
/var/log/messages: `Nov 24 16:39:26 scoop adduser[1207]: new user: name=user, uid=0, gid=0, home=/root, shell=/bin/bash`	
Nov 24 16:39:59	Telnet session was initiated
/var/log/messages: `Nov 24 16:39:59 scoop in.telnetd[1210]: connect from 213.233.106.180`	
Nov 24 16:40:09	Authentication failure for user 'user'
/var/log/secure: `Nov 24 16:40:09  scoop login: FAILED LOGIN 1 FROM 106dial180.xnet.ro FOR user, Authentication failure`	
Nov 24 16:41:00	Password changed for user 'user'
/var/log/messages: `Nov 24 16:41:00 scoop PAM_pwdb[1212]: password for (user/0) changed by ((null)/0)`	
Nov 24 16:49:35	Added user with username 'jmk'
/var/log/messages: `Nov 24 16:49:35 scoop adduser[1432]: new user: name=jmk, uid=500, gid=500, home=/home/jmk, shell=/bin/bash`	
Nov 24 16:49:40	Password changed for user 'jmk'
/var/log/messages: `Nov 24 16:49:40 scoop PAM_pwdb[1433]: password for (jmk/500) changed by ((null)/0)`	
Nov 24 16:50:22	Password changed for user 'jmk'
/var/log/messages: `Nov 24 16:50:22 scoop PAM_pwdb[1436]: password for (jmk/500) changed by ((null)/0)`	
Nov 24 16:50:39	Telnet session was initiated
/var/log/secure: `Nov 24 16:50:39  scoop in.telnetd[1437]: connect from 213.233.106.180`	

Timeline Analysis	
Nov 24 16:50:52	Login by user jmk from 106dial180.xnet.ro

/var/log/messages:
`Nov 24 16:50:52 scoop PAM_pwdb[1438]: (login) session opened for user jmk by (uid=0)`
`Nov 24 16:50:52 scoop login[1438]: LOGIN ON ttyp0 BY jmk FROM 106dial180.xnet.ro`	

Nov 24 16:50:52	Login session closed for user 'jmk'
/var/log/messages: `Nov 24 16:50:52  scoop PAM_pwdb[1438]: (login) session closed for user jmk`	
Nov 24 16:56:42	File strike.tgz was downloaded in /home/ftp
File Activity Timeline: `781194 m.c -/-rw-r--r-- 0  50  2011  /home/ftp/strike.tgz`	
Nov 24 16:59:03	File strike.tgz was extracted.
File Activity Timeline: `781194 .a. -/-rw-r--r-- 0  50  2011  /home/ftp/strike.tgz` …	

Nov 24 16:59:07	The following files have been copied or overwritten:
	`/usr/bin/lsof`
	`/sbin/ifconfig`
	`/usr/bin/md5sum`
	`/bin/netstat`
	`/bin/ps`
	`/lib/libproc.so.2.0.6`

File Activity Timeline:
`Sun Nov 24 2002 16:59:07`
`82628 ..c -/-rwxr-xr-x 0  0  98143  /usr/sbin/lsof`
`31504 ..c -/-rwxr-xr-x 0  0  36206  /sbin/ifconfig`
`31452 ..c -/-rwxr-xr-x 0  0  14868  /usr/bin/md5sum`
`54152 .ac -/-rwxr-xr-x 0  0  18137  /bin/netstat`
`62920 ..c -/-rwxr-xr-x 0  0  18141  /bin/ps`
`37984 ..c -/-rwxr-xr-x 0  0  22152  /lib/libproc.so.2.0.6`	

Timeline Analysis	

Nov 24 16:59:08	The following files have been copied or overwritten:
	`/usr/X11R6/lib/X11/fonts/misc/ /sl2`
	`/usr/bin/vdir`
	`/usr/bin/pstree`
	`/usr/X11R6/lib/X11/fonts/misc/ /dan2`
	`/usr/X11R6/lib/X11/fonts/misc/ /scan`
	`/usr/lib/.shk2`
	`/usr/X11R6/lib/X11/fonts/misc/ /write`
	`/usr/X11R6/lib/X11/fonts/misc/ /statdx`
	`/usr/bin/dir`
	`/usr/bin/top`
	`/usr/lib/shk`
	`/usr/lib/.srs`
	`/usr/X11R6/lib/X11/fonts/misc/ /cl`
	`/usr/include/hosts.h`
	`/usr/X11R6/lib/X11/fonts/misc/ /wroot`
	`/usr/X11R6/lib/X11/fonts/misc/ /v`
	`/usr/X11R6/lib/X11/fonts/misc/ /dan1`
	`/usr/X11R6/lib/X11/fonts/misc/ /wscan`
	`/bin/login`
	`/usr/include/file.h`
	`/usr/include/proc.h`
	`/usr/lib/.sdc`
	`/usr/X11R6/lib/X11/fonts/misc/ /sc`
	`/usr/X11R6/lib/X11/fonts/misc/ /.x.tgz`
	`/usr/X11R6/lib/X11/fonts/misc/ /wted`
	`/usr/X11R6/lib/X11/fonts/misc/ /read`
	`/usr/X11R6/lib/X11/fonts/misc/ /tcp.log`
	`/usr/bin/killall`
	`/usr/bin/du`
	`/usr/lib/.x`
	`/bin/ls`
	
	The following files were executed:
	/ftp/raven/remove
	`/etc/rc.d/init.d/portmap`
	
	The portmap start scripts were removed from the `/etc/rc.d/rc?.d` directories.
	
	Extract and begin compilation of .x.tgz (adore)

Timeline Analysis

File Activity Timeline:

```
Sun Nov 24 2002 16:59:08
 16776 ..c -/-rwxr-xr-x 0        0        126502
/usr/X11R6/lib/X11/fonts/misc/   /sl2
155464 ..c -/-rwxr-xr-x 0        0        14390    /usr/bin/vdir
 12340 ..c -/-rwxr-xr-x 0        0        14999    /usr/bin/pstree
710020 m.c -/-rwxr-xr-x 0        50       126508
/usr/X11R6/lib/X11/fonts/misc/   /dan2
   982 ..c -/-rwxr-xr-x 0        0        126500
/usr/X11R6/lib/X11/fonts/misc/   /scan
   523 mac -/-rw------- 0        50       28812    /usr/lib/.shk2
  6324 ..c -/-rwxr-xr-x 0        0        126499
/usr/X11R6/lib/X11/fonts/misc/   /write
 11472 ..c -/-rwxr-xr-x 0        0        126498
/usr/X11R6/lib/X11/fonts/misc/   /statdx
 39696 ..c -/-rwxr-xr-x 0        0        14385    /usr/bin/dir
 33992 ..c -/-rwxr-xr-x 0        0        14993    /usr/bin/top
   541 ..c -/-rw------- 0        0        28811    /usr/lib/.shk
   512 mac -/-rw------- 0        50       28813    /usr/lib/.srs
  1345 ..c -/-rwxr-xr-x 0        0        126496
/usr/X11R6/lib/X11/fonts/misc/   /cl
   161 ..c -/-rw-r--r-- 0        0        24780    /usr/include/hosts.h
  1187 ..c -/-rwxr-xr-x 0        0        126503
/usr/X11R6/lib/X11/fonts/misc/   /wroot
  4684 ..c -/-rwxr-xr-x 0        0        126505
/usr/X11R6/lib/X11/fonts/misc/   /v
710020 m.c -/-rwxr-xr-x 0        50       126507
/usr/X11R6/lib/X11/fonts/misc/   /dan1
  6340 ..c -/-rwxr-xr-x 0        0        126504
/usr/X11R6/lib/X11/fonts/misc/   /wscan
  3744 ..c -/-rwxr-xr-x 0        0        18150    /bin/login
   250 ..c -/-rw-r--r-- 0        0        24781    /usr/include/file.h
   120 ..c -/-rw-r--r-- 0        0        24779    /usr/include/proc.h
   998 ..c -/-rw-r--r-- 0        0        28810    /usr/lib/.sdc
  6532 ..c -/-rwxr-xr-x 0        0        126501
/usr/X11R6/lib/X11/fonts/misc/   /sc
 14796 .ac -/-rw-r--r-- 0        0        126497
/usr/X11R6/lib/X11/fonts/misc/   /.x.tgz
  6324 ..c -/-rwxr-xr-x 0        0        126495
/usr/X11R6/lib/X11/fonts/misc/   /wted
  4060 ..c -/-rwxr-xr-x 0        0        126506
/usr/X11R6/lib/X11/fonts/misc/   /read
     0 mac -/-rw-r--r-- 0        50       126509
/usr/X11R6/lib/X11/fonts/misc/   /tcp.log
 21306 ..c -/-rwxr-xr-x 0        0        14998    /usr/bin/killall
114848 ..c -/-rwxr-xr-x 0        0        14387    /usr/bin/du
 15284 ..c -/-rws--x--x 0        0        28809    /usr/lib/.x
155462 ..c -/-rwxr-xr-x 0        0        18084    /bin/ls
  3238 .a. -/-rwxr-xr-x 0        0        14068
/home/ftp/raven/remove
   986 .a. -/-rwxr-xr-x 0        0        50220
/etc/rc.d/init.d/portmap
  1024 m.c -/drwxr-xr-x 0        0        54217    /etc/rc.d/rc1.d
  1024 m.c -/drwxr-xr-x 0        0        56225    /etc/rc.d/rc2.d
  1024 m.c -/drwxr-xr-x 0        0        58233    /etc/rc.d/rc3.d
  1024 m.c -/drwxr-xr-x 0        0        60241    /etc/rc.d/rc4.d
  1024 m.c -/drwxr-xr-x 0        0        2010     /etc/rc.d/rc5.d
  1024 m.c -/drwxr-xr-x 0        0        4018     /etc/rc.d/rc6.d
```

Timeline Analysis	

Nov 24 16:59:09	The following files have been copied or overwritten:
	`/etc/rc.d/init.d/init`
	
	The following file had been modified:
	`/etc/rc.d/rc.sysinit`
	
	The following files were modified:
	`/home/ftp/raven/startfile`
	`/home/ftp/raven/check`

File Activity Timeline:

```
Sun Nov 24 2002 16:59:09
 306 ..c -/-rwxr-xr-x 0        0        50228    /etc/rc.d/init.d/init
7206 m.c -/-rwxr-xr-x 0        0        48196    /etc/rc.d/rc.sysinit
1244 .a. -/-rwxr-xr-x 0        0        14080
/home/ftp/raven/startfile
 302 .a. -/-rwxr-xr-x 0        0        14098    /home/ftp/raven/check
```

Timeline Analysis	

Nov 24 16:59:11	E-mail message was sent to jmk@emoka.ro
	
	The following files were executed:
	`/home/ftp/raven/clean`
	`/home/ftp/raven/patch`
	`/home/ftp/raven/mailme`
	`/etc/rc.d/init.d/init`
	
	There was an attempt to start the following executables:
	`/usr/X11R6/lib/X11/fonts/misc/ /dan1 (sshd)`
	`/usr/X11R6/lib/X11/fonts/misc/ /dan2 (sshd)`
	`/usr/X11R6/lib/X11/fonts/misc/ /.x/start (adore)`

/var/log/maillog:

```
Nov 24 16:59:11  scoop sendmail[1813]: QAA01813: from=root, size=1371,
class=0, pri=31371, nrcpts=1,
msgid=<200211242159.QAA01813@scoop.secdog.com>, relay=root@localhost
```

File Activity Timeline:

```
Sun Nov 24 2002 16:59:11
   197 .a. -/-rwxr-xr-x 0        0        14081
/home/ftp/raven/clean
  1292 .a. -/-rwxr-xr-x 0        0        14077
/home/ftp/raven/patch
   329 .a. -/-rwxr-xr-x 0        0        14071
/home/ftp/raven/mailme
   306 .a. -/-rwxr-xr-x 0        0        50228
/etc/rc.d/init.d/init
710020 .a. -/-rwxr-xr-x 0        50       126507
/usr/X11R6/lib/X11/fonts/misc/   /dan1
710020 .a. -/-rwxr-xr-x 0        50       126508
/usr/X11R6/lib/X11/fonts/misc/   /dan2
   307 .a. -/-rwxr-xr-x 500      500      12391
/usr/X11R6/lib/X11/fonts/misc/   /.x/start
```

Nov 24 16:59:20	E-mail message accepted for delivery to jmk@emoka.ro by foreign host.

/var/log/messages:

```
Nov 24 16:59:20  scoop sendmail[1894]: QAA01813: to=jmk@emoka.ro,
ctladdr=root (0/0), delay=00:00:09, xdelay=00:00:09, mailer=esmtp,
relay=mx.emoka.ro. [217.156.96.21], stat=Sent (ok 1038179091 qp 23905)
```

Nov 24 17:00:46	/usr/bin/ftp was used to, most likely, download psyBNC.3.1

File Activity Timeline:

```
Sun Nov 24 2002 17:00:46
56548 .a. -/-rwxr-xr-x 0        0        14503    /usr/bin/ftp
```

Timeline Analysis	
Nov 24 17:02:28	Extraction of psyBNC.3.1 begins in /home/jmk/.. /httpd

File Activity Timeline:

```
Sun Nov 24 2002 17:02:28
739 .ac -/-rw-r--r-- 500      500      20146    /home/jmk/..
/httpd/help/ADDIGNORE.TXT
119 .a. -/-rw-r--r-- 500      500      20231    /home/jmk/..
/httpd/help/SETUSERNAME.DEU
149 .a. -/-rw-rw-r-- 500      500      20259    /home/jmk/..
/httpd/help/BQUIT.ITA
296 .a. -/-rw-rw-r-- 500      500      20274    /home/jmk/..
/httpd/help/DELENCRYPT.ITA
…
```

Nov 24 17:03:38	Compilation of psyBNC.3.1 begins

File Activity Timeline:

```
Sun Nov 24 2002 17:03:38
2129 .a. -/-rw-r--r-- 500      500      18077    /home/jmk/..
/httpd/Makefile
```

Nov 24 17:08:27	Start /home/jmk/.. /httpd/httpd (psyBNC.3.1)

File Activity Timeline:

```
Sun Nov 24 2002 17:08:27
194724 .a. -/-rwxrwxr-x 500      500      18090    /home/jmk/..
/httpd/httpd
```

Nov 24 18:46:24	Opening of telnet session

/var/log/secure:

```
Nov 24 18:46:24  scoop in.telnetd[4624]: connect from 213.233.106.180
```

Nov 24 18:46:26 System last accessed

File Activity Timeline:

```
Sun Nov 24 2002 18:46:26
   0 .a. -/crw-rw-rw- 0        0        12577    /dev/ptyp0
1356 .a. -/-rw-r--r-- 0        0        73514
/usr/share/terminfo/x/xterm
  61 .a. -/-rw-r--r-- 0        0        14129    /etc/issue.net
3744 .a. -/-rwxr-xr-x 0        0        18150    /bin/login
```

String Search

A string search was conducted using the keyword search feature in Autopsy. The keyword search feature allows an investigator to search for select strings within the media that is being analyzed. The benefit of using this within Autopsy is that it not only searches allocated files but also deleted inodes. There were two items left to be accomplished, finding the deleted e-mail to jmk@emoka.ro and finding an RPM believed to have been installed on the system. For these items the strings "jmk@emoka", "rpm", "wu-ftp" and "wuftp" were used. Nothing, other than what was already discovered, was uncovered using these strings. I attempted to find any other evidence on the system by using profane keywords. The cracker

community has a loose mouth when it comes to profanity and is often used in their source code. Nothing new was found with these string search attempts.

Conclusion

Today, getting the latest root exploit is like going to the candy store and picking up your favorite candy. It's not difficult, and if you choose the right one you can find yourself on a victim host in no time. It's difficult to know the extent of a hacker's knowledge since most of their tools are available to the public. But you can get an idea of their expertise, or lack there of, by the way they conduct their business on the victim host. An experienced hacker is going to take the time to get to know the system their working on and cover up their tracks. On the other hand, a script-kiddie, which is a hacker with minimal experience, is liable to make sloppy mistakes.

Based on the evidence collected from this particular system compromise it is possible to identify certain characteristics of the hacker responsible for breaking in. In the case of this compromise it is fair to say that this hacker was relatively inexperienced and lacked foresight during his utility installs and clean-up procedure. It was found that the compromised system had a relatively old distribution of Redhat Linux installed. Its version was 5.2 with kernel version 2.0.26. This could cause problems towards getting precompiled binaries to run on the machine.

There were a number of precompiled binaries transferred to the machine including dan1 and dan2 that were copies of the *sshd* daemon. During the execution of the 'strike' install script, an attempt was made to start those daemons, but no proof was gathered signifying they were ever started. This could be due to version incompatibility or the lack of necessary libraries. There was also no evidence showing that the *adore* LKM was ever compiled and installed. This could have been due to compilation errors during the attempt to compile the kernel module.

The hacker's clean-up procedures were also lacking. He had the tools to remove all the evidence relating to the compromise. However, they were used inefficiently. While using *sc*, which is a log cleaning tool, he used arguments that didn't pertain to the current hack event. In the `.clean` script, *sc* was used to remove logs with events pertaining to *sshd* even though *the attacker never utilized sshd*. He tried to remove logs that included the IP segments 208.158 and 209.235 even though he was connecting from 213.233.106.180 and 208.177.157.114.

Evidence showed that the last connection to the machine was made at Nov 24 18:46:24 via telnet. Yet, when checking connections to the machine more than 20 hours later, there was still an open shell session via port 21 (ftpd) identifying where the hacker was working from at that precise moment.

Leaving the strike.tgz file in the `/home/ftp` directly is a dead giveaway that something is amiss. There is no evidence signifying that there was any attempt at hiding or removing this file. Overall, I believe that hacker's attempts were quite sloppy.

References

1. Sharma, Kapil. "Who's Sniffing Your Network?." 2000. http://www.linux4biz.net/articles/articlesniff.htm

2. "Wu-Ftpd File Globbing Heap Corruption Vulnerability." Security Focus. November 27, 2001. http://www.securityfocus.com/bid/3581

3. "CA-2001-33: Multiple Vulnerabilities in WU-FTPD." Security Focus. November 29, 2001. http://online.securityfocus.com/advisories/3701

4. University of Toronto, Computing and Network Services. http://cns.utoronto.ca/~scan/expltool.txt

5. "TASK Tools." @Stake web site. http://www.atstake.com/research/tools/task

Part 3: Legal Issues and Incident Handling

The answers to the questions in this section refer to the situation where a system administrator of an Internet Service Provider (ISP) is contacted by a law enforcement agent who reveals that an account on one of the ISP's computers was used to hack into a government computer system. They discuss what the system administrator is permitted and not permitted to discuss with the agent. The answers to the questions also reveal what the ISP is allowed to investigate after discovering a crime has been committed by means of their facility. The laws referenced to answer the questions are federal computer crime laws of the United States.

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

Since an ISP is considered a business that provides services to the public, there are a number of restrictions placed upon that business. According to the Electronic Communications Privacy Act (ECPA), Disclosure of contents (18 USC 7202)

> "a person or entity providing remote computing services to the public shall not knowingly divulge to any person or entity the contents of any communication which is carried or maintained by that service to any governmental entity." [3]

It also states that

> "a provider of remote computing service or electronic communication service to the public shall not knowingly divulge a record or other information pertaining to a subscriber to or customer of such service to any governmental entity."[3]

These two articles give an ISP little choice regarding what they can or cannot divulge. Simply stated, they cannot divulge anything concerning the incident. However, there is a clause in the Computer fraud and abuse act (18 USC 1030) that prohibits anyone from "knowingly accessing a computer without authorization." [2] 18 USC 1030 (a)(3) states that anyone who "intentionally, without authorization to access any computer of a department or agency of the United States" shall be punished by fine or imprisonment, which means that such an act is considered a crime.[2]

Because of this fact, there is a clause in 18 USC 7202 that states that a provider of public service may divulge the contents of electronic communication "to a law enforcement agency if the contents appear to pertain to a crime."[3] This exception only applies to the contents of the communication. It does not allow for the ISP to

provide information concerning the customer records unless the divulgence is in self-defense.

In short, the given situation would allow an ISP to divulge only the contents of an electronic transmission.

If the subscriber of the service had signed an agreement stating that all communications was liable to be monitored and shared with law enforcement, the rights of the subscriber would be limited and all information pertaining to the transactions could be divulged to authorities. This is stated in both the Wiretap Act (18 USC 2511 (2) (c)) and in the Pen/Trap statutes (18 USC 3121 (b)). The same would go for an unregistered user of the service if the ISP had in some fashion bannered the connection to their services.

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

If the normal course of business was to delete or rotate the collected information and company policy mandated this act in a policy, law enforcement would need to provide a "good faith reliance on a court warrant or order, grand jury subpoena or legislative authorization."[5] This is stated in 18 USC 2707 (e)(1). If the ISP were to preserve the data without a good faith reliance, they could be liable for damages if its storage was not within the normal course of business (USC 2511 (2)(a).

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?

According to 18 USC 2703, Law enforcement must provide a warrant to receive any stored information pertaining to a crime that has been in storage for any length of time. The warrant must be "issued under the Federal Rules of Criminal Procedure or an equivalent state warrant." [4] Prior notice regarding the request for a warrant does not need to be disclosed to the ISP.

Data that has been stored for more than 180 days can also be obtained by law enforcement by issuing an administrative, grand jury or trial subpoena to the ISP. Prior notice must be given in the event of issuing a subpoena.

The records of an ISP customer may be disclosed to law enforcement if a warrant or court order has been obtained or if the subscriber has consented to

law enforcement that such information be disclosed. Once approval for the discloser of records is granted the ISP must provide the following information as described in Title 18 U.S.C 2703:

1. name
2. address
3. local and long distance telephone connection records, or records of session times and durations
4. length of service (including start date) and types of service utilized
5. telephone or instrument number or other subscriber number or identity, including any temporarily assigned network address
6. (F) means and source of payment for such service (including any credit card or bank account number), of a subscriber to or customer of such service when the governmental entity uses an administrative subpoena authorized by a Federal or State statute or a Federal or State grand jury or trial subpoena or any means available under paragraph (1).

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

According to the Wiretap Act (18 USC 2511) an ISP is permitted to gather information regarding communications facilitated by them as long as the interception of data is necessary during the normal course of business or to protect the rights of property of the ISP. Since the gathering of information regarding the compromise of a government computer does not directly affect the ISP, the ISP does not have the right to monitor or record the transmission beyond their normal monitoring efforts such as intrusion detection.

The Pen/Trap statute (18 USC 3121) has close to the same wording, stating the pen/trap devices used to gather connection information such as IP addresses and port numbers can only be done as necessary during the normal course of business. Many ISP's gather information regarding their customer's connections in an effort to protect their infrastructure from malicious hacker attacks but stop short when it comes to collecting the actual content of the transmission. The only time that complete collection of data could be deemed necessary would be during testing or trouble-shooting.

Given the situation, the only case where the ISP would have full rights to monitor the stream of data would be if the customer or user had, in some way, authorized it. This could be done upon the customer signing an agreement with the ISP or if the ISP had somehow bannered the initial connection between them and the user utilizing their service. This is stated in both the Wiretap Act and the Pen/Register statute.

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?

In the case that the logs uncover the use of the ISP's computers as a staging area for attack against a governmental computer system, the rights of the ISP broaden to allow the ISP to monitor additional traffic to protect their rights to property. As stated earlier, the ECPA, Discloser of Contents (USC 18 1702) limits the rights of a provider as to what they may divulge to law enforcement without the introduction of a search warrant. We have already uncovered that the ISP is permitted to share the contents of the stored data because the incident is regarded as a crime against the US Government. However, since the ISP is, in essence, directly responsible for an attack that occurred from one of their computer systems, the ISP has the right to divulge additional information regarding the incident to law enforcement. This includes the contents of stored information pertain to the crime as well as subscriber information regarding the crime, considering the ISP discovers who committed the act.

This type of incident also broadens the rights of the ISP to allow further investigation with regards to monitoring traffic on their network. The Wiretap Act and the Pen/Trap statutes both limit an ISP, only allowing them to monitor and record traffic or collect connection information as necessary during the normal course of business. However, in a situation where an ISP needs to protect their rights to property, they have complete access to all information pertaining to traffic regarding the illegal incident and may divulge that information to law enforcement in the act of self-defense.

The Pen/Trap Statute includes the section that states that

> "The prohibition of subsection (a) does not apply with respect to the use of a pen register or a trap and trace device by a provider of electronic or wire communication service relating to the operation, maintenance, and testing of a wire or electronic communication service or to the protection of the rights or property of such provider, or to the protection of users of that service from abuse of service or unlawful use of service."[6]

The Wiretap Act states that

> "It shall not be unlawful under this chapter for an operator of a switchboard, or on officer, employee, or agent of a provider of wire or electronic communication service, whose facilities are used in

> the transmission of a wire or electronic communication, to intercept, disclose, or use that communication in the normal course of his employment while engaged in any activity which is a necessary incident to the rendition of his service or to the protection of the rights or property of the provider of that service."[1]

These statements clarify that the ISP gains additional privileges when faced with a need to protect themselves against criminal penalties.

References:

1. "Interception and Disclosure of Wire, Oral, or Electronic Communications." 18 U.S.C. 2511. April 24, 2000. http://www.usdoj.gov/criminal/cybercrime/usc2511.htm

2. "Computer Fraud and Abuse Act 1986." 18 U.S.C. 1030. http://www.underground-book.com/chapters/ccm/123.html

3. "Disclosure of Contents." Title 18 U.S.C. 2702. December 19, 2001. http://www.usdoj.gov/criminal/cybercrime/usc2702.htm

4. "Requirements for Governmental Access." Title 18 U.S.C. 2703. December 17, 2001. http://www.usdoj.gov/criminal/cybercrime/usc2703.htm

5. "Civil Action." Title 18 U.S.C. 2707. http://www4.law.cornell.edu/cgi-bin/htm_hl?DB=uscode18&STEMMER=en&WORDS=2707+&COLOUR=Red&STYLE=s&URL=/uscode/18/2707.html#muscat_highlighter_first_match

6. "Pen Registers." Title 18 U.S.C. 3121. http://www.tscm.com/penreglaw.html

Last Updated: July 25th, 2017

Upcoming Training

Community SANS Columbia FOR508	Columbia, MD	Aug 21, 2017 - Aug 26, 2017	Community SANS
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
SANS Dublin 2017	Dublin, Ireland	Sep 11, 2017 - Sep 16, 2017	Live Event
Data Breach Summit & Training	Chicago, IL	Sep 25, 2017 - Oct 02, 2017	Live Event
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
SANS October Singapore 2017	Singapore, Singapore	Oct 09, 2017 - Oct 28, 2017	Live Event
SANS Tysons Corner Fall 2017	McLean, VA	Oct 14, 2017 - Oct 21, 2017	Live Event
SANS vLive - FOR508: Advanced Digital Forensics, Incident Response, and Threat Hunting	FOR508 - 201710,	Oct 16, 2017 - Nov 22, 2017	vLive
SANS Tokyo Autumn 2017	Tokyo, Japan	Oct 16, 2017 - Oct 28, 2017	Live Event
Mentor Session - FOR508	Brasilia, Brazil	Oct 18, 2017 - Oct 21, 2017	Mentor
SANS Berlin 2017	Berlin, Germany	Oct 23, 2017 - Oct 28, 2017	Live Event
SANS Seattle 2017	Seattle, WA	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Gulf Region 2017	Dubai, United Arab Emirates	Nov 04, 2017 - Nov 16, 2017	Live Event
SANS Milan November 2017	Milan, Italy	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Sydney 2017	Sydney, Australia	Nov 13, 2017 - Nov 25, 2017	Live Event
SANS Paris November 2017	Paris, France	Nov 13, 2017 - Nov 18, 2017	Live Event
SANS London November 2017	London, United Kingdom	Nov 27, 2017 - Dec 02, 2017	Live Event
SANS Austin Winter 2017	Austin, TX	Dec 04, 2017 - Dec 09, 2017	Live Event
SANS Cyber Defense Initiative 2017	Washington, DC	Dec 12, 2017 - Dec 19, 2017	Live Event
SANS Security East 2018	New Orleans, LA	Jan 08, 2018 - Jan 13, 2018	Live Event
SANS Las Vegas 2018	Las Vegas, NV	Jan 28, 2018 - Feb 02, 2018	Live Event
Cyber Threat Intelligence Summit & Training 2018	Bethesda, MD	Jan 29, 2018 - Feb 05, 2018	Live Event
SANS Dallas 2018	Dallas, TX	Feb 19, 2018 - Feb 24, 2018	Live Event
SANS New York City Winter 2018	New York, NY	Feb 26, 2018 - Mar 03, 2018	Live Event
SANS San Francisco Spring 2018	San Francisco, CA	Mar 12, 2018 - Mar 17, 2018	Live Event
SANS OnDemand	Online	Anytime	Self Paced
SANS SelfStudy	Books & MP3s Only	Anytime	Self Paced