Chapter 4

Initial Triage and Live Response: Data Analysis

Solutions in this chapter:

Introduction

OK… so now you have gathered all of the volatile information from the target system(s), and powered them down. Now what? How do you go from a bunch of seemingly unrelated data, to meaningful information that will help to bring you closer to figuring out what has occurred? The information that needs to be gleaned from the volatile data will obviously change from case to case, but the means by which you parse out this information should remain the same. You should look for things in roughly the same way each time, allowing the data in the case to determine which trails you follow along the way.

Think of a tree. All trees are roughly the same in that they have roots, a trunk, and branches. Some trees may have large, thick branches like an Oak, some may have small brittle branches like a Pine, while others may have long drooping branches like a Willow. The point is, no matter how much they differ, they are all trees. Now, take this logic and apply is to forensic analysis. All of your cases will more or less be the same in that you have some computer systems, a network, an incident, and a bad guy. The specifics of the case will change each time, but the core of the incident will, for the most part, remain the same.

It is important to note that everybody is different, and will have a different way of doing things, and that's OK. Personally, I like to begin by looking at log files. Having been a UNIX administrator for several years before moving into the security field, I have a good feel for the way things are supposed to look. I have spoken to other investigators who like to begin with the users, who has accounts, who logged in last, that sort of thing. Still others like to start with the network connections, what was being made to the box, what was coming from the box, and so forth. All of this information can be important to the investigation, however, it's subjective with regards to what order the information is analyzed in. The remainder of the chapter contains my personal outline for volatile analysis, and is meant to be a guide. Feel free to modify it to fit your own personal style and level of comfort.

Initial Triage

Before delving into the forensics of an actual host, you need to establish the baseline parameters of the incident. What does the customer "think" happened, do they have a rough timeline, which systems are involved, and so forth. At this stage of the investigation, it is important to simply ask questions, write down information, and try and clarify what the parameters of the incident are. From my experience, due to stress created by the incident and the pressure placed on the individual(s) from their superiors, the information provided to you during Initial Triage is sketchy at best. So it's important to understand that you should be the calm voice of reason from the minute you walk through the door. Remember, the customer is relying on you to help them, so be cool.

After the customer has told you their version of what has happened, it is important to ask probing questions. You need to be able to fill in the gaps between what the customer thinks happened and what really did happen, which in many cases is easier said than done. The following are items that you will need to make sure you understand completely so that you can conduct an effective and efficient response.

- **Timeline** If at all possible, you need to try and put the incident within a specific window. This may or may not be possible based on the nature of the incident, but it should be attempted inasmuch as the situation allows. Some cases will enable you to narrow your focus down to a specific day, or maybe even to specific hours within a day, while others may encompass several years. Whatever the case may be, make sure you are as thorough as possible. Failure to do so can seriously affect the rest of the investigation.
- **Network Topography** Get the lay of the land. I have not been in a situation yet where the customer does not at least have a high-level network diagram of their infrastructure, so make sure you ask them for it.
- **Data Flow** Once you have the network diagram, make sure you understand the data flow. Where are the ingress and egress points? What other systems are on the same subnet? If in a Windows domain, are there domain cross trusts which allow access to other domains? You need to understand not just what systems are involved in the incident, but also what system could be involved in the incident. Many customers are only focused on what their Information Technology (IT) staff has determined to be the scope of the incident, and don't see the bigger picture. Your job is to widen the area of focus to include all potentially involved systems. You can determine if they were involved later, during log analysis.
- **Security Appliances** What does the customer have on their network, and does it log. Best security practices are easy to recite, and preach, but implementation is an entirely different story. Many customers know they should

log, but don't. They have wanted to put in an Intrusion Detection System (IDS), or Intrusion Prevention System (IPS), but have not had the resources. You will need to find out what they have, where it sits on the network, and does it log. Make sure you request any logs they do have.

- **Status of Effected Systems** This is another one of those items in which the customer may not really know much about. I have been involved in several cases in which you are told one thing, for instance that a particular system as not been rebooted; only to find out the exact opposite is true once you arrive on-site. So even if you asked the questions prior to your arrival at the customer location, you have to ask them again, and verify if possible. This information can affect the direction of your investigation.
- **Business as Usual** As much as possible, you need to understand what "normal" is to the customer. When responding to an incident, you are most likely looking at the customer infrastructure for the first time. You will have no idea what their user ID naming convention is, what kind of traffic they experience on an average day, which systems normally communicate with each other, or any one of literally hundreds of potential variables that make up their typical day. For you to perform any sort of initial analysis you need to understand this as possible.

While you need to ask as many questions regarding this subject as possible, understand that more questions will undoubtedly arise as the investigation progresses. Make sure you let the customer know that you will have more questions, and you will need a solid point of contact (POC) that is knowledgeable enough about their technical and business processes to intelligibly answer them.

Once you have gathered all of the information you deem to be relevant (which will most assuredly change), you can begin initial analysis. The most important thing at this stage in the investigation is to remain neutral about the incident. Try not to formulate any premature conclusions about the direction the data is taking you. Simply let the data dictate the path you take.

Log Analysis

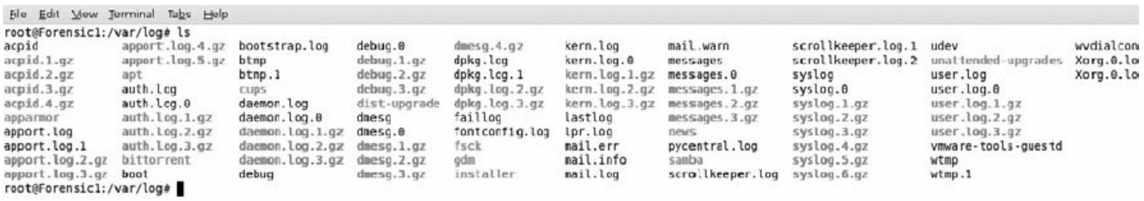
In my humble opinion, the starting point for your investigation should be an analysis of whatever log files the customer was able to provide. This may be nothing, in which case you might as well take some aspirin now, because a headache is rapid approaching, or it may be several terabytes of data, in which case you should probably take the aspirin anyway, as that same headache is rapidly approaching.

Start at the beginning. This is the simple notion that in any incident, the intruder has to get onto the customer network from somewhere, so start there. It may be a Virtual Private Network (VPN) concentrator, it may be a satellite office, and it may be from a specific workstation or server. Whatever the case may be, start there.

Log files on Linux systems can be wonderful things. They are highly configurable, efficient, and detailed. With any luck, the system you are gathering and/or analyzing logs from has at the very least, the default configurations for logging in place. Linux logs are in plain text, so you will not need to use any third-party software or utilities to perform effective searches. Additionally, you can write custom scripts to perform automatic actions based on the content of the logs, and the desired output.

Linux logs are located in the */var/log* directory. These are the log files both maintained by the system, and more than likely (and usually by default), from any third-party software that has been installed onto the system. You will also see some files in the directory which end in a number, as can be seen in Figure 4.1.

Figure 4.1 Files Ending in Numbers



```
root@Forensic1:/var/log# ls
acpid            apport.log.4.gz  bootstrap.log    debug.0       dmesg.4.gz      kern.log       mail.warn         scrollkeeper.log.1  udev                 wvdialcon
acpid.1.gz       apport.log.5.gz  btmp             debug.1.gz    dpkg.log        kern.log.0     messages          scrollkeeper.log.2  unattended-upgrades  Xorg.0.lo
acpid.2.gz       apt              btmp.1           debug.2.gz    dpkg.log.1      kern.log.1.gz  messages.0        syslog              user.log             Xorg.0.lo
acpid.3.gz       auth.log         cups             debug.3.gz    dpkg.log.2.gz   kern.log.2.gz  messages.1.gz     syslog.0            user.log.0
acpid.4.gz       auth.log.0       daemon.log       dist-upgrade  dpkg.log.3.gz   kern.log.3.gz  messages.2.gz     syslog.1.gz         user.log.1.gz
apparmor         auth.log.1.gz    daemon.log.0     dmesg         faillog         lastlog        messages.3.gz     syslog.2.gz         user.log.2.gz
apport.log       auth.log.2.gz    daemon.log.1.gz  dmesg.0       fontconfig.log  lpr.log        news              syslog.3.gz         user.log.3.gz
apport.log.1     auth.log.3.gz    daemon.log.2.gz  dmesg.1.gz    fsck            mail.err       pycentral.log     syslog.4.gz         vmware-tools-guestd
apport.log.2.gz  bittorrent       daemon.log.3.gz  dmesg.2.gz    gdm             mail.info      samba             syslog.5.gz         wtmp
apport.log.3.gz  boot             debug            dmesg.3.gz    installer       mail.log       scrollkeeper.log  syslog.6.gz         wtmp.1
root@Forensic1:/var/log# ▌
```

These are called *rotated archives*. Depending on what level of granularity the logging has been set to, logs can get large and cumbersome. Linux provides a command called "logrotate," which does exactly what you would think it does: it rotates the log files by appending a number to the end of the file. For example, in Figure 4.1 you can see "syslog" without a numerical suffix. That is the current log. The first log in the archive is the "syslog.0" file, and is the previously active log file. Next you will see "syslog1.gz" through "syslog.6.gz". These are the remainder of the archived log files in a gzipped format. When "logrotate" runs, usually daily (can be found by default in */etc/cron.daily*), it takes the current file, appends a ".0" to the end of the filename, and starts logging again in the standard log file, in this example, "syslog."

The other log files are then increased in numerical value by 1, with the oldest log being deleted. All of this information is configurable in the */etc/logrotate.conf* file. The system administrator at the customer location should be able to tell you what the log gathering parameters are for his servers.

To begin searching through the logfiles, you can use the native Linux commands, or any text tool. Below are some useful commands:

zgrep

Zgrep invokes grep on compressed or gzipped files.

```
zgrep <search_parameter> *
```

This will search all compressed files in the current working directory for the <search_parameter>.

Tail

Tail will display the last output of the file as denoted by the next argument. For example, "tail -100 <filename>" will display the last 100 lines of the file. Additionally, using the –f switch will display a log file as it gathers information in real time.

```
tail –f /var/log/messages
```

This will display the contents of /var/log/messages as new output occurs.

More

More works the same as the MS DOS version of the command, by simply sending the contents of the specified file to stdout.

```
more <filename>
```

This will display the contents of the <filename>, stopping output at the bottom of the screen with the word "—More—(x%)." This indicates that you are currently looking (or have looked at) x% of the total file. You can press the enter key to scroll down by one line, or the space bar to scroll down by one page. To scroll backwards by one page, simply press the letter "b." Additionally, you can search thorough the file with the "/" key followed by the <search_parameter>. If multiple occurrences of the <search_parameter> are found, simply press the "n" key to skip to the next entry, while the letter "p" will take you to the previous entry. The letter "q" allows you to quit the current view and returns you to the command prompt.

Less

Less is the opposite of more. It allows you to perform the same functions as the "more" command, but with much more control, like adding the ability to move both backwards and forwards in the file. It also loads much faster than more, since it does not read the entire file before opening it up.

```
less <filename>
```

This will display the contents of the <filename>. You can scroll backward with the "b" key, and forward with the "d" key. Just like "more," you can press the "/" key followed by a <search_parameter> to conduct a string search; "n" will take you to the next occurrence of the hit, "p" will take you to the previous one.

Keyword Searches

Keyword searches are a quick and easy way to help you identify points of interest on the targeted machines. These can either be performed on the live system, provided that the volatile information has already been gathered and the forensic imaging has already taken place, or post mortem in a laboratory environment. The important thing to remember is that keyword searches operate under the assumption that the bad guys have left the original names for whatever tools they have used, in place on the system. You obviously are not going to be able to guess if the names of any files that have been changed, so just keep it in the back of your mind during the data analysis and let the data guide you.

To perform our keyword searches, we are going to stick with the resident utilities found on the Linux operating system:

- strings
- grep
- less

For our example, I have gathered the contents of */proc/kore* from my Ubuntu 7.10 (Gutsy) machine.

NOTE

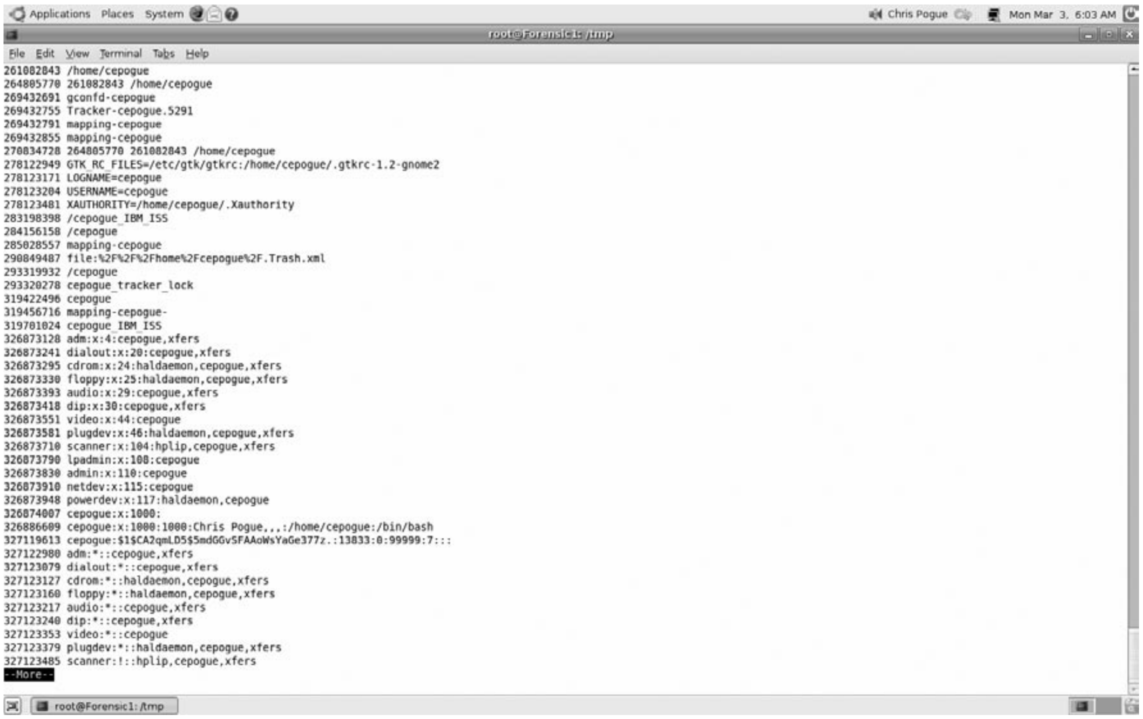
The kcore is an extremely useful file to capture and analyze from a compromised Linux machine. Like the rest of the information found in */proc*, kcore are virtual files created by the kernel to provide the user with valuable information about the running system, and is the exact size as available memory. Think of the kcore as the physical, tangible files (sort of) that directly correlate to what the system is doing in memory (but remember, they are not "real," they are virtual). If you try to "cat" kcore, the system will display a bunch of seemingly useless garble with some recognizable characters thrown in every so often. For the purposes of forensic analysis, make sure to use the "strings" command, which will only display printable characters.

strings /proc/kcore –t d > /tmp/kcore_outfile

In this particular command, I have chosen to use the "–t" and "d" switches. The "–t" option will print the offset at the front of each line, while the "d" option will put those offset numbers in decimal format (called the radix). You can get the full listing of options available using this (and any other command for that matter) from the man page.

Now that I have my strings output from kcore, I can perform my keyword searches to see if any nefarious processes are running on the system. In Figure 4.2, I grep'ed for my username, "cepogue" and piped the output through more. The result is every occurrence of my username that is currently loaded into memory. Since this is my machine, obviously there are going to be quite a large number of hits from my search. Hopefully, on the system that is being investigated, this will not be the case.

Figure 4.2 Greg'ed Username


Applications Places System
Chris Pogue Mon Mar 3, 6:03 AM
root@Forensic1: /tmp
File Edit View Terminal Tabs Help
root@Forensic1: /tmp

```
261082843 /home/cepogue
264805770 261082843 /home/cepogue
269432691 gconfd-cepogue
269432755 Tracker-cepogue.5291
269432791 mapping-cepogue
269432855 mapping-cepogue
270834728 264805770 261082843 /home/cepogue
278122949 GTK_RC_FILES=/etc/gtk/gtkrc:/home/cepogue/.gtkrc-1.2-gnome2
278123171 LOGNAME=cepogue
278123204 USERNAME=cepogue
278123481 XAUTHORITY=/home/cepogue/.Xauthority
283198398 /cepogue_IBM_ISS
284156158 /cepogue
285028557 mapping-cepogue
290849487 file:%2F%2F%2Fhome%2Fcepogue%2F.Trash.xml
293319932 /cepogue
293320278 cepogue_tracker_lock
319422496 cepogue
319456716 mapping-cepogue-
319701024 cepogue_IBM_ISS
326873128 adm:x:4:cepogue,xfers
326873241 dialout:x:20:cepogue,xfers
326873295 cdrom:x:24:haldaemon,cepogue,xfers
326873330 floppy:x:25:haldaemon,cepogue,xfers
326873393 audio:x:29:cepogue,xfers
326873418 dip:x:30:cepogue,xfers
326873551 video:x:44:cepogue
326873581 plugdev:x:46:haldaemon,cepogue,xfers
326873710 scanner:x:104:hplip,cepogue,xfers
326873790 lpadmin:x:108:cepogue
326873830 admin:x:110:cepogue
326873910 netdev:x:115:cepogue
326873948 powerdev:x:117:haldaemon,cepogue
326874007 cepogue:x:1000:
326886609 cepogue:x:1000:1000:Chris Pogue,,,:/home/cepogue:/bin/bash
327119613 cepogue:$1$CA2qmLD5$5mdGGvSFAAoWsYaGe377z.:13833:0:99999:7:::
327122980 adm:*::cepogue,xfers
327123079 dialout:*::cepogue,xfers
327123127 cdrom:*::haldaemon,cepogue,xfers
327123160 floppy:*::haldaemon,cepogue,xfers
327123217 audio:*::cepogue,xfers
327123240 dip:*::cepogue,xfers
327123353 video:*::cepogue
327123379 plugdev:*::haldaemon,cepogue,xfers
327123485 scanner:!::hplip,cepogue,xfers
--More--
```

In addition to keywords provided to you by the customer, it is a good idea to keep your own personal keyword list, which is updated at the conclusion of each case. In my experience, I learn something new with each case, so keeping a dynamic keyword list helps me to not only remember what I have found in the past (and need to research further), but it helps me to find it again in future cases. Here are some of the keywords that I search for on a regular basis.

File and Directory Names

- grep –e (the "–e" is used here for pattern matching) "\/proc/" –e "\/bin" –e "\/bin\/.*?sh" kcore_strings.
- grep –e "ftp" –e "root" kcore_strings
- grep –e "rm –r" kcore_strings
- grep –e ".tgz" kcore_strings

IP Addresses and Domain Names

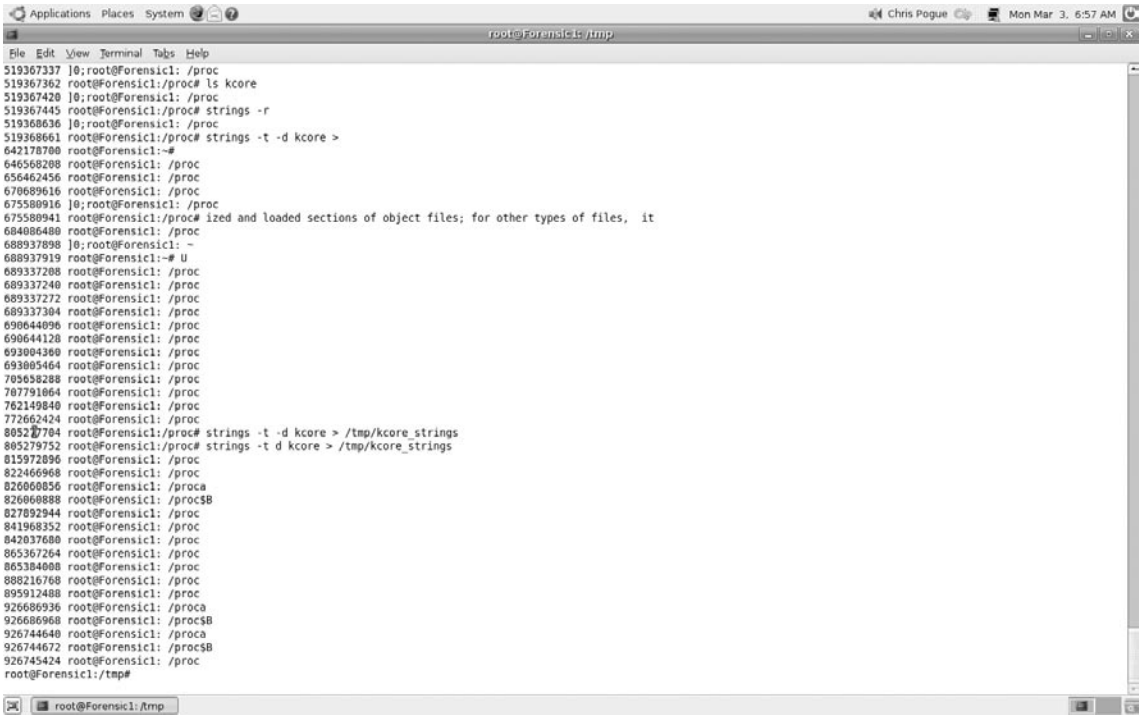
- grep –e "[0-9]\+\.[0-9]\+\.[0-9]\+\.[0-9]\+" kcore_strings
- grep –e "\.pl\" kcore strings

Tool Keywords

- msf (Metasploit Framework)
- select
- insert
- dump
- update
- nmap
- nessus
- nikto
- wireshark
- tcpdump
- kismet
- airsnarf
- paros
- hping2
- ettercap
- aircrack
- aircrack-ng
- airsnort
- nc (netcat)

Now, let's suppose you find something of interest, and you want to probe into it a bit deeper. In Figure 4.3, you will see the results of me searching kcore_strings for the keyword, "root@Forensic1" (my localhost root account).

Figure 4.3 Search Results



```
519367337 ]0;root@Forensic1: /proc
519367362 root@Forensic1:/proc# ls kcore
519367420 ]0;root@Forensic1: /proc
519367445 root@Forensic1:/proc# strings -r
519368636 ]0;root@Forensic1: /proc
519368661 root@Forensic1:/proc# strings -t -d kcore >
642178700 root@Forensic1:~#
646568208 root@Forensic1: /proc
656462456 root@Forensic1: /proc
670689616 root@Forensic1: /proc
675580916 ]0;root@Forensic1: /proc
675580941 root@Forensic1:/proc# ized and loaded sections of object files; for other types of files,  it
684086480 root@Forensic1: /proc
688937898 ]0;root@Forensic1: ~
688937919 root@Forensic1:~# U
689337208 root@Forensic1: /proc
689337240 root@Forensic1: /proc
689337272 root@Forensic1: /proc
689337304 root@Forensic1: /proc
690644096 root@Forensic1: /proc
690644128 root@Forensic1: /proc
693004360 root@Forensic1: /proc
693005464 root@Forensic1: /proc
705658288 root@Forensic1: /proc
707791064 root@Forensic1: /proc
762149840 root@Forensic1: /proc
772662424 root@Forensic1: /proc
805277704 root@Forensic1:/proc# strings -t -d kcore > /tmp/kcore_strings
805279752 root@Forensic1:/proc# strings -t d kcore > /tmp/kcore_strings
815972896 root@Forensic1: /proc
822466968 root@Forensic1: /proc
826060856 root@Forensic1: /proca
826060888 root@Forensic1: /proc$B
827892944 root@Forensic1: /proc
841968352 root@Forensic1: /proc
842037680 root@Forensic1: /proc
865367264 root@Forensic1: /proc
865384008 root@Forensic1: /proc
888216768 root@Forensic1: /proc
895912488 root@Forensic1: /proc
926686936 root@Forensic1: /proca
926686968 root@Forensic1: /proc$B
926744640 root@Forensic1: /proca
926744672 root@Forensic1: /proc$B
926745424 root@Forensic1: /proc
root@Forensic1:/tmp#
```

As you can see, the search returned anything from my strings output from running memory that matched my search criteria, "root@Forensic1." Now let's say that you think that offset 805277704 looks interesting. The next step would be to open kcore_strings with a text editor. In this example, I use the "less" command.

```
less kcore_strings
```

This will open up the kcore_strings file in a searchable format

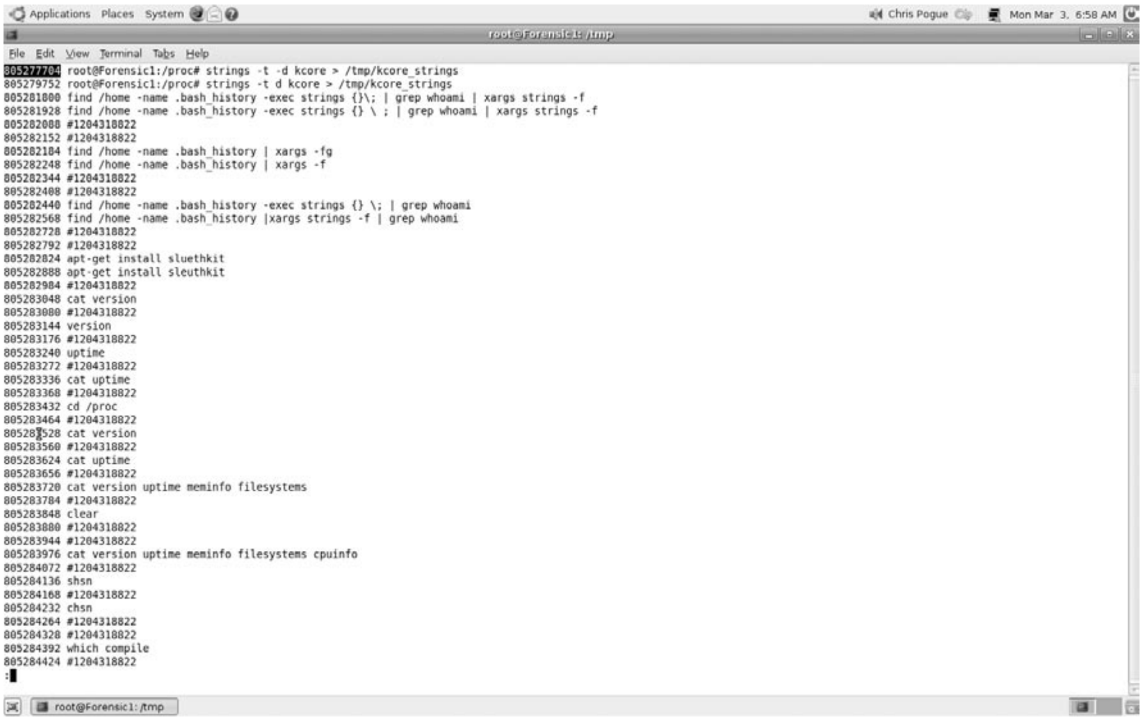
```
/<search_argument>
```

This will search through the file for an occurrence of the <search_argument>.

In my example, I used the following search: */805277704* (This is quite a few lines up in the kore_strings file, so this may take a few seconds to return with your results.

With any luck, we might be able to find some of the commands typed in during the time that this offset was recorded into memory. In Figure 4.4, you will see that I fat-fingered my search by putting in a "–" on my second argument. You will also see that I was playing with some scripting using "xargs," as well as installing The Sleuth Kit.

Figure 4.4 Fat Fingered Search



```
805277704 root@Forensic1:/proc# strings -t -d kcore > /tmp/kcore_strings
805279752 root@Forensic1:/proc# strings -t d kcore > /tmp/kcore_strings
805281800 find /home -name .bash_history -exec strings {}\; | grep whoami | xargs strings -f
805281928 find /home -name .bash_history -exec strings {} \ ; | grep whoami | xargs strings -f
805282088 #1204318822
805282152 #1204318822
805282184 find /home -name .bash_history | xargs -fg
805282248 find /home -name .bash_history | xargs -f
805282344 #1204318822
805282408 #1204318822
805282440 find /home -name .bash_history -exec strings {} \; | grep whoami
805282568 find /home -name .bash_history |xargs strings -f | grep whoami
805282728 #1204318822
805282792 #1204318822
805282824 apt-get install sluethkit
805282888 apt-get install sleuthkit
805282984 #1204318822
805283048 cat version
805283080 #1204318822
805283144 version
805283176 #1204318822
805283240 uptime
805283272 #1204318822
805283336 cat uptime
805283368 #1204318822
805283432 cd /proc
805283464 #1204318822
805283528 cat version
805283560 #1204318822
805283624 cat uptime
805283656 #1204318822
805283720 cat version uptime meminfo filesystems
805283784 #1204318822
805283848 clear
805283880 #1204318822
805283944 #1204318822
805283976 cat version uptime meminfo filesystems cpuinfo
805284072 #1204318822
805284136 shsn
805284168 #1204318822
805284232 chsn
805284264 #1204318822
805284328 #1204318822
805284392 which compile
805284424 #1204318822
:
```

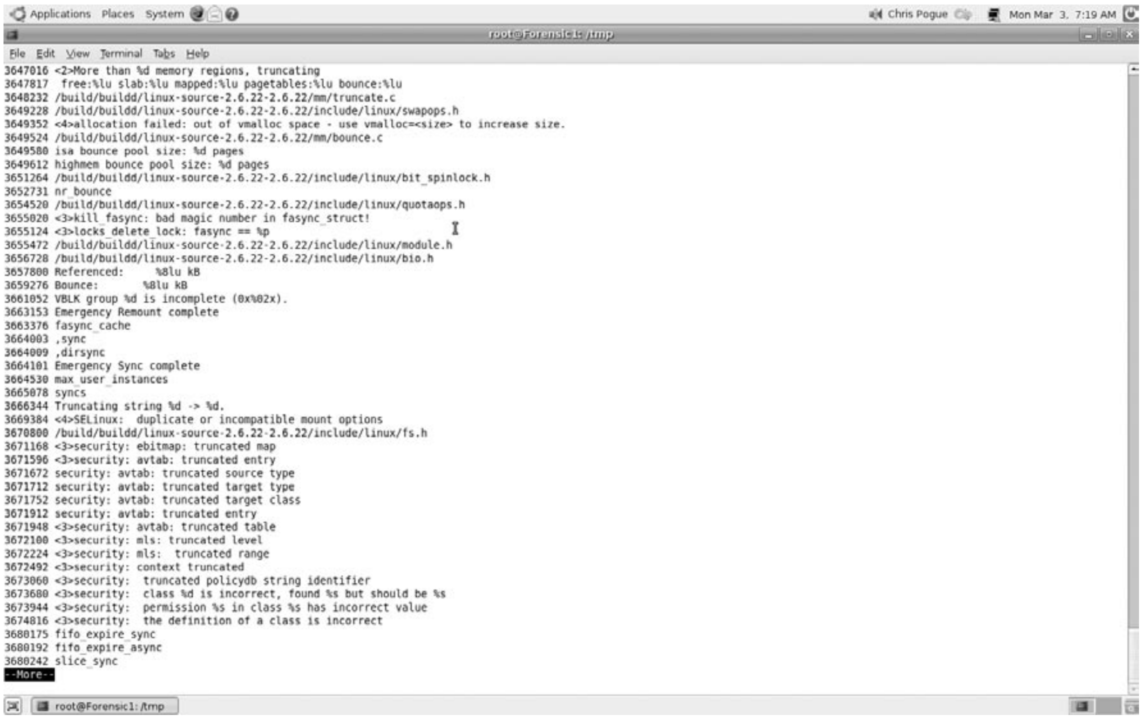
So, as you can see, this is a simple yet powerful way to perform string searches on a live system or booted image. Remember, pages within the virtual memory, physical memory, and swap are overwritten in an unorganized manner. This means that your attempts may hit a dead end here, or they may be invalid. Use the information gleaned from this process in conjunction with other information acquired during the course of the investigation.

Tricks of the Trade

The operating system is not going to ask you to "be more specific" in your search requests. Put simply, you are going to get what you tell it to find. So, you have to know how to properly stack your search arguments to make your keyword searches as efficient and effective as possible.

In this example, I am again, going to use my Ubuntu 7.10 (Gutsy) machine. In Figure 4.5, you can see that I performed a keyword search against kcore_strings for the term, "nc".

Figure 4.5 Keyword Search



```
3647016 <2>More than %d memory regions, truncating
3647817  free:%lu slab:%lu mapped:%lu pagetables:%lu bounce:%lu
3648232 /build/buildd/linux-source-2.6.22-2.6.22/mm/truncate.c
3649228 /build/buildd/linux-source-2.6.22-2.6.22/include/linux/swapops.h
3649352 <4>allocation failed: out of vmalloc space - use vmalloc=<size> to increase size.
3649524 /build/buildd/linux-source-2.6.22-2.6.22/mm/bounce.c
3649580 isa bounce pool size: %d pages
3649612 highmem bounce pool size: %d pages
3651264 /build/buildd/linux-source-2.6.22-2.6.22/include/linux/bit_spinlock.h
3652731 nr_bounce
3654520 /build/buildd/linux-source-2.6.22-2.6.22/include/linux/quotaops.h
3655020 <3>kill_fasync: bad magic number in fasync_struct!
3655124 <3>locks_delete_lock: fasync == %p
3655472 /build/buildd/linux-source-2.6.22-2.6.22/include/linux/module.h
3656728 /build/buildd/linux-source-2.6.22-2.6.22/include/linux/bio.h
3657800 Referenced:     %8lu kB
3659276 Bounce:         %8lu kB
3661052 VBLK group %d is incomplete (0x%02x).
3663153 Emergency Remount complete
3663376 fasync_cache
3664003 ,sync
3664009 ,dirsync
3664101 Emergency Sync complete
3664530 max_user_instances
3665078 syncs
3666344 Truncating string %d -> %d.
3669384 <4>SELinux:  duplicate or incompatible mount options
3670800 /build/buildd/linux-source-2.6.22-2.6.22/include/linux/fs.h
3671168 <3>security: ebitmap: truncated map
3671596 <3>security: avtab: truncated entry
3671672 security: avtab: truncated source type
3671712 security: avtab: truncated target type
3671752 security: avtab: truncated target class
3671912 security: avtab: truncated entry
3671948 <3>security: avtab: truncated table
3672100 <3>security: mls: truncated level
3672224 <3>security: mls:  truncated range
3672492 <3>security: context truncated
3673060 <3>security:  truncated policydb string identifier
3673680 <3>security:  class %d is incorrect, found %s but should be %s
3673944 <3>security:  permission %s in class %s has incorrect value
3674816 <3>security:  the definition of a class is incorrect
3680175 fifo_expire_sync
3680192 fifo_expire_async
3680242 slice_sync
--More--
```

As you can see, the search for the term "nc" yielded every entry in which the letters "n" and "c" appeared together. This is obviously not a very clean search, so I will have to refine my parameters to give me something a bit more useable.

Using what I know about netcat, it can either be used to send or receive information. The command structure to send a file is:

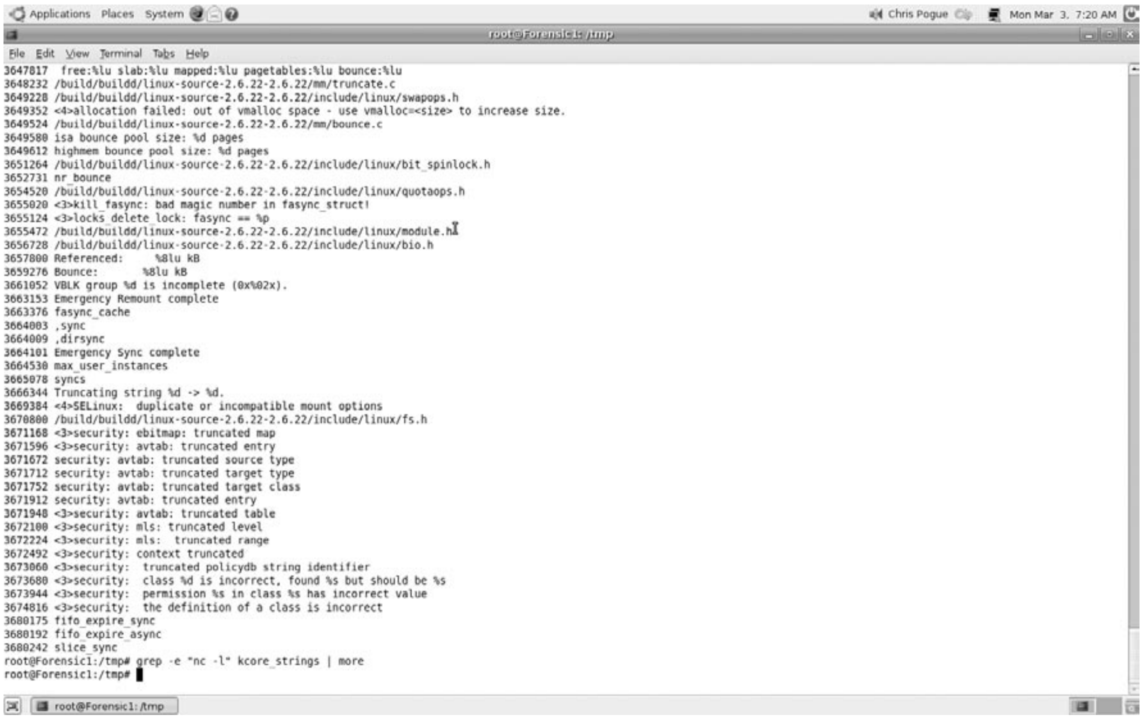
```
nc <host.example.com> (or IP address) <port> < infile
```

The command structure to receive a file is:

```
nc –l (for *l*isten) <port> > outfile
```

Based on this information, I can refine my search to see if the host I am investigating has been used to either send or receive a file using netcat. In Figure 4.6, you can see that my search argument did not yield any hits, so I can safely assume that the host did not receive any files using "nc –l."

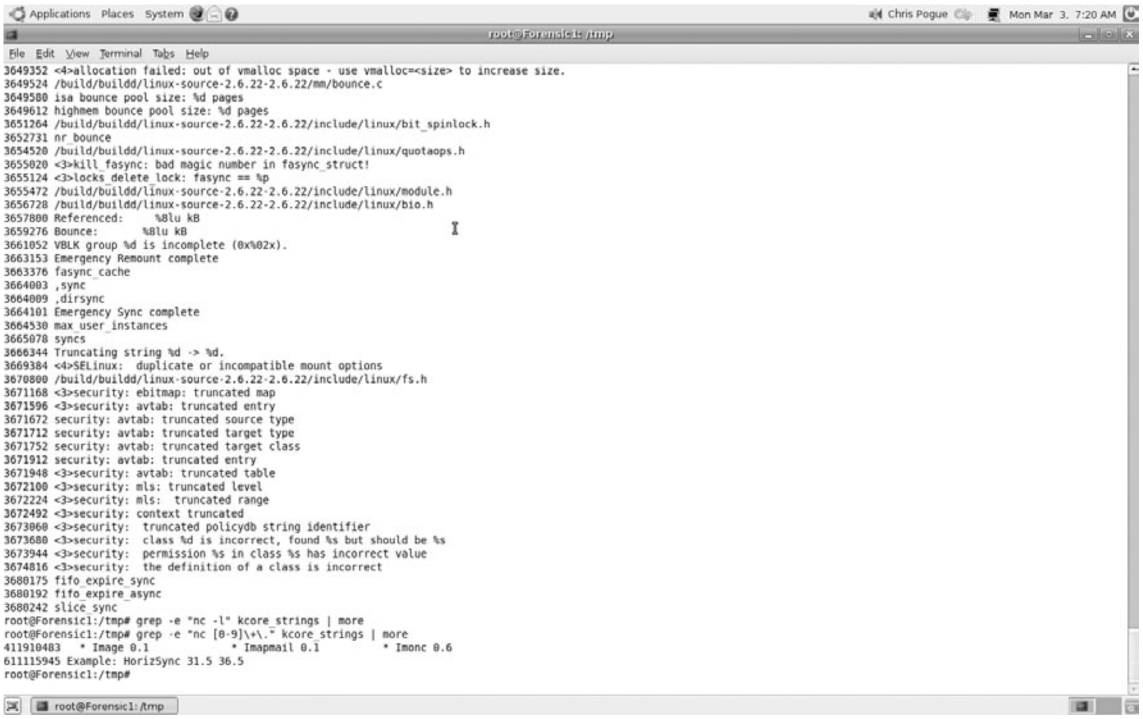
Figure 4.6 No Hits from Search



```
3647817  free:%lu slab:%lu mapped:%lu pagetables:%lu bounce:%lu
3648232 /build/buildd/linux-source-2.6.22-2.6.22/mm/truncate.c
3649228 /build/buildd/linux-source-2.6.22-2.6.22/include/linux/swapops.h
3649352 <4>allocation failed: out of vmalloc space - use vmalloc=<size> to increase size.
3649524 /build/buildd/linux-source-2.6.22-2.6.22/mm/bounce.c
3649580 isa bounce pool size: %d pages
3649612 highmem bounce pool size: %d pages
3651264 /build/buildd/linux-source-2.6.22-2.6.22/include/linux/bit_spinlock.h
3652731 nr_bounce
3654520 /build/buildd/linux-source-2.6.22-2.6.22/include/linux/quotaops.h
3655020 <3>kill_fasync: bad magic number in fasync_struct!
3655124 <3>locks_delete_lock: fasync == %p
3655472 /build/buildd/linux-source-2.6.22-2.6.22/include/linux/module.h
3656728 /build/buildd/linux-source-2.6.22-2.6.22/include/linux/bio.h
3657800 Referenced:     %8lu kB
3659276 Bounce:         %8lu kB
3661052 VBLK group %d is incomplete (0x%02x).
3663153 Emergency Remount complete
3663376 fasync_cache
3664003 ,sync
3664009 ,dirsync
3664101 Emergency Sync complete
3664530 max_user_instances
3665078 syncs
3666344 Truncating string %d -> %d.
3669384 <4>SELinux:  duplicate or incompatible mount options
3670800 /build/buildd/linux-source-2.6.22-2.6.22/include/linux/fs.h
3671168 <3>security: ebitmap: truncated map
3671596 <3>security: avtab: truncated entry
3671672 security: avtab: truncated source type
3671712 security: avtab: truncated target type
3671752 security: avtab: truncated target class
3671912 security: avtab: truncated entry
3671948 <3>security: avtab: truncated table
3672100 <3>security: mls: truncated level
3672224 <3>security: mls:  truncated range
3672492 <3>security: context truncated
3673060 <3>security:  truncated policydb string identifier
3673680 <3>security:  class %d is incorrect, found %s but should be %s
3673944 <3>security:  permission %s in class %s has incorrect value
3674816 <3>security:  the definition of a class is incorrect
3680175 fifo_expire_sync
3680192 fifo_expire_async
3680242 slice_sync
root@Forensic1:/tmp# grep -e "nc -l" kcore_strings | more
root@Forensic1:/tmp# 
```

Next, as seen in Figure 4.7, I searched to see if the host had been used to send a file with netcat.

Figure 4.7 New Search


Applications Places System
Chris Pogue Mon Mar 3, 7:20 AM
root@Forensic1: /tmp
File Edit View Terminal Tabs Help
root@Forensic1: /tmp

```
3649352 <4>allocation failed: out of vmalloc space - use vmalloc=<size> to increase size.
3649524 /build/buildd/linux-source-2.6.22-2.6.22/mm/bounce.c
3649580 isa bounce pool size: %d pages
3649612 highmem bounce pool size: %d pages
3651264 /build/buildd/linux-source-2.6.22-2.6.22/include/linux/bit_spinlock.h
3652731 nr_bounce
3654520 /build/buildd/linux-source-2.6.22-2.6.22/include/linux/quotaops.h
3655020 <3>kill_fasync: bad magic number in fasync_struct!
3655124 <3>locks_delete_lock: fasync == %p
3655472 /build/buildd/linux-source-2.6.22-2.6.22/include/linux/module.h
3656728 /build/buildd/linux-source-2.6.22-2.6.22/include/linux/bio.h
3657800 Referenced:     %8lu kB
3659276 Bounce:         %8lu kB
3661052 VBLK group %d is incomplete (0x%02x).
3663153 Emergency Remount complete
3663376 fasync_cache
3664003 ,sync
3664009 ,dirsync
3664101 Emergency Sync complete
3664530 max_user_instances
3665078 syncs
3666344 Truncating string %d -> %d.
3669384 <4>SELinux:  duplicate or incompatible mount options
3670800 /build/buildd/linux-source-2.6.22-2.6.22/include/linux/fs.h
3671168 <3>security: ebitmap: truncated map
3671596 <3>security: avtab: truncated entry
3671672 security: avtab: truncated source type
3671712 security: avtab: truncated target type
3671752 security: avtab: truncated target class
3671912 security: avtab: truncated entry
3671948 <3>security: avtab: truncated table
3672100 <3>security: mls: truncated level
3672224 <3>security: mls:  truncated range
3672492 <3>security: context truncated
3673060 <3>security:  truncated policydb string identifier
3673680 <3>security:  class %d is incorrect, found %s but should be %s
3673944 <3>security:  permission %s in class %s has incorrect value
3674816 <3>security:  the definition of a class is incorrect
3680175 fifo_expire_sync
3680192 fifo_expire_async
3680242 slice_sync
root@Forensic1:/tmp# grep -e "nc -l" kcore_strings | more
root@Forensic1:/tmp# grep -e "nc [0-9]\+\." kcore_strings | more
411910483   * Image 0.1             * Imapmail 0.1          * Imonc 0.6
611115945 Example: HorizSync 31.5 36.5
root@Forensic1:/tmp#
```

So while my search did return a hit with, "nc ##.##," it was obviously not an Internet Protocol (IP) address. So, now I can also safely assume that the host had not been used to send a file using netcat.

Keyword searching is as much an art as it is a science. You need to develop an understanding of how the system is supposed to work, where things are supposed to be stored, and how they are supposed to look before you will effectively be able to spot anomalies. Build your own list of keywords as you work cases for items you have found, as you will likely see them again. Additionally, it is always a good idea to use some kind of virtualization utility or a test box to perform fingerprinting. By doing this, you can see where the default locations are for many of the utilities commonly used by hackers. This subject will actually be covered in depth in Chapter 5: "The Hacking Underground."

Again, knowing what the standard operating parameters are is critical! I cannot stress that point enough!

User Activity

Shell History

Knowing who has done what is always of the utmost concern in any investigation. Fortunately for us, by default, Linux keeps a trail of user activity in the shell history, located in the */home/<user>* directory. Remember, the shell history is only a recording of one side of the conversation. It does not show you how the system responded to the particular command. So while gathering information about what was typed in at the command line is a good information gathering technique, like anything else, it should be used in conjunction with other data retrieved from the investigation.

In my examples, I am again using an Ubuntu 7.10 (Gutsy) distribution, which uses the BASH shell by default. To find the history files that exist on my system, I simply type:

```
locate bash_history
```

Understand that this command will only give you the *.bash_history* files. There are other shells which will create other history files. The most popular shells store their history files in the following locations with the */home/<user>* directory:

- **BASH** .bash_history
- **C-Shell** history.csh
- **Korn** .sh_history
- **POSIX** .sh_history
- **Z-Shell** .history

Again, by default, most Linux variants maintain a 500-line command history. To view the current command history of a system, type:

```
echo $HISTSIZE
```

Like anything else in Linux, this environment variable is configurable within the .profile of the individual user. If you find that the HISTSIZE has been modified from the default value, take note of it, and follow up with the customer's system administrator to find out if this was a configuration change on their part, or something that was done maliciously by an intruder (especially if the value has been set to zero).

Included on your tools disk are two scripts I have written to make parsing through user shell history files a bit easier. The first script is called "history_search.sh."

It takes the commands from all of the user history files on the local host, regardless of which shell has been used, removes the duplicate entries, and puts them into a single file in the current working directory called "outfile". You can use this file to review all of the commands used on the target host and determine if any of them requires further investigation. For example, if a command is found such as, "msf", indicating that the Metasploit Framework binary was invoked from the command line, you can then use the second script, "user_driller.sh" to find out which user(s) typed in that specific command. This script will create a directory called *driller* in a user-specified location; however, the default is the current working directory.

NOTE

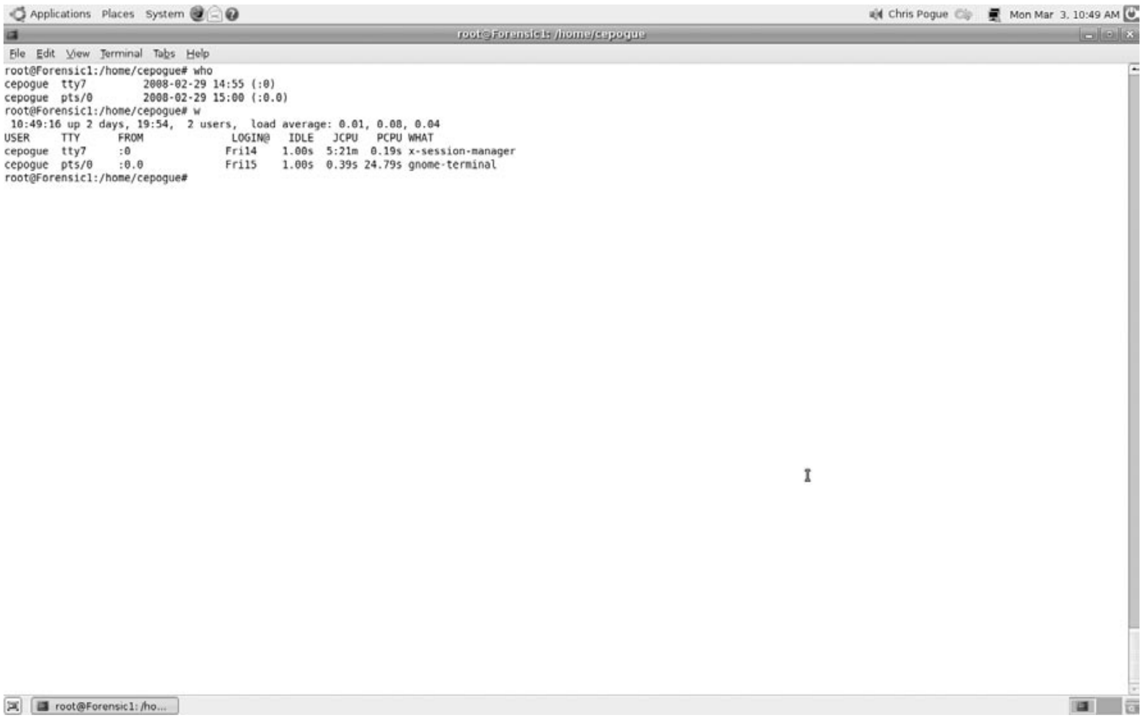
One of the limitations of the shell history files is that other than the Media Access Control (MAC) time, there are no timestamps within the file itself. So while knowing what was typed in can be useful for formulating an idea of what may have happened on the host, other correlative measures will have to be taken to determine when those specific actions took place. Also tying a username to a command, or series commands only shows which username was used, not necessarily which user was actually using that account. A good hacker will most likely use an existing user account to perform his nefarious tasks. This means that log file correlations will be of the utmost importance for you to be able to piece together the different aspects of the investigation.

Logged on Users

When analyzing volatile data it is import to know which users are currently logged onto a system. Understand that most intruders are not so dumb as to create a user ID called, "hacker." They will more than likely use an existing user ID to conduct their illicit activities. Just like with shell history, additional chronological correlations will have to be made to determine if the activities were part of normal business operations, or if they were the work of an intruder.

The output from the "who" and "w" commands are shown below in Figure 4.8. The results from the "w" may require a bit more information to fully comprehend what the user is looking at.

Figure 4.8 Output from "who" and "w"


Applications Places System
Chris Pogue
Mon Mar 3, 10:49 AM
root@Forensic1: /home/cepogue
File Edit View Terminal Tabs Help
root@Forensic1: /ho...

```
root@Forensic1:/home/cepogue# who
cepogue  tty7         2008-02-29 14:55 (:0)
cepogue  pts/0        2008-02-29 15:00 (:0.0)
root@Forensic1:/home/cepogue# w
 10:49:16 up 2 days, 19:54,  2 users,  load average: 0.01, 0.08, 0.04
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
cepogue  tty7     :0               Fri14    1.00s  5:21m  0.19s x-session-manager
cepogue  pts/0    :0.0             Fri15    1.00s  0.39s 24.79s gnome-terminal
root@Forensic1:/home/cepogue#
```

The header is pretty self explanatory. The fields shown are the current time, how long the system has been running, how many users are currently logged on, and the system load averages. However, starting at the usernames, let's go into a bit more depth on what each of the items listed actually mean.

- **User** Username
- **TTY Teletype** In the old days of computing, terminals were keyboards that were attached directly to printers called "teletypes." The output from commands issued to the system were read from the teletype. Where tty is either zero or a positive integer signifies a login from the console. Where tty is either pts or ttyp#, it signifies a login over the network.

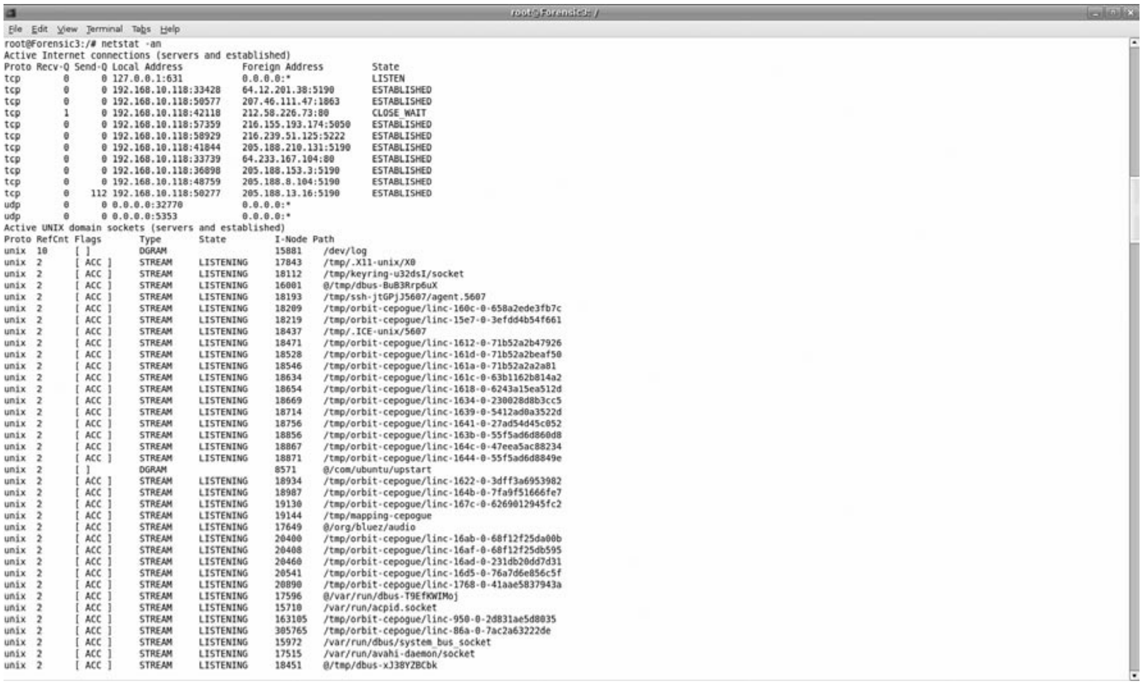
- **From** This field shows where the user is logging in. In Figure 4.8, the ":0" and ":0.0" indicates that I am logged in from the console. If, for example, I was Secure Shelled (SSH'd) into the localhost from another box on the network, then that IP address or fully qualified domain name (FQDN) would appear in place of the "0.0."
- **Login@** This is pretty self explanatory. It shows the time of the last login.
- **Idle** This shows how long it has been since the last user activity. This information can be of specific note if you see users with long idle times.
- **JCPU** The JCPU time is the time used by all processes attached to the tty. It does not include past background jobs, but does include currently running background jobs.
- **PCPU** The PCPU time is the time used by the current process, named in the "what" field.
- **What** The What is the process that the user is currently running.

Network Connections

Recall from the introduction, I referred to the lack of information sometimes being called "negative evidence." While the data necessary to prove negative evidence is mainly found in the customer's network logs, the local network connections can prove to be equally as useful.

By running the netstat command with the "–an" and "–rn" switches, you can see which connections are being made to and from the host as well as on which port is being used and the state of that connection (either LISTEN, ESTABLISHED, or CLOSE_WAIT).

Figure 4.9 Established Outbound Connections



```
root@Forensic3:/# netstat -an
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 192.168.10.118:33428    64.12.201.38:5190       ESTABLISHED
tcp        0      0 192.168.10.118:50577    207.46.111.47:1863      ESTABLISHED
tcp        1      0 192.168.10.118:42118    212.58.226.73:80        CLOSE_WAIT
tcp        0      0 192.168.10.118:57359    216.155.193.174:5050    ESTABLISHED
tcp        0      0 192.168.10.118:58929    216.239.51.125:5222     ESTABLISHED
tcp        0      0 192.168.10.118:41844    205.188.210.131:5190    ESTABLISHED
tcp        0      0 192.168.10.118:33739    64.233.167.104:80       ESTABLISHED
tcp        0      0 192.168.10.118:36898    205.188.153.3:5190      ESTABLISHED
tcp        0      0 192.168.10.118:48759    205.188.8.104:5190      ESTABLISHED
tcp        0    112 192.168.10.118:50277    205.188.13.16:5190      ESTABLISHED
udp        0      0 0.0.0.0:32770           0.0.0.0:*
udp        0      0 0.0.0.0:5353            0.0.0.0:*
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node Path
unix  10     [ ]         DGRAM                    15881    /dev/log
unix  2      [ ACC ]     STREAM     LISTENING     17843    /tmp/.X11-unix/X0
unix  2      [ ACC ]     STREAM     LISTENING     18112    /tmp/keyring-u32dsI/socket
unix  2      [ ACC ]     STREAM     LISTENING     16001    @/tmp/dbus-BuB3Rrp6uX
unix  2      [ ACC ]     STREAM     LISTENING     18193    /tmp/ssh-jtGPjJ5607/agent.5607
unix  2      [ ACC ]     STREAM     LISTENING     18209    /tmp/orbit-cepogue/linc-160c-0-658a2ede3fb7c
unix  2      [ ACC ]     STREAM     LISTENING     18219    /tmp/orbit-cepogue/linc-15e7-0-3efdd4b54f661
unix  2      [ ACC ]     STREAM     LISTENING     18437    /tmp/.ICE-unix/5607
unix  2      [ ACC ]     STREAM     LISTENING     18471    /tmp/orbit-cepogue/linc-1612-0-71b52a2b47926
unix  2      [ ACC ]     STREAM     LISTENING     18528    /tmp/orbit-cepogue/linc-161d-0-71b52a2beaf50
unix  2      [ ACC ]     STREAM     LISTENING     18546    /tmp/orbit-cepogue/linc-161a-0-71b52a2a2a81
unix  2      [ ACC ]     STREAM     LISTENING     18634    /tmp/orbit-cepogue/linc-161c-0-63b1162b814a2
unix  2      [ ACC ]     STREAM     LISTENING     18654    /tmp/orbit-cepogue/linc-1618-0-6243a15ea512d
unix  2      [ ACC ]     STREAM     LISTENING     18669    /tmp/orbit-cepogue/linc-1634-0-230028d8b3cc5
unix  2      [ ACC ]     STREAM     LISTENING     18714    /tmp/orbit-cepogue/linc-1639-0-5412ad0a3522d
unix  2      [ ACC ]     STREAM     LISTENING     18756    /tmp/orbit-cepogue/linc-1641-0-27ad54d45c052
unix  2      [ ACC ]     STREAM     LISTENING     18856    /tmp/orbit-cepogue/linc-163b-0-55f5ad6d860d8
unix  2      [ ACC ]     STREAM     LISTENING     18867    /tmp/orbit-cepogue/linc-164c-0-47eea5ac88234
unix  2      [ ACC ]     STREAM     LISTENING     18871    /tmp/orbit-cepogue/linc-1644-0-55f5ad6d8849e
unix  2      [ ]         DGRAM                    8571     @/com/ubuntu/upstart
unix  2      [ ACC ]     STREAM     LISTENING     18934    /tmp/orbit-cepogue/linc-1622-0-3dff3a6953982
unix  2      [ ACC ]     STREAM     LISTENING     18987    /tmp/orbit-cepogue/linc-164b-0-7fa9f51666fe7
unix  2      [ ACC ]     STREAM     LISTENING     19130    /tmp/orbit-cepogue/linc-167c-0-6269012945fc2
unix  2      [ ACC ]     STREAM     LISTENING     19144    /tmp/mapping-cepogue
unix  2      [ ACC ]     STREAM     LISTENING     17649    @/org/bluez/audio
unix  2      [ ACC ]     STREAM     LISTENING     20400    /tmp/orbit-cepogue/linc-16ab-0-68f12f25da00b
unix  2      [ ACC ]     STREAM     LISTENING     20408    /tmp/orbit-cepogue/linc-16af-0-68f12f25db595
unix  2      [ ACC ]     STREAM     LISTENING     20460    /tmp/orbit-cepogue/linc-16ad-0-231db20dd7d31
unix  2      [ ACC ]     STREAM     LISTENING     20541    /tmp/orbit-cepogue/linc-16d5-0-76a7d6e856c5f
unix  2      [ ACC ]     STREAM     LISTENING     20890    /tmp/orbit-cepogue/linc-1768-0-41aae5837943a
unix  2      [ ACC ]     STREAM     LISTENING     17596    @/var/run/dbus-T9EfKWIMoj
unix  2      [ ACC ]     STREAM     LISTENING     15710    /var/run/acpid.socket
unix  2      [ ACC ]     STREAM     LISTENING     163105   /tmp/orbit-cepogue/linc-950-0-2d831ae5d8035
unix  2      [ ACC ]     STREAM     LISTENING     305765   /tmp/orbit-cepogue/linc-86a-0-7ac2a63222de
unix  2      [ ACC ]     STREAM     LISTENING     15972    /var/run/dbus/system_bus_socket
unix  2      [ ACC ]     STREAM     LISTENING     17515    /var/run/avahi-daemon/socket
unix  2      [ ACC ]     STREAM     LISTENING     18451    @/tmp/dbus-xJ38YZBCbk
```

The results of the netstat "–an" command are separated into two sections, Active Connections and Domain Sockets. The Active Connections are separated into six columns, however, for our purposes we are only interested in four, Proto (Protocol), Local Address, Foreign Address, and State. As you can see from Figure 4.9, I have several established outbound connections to various destinations on various ports. Obviously the connections on port 80 are Web, while ports 5190, 1863, 5050, and 5222 are instant messaging via Pidgin. I was able to confirm this by running ARIN WHOIS queries against the IP addresses in the Foreign Address column.

In your case, this information will be different, but will show you the same type of information. Knowing the connection status of your machine is of critical importance to your case, and can provide key insight into specifics of the incident.

The second section, Active Unix Domain Sockets, has the following breakdown:

- **Proto** Protocol (usually UNIX) used
- **RefCnt** Reference count (i.e., attached processes via this socket).
- **Flags** Flags displayed is SO_ACCEPTON (displayed as ACC), SO_WAITDATA (W), or SO_NOSPACE (N). SO_ACCECPTON is used

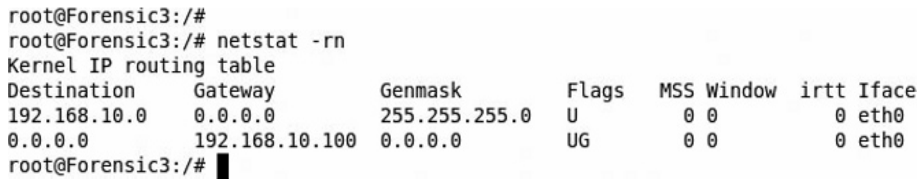
on unconnected sockets if their corresponding processes are waiting for a connect request. The other flags are not of normal interest.

- **Type** Types of socket access:
  - **DGRAM** Used in Datagram (connectionless) mode
  - **STREAM** Stream (connection) socket
  - **RAW** Raw socket
  - **RDM** Reliably-delivered messages
  - **SEQPACKET** Sequential packet socket
  - **PACKET** RAW Interface access socket
- **State:**
  - **FREE** Not allocated
  - **LISTENING** Listening for a connection request. Those sockets are only displayed if the -a switch is set.
  - **CONNECTING** About to establish a connection.
  - **CONNECTED** Connected.
  - **DISCONNECTING** Disconnecting.
  - **(empty)** Not connected to another socket.

The last two columns are I-Node and Path, which identify the process attached to the socket. Since there are likely to be a whole slew of entries, and you will likely have no clue which ones are normal and which are not, be sure to get with the customer's system and network administrators to help you establish a baseline of "normal" operating parameters. Additionally, simply using an Internet search engine will provide you with good information regarding the nature of a process. Since there are so many, this should be reserved only for processes that stand out as being potentially outside of what the customer administrator teams determine to be normal.

The netstat command with the "-rn" switch will display the hosts' routing table. (See Figure 4.10.)

Figure 4.10 Displaying the Host Routing Table



```
root@Forensic3:/#
root@Forensic3:/# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
192.168.10.0    0.0.0.0         255.255.255.0   U         0 0          0 eth0
0.0.0.0         192.168.10.100  0.0.0.0         UG        0 0          0 eth0
root@Forensic3:/# 
```

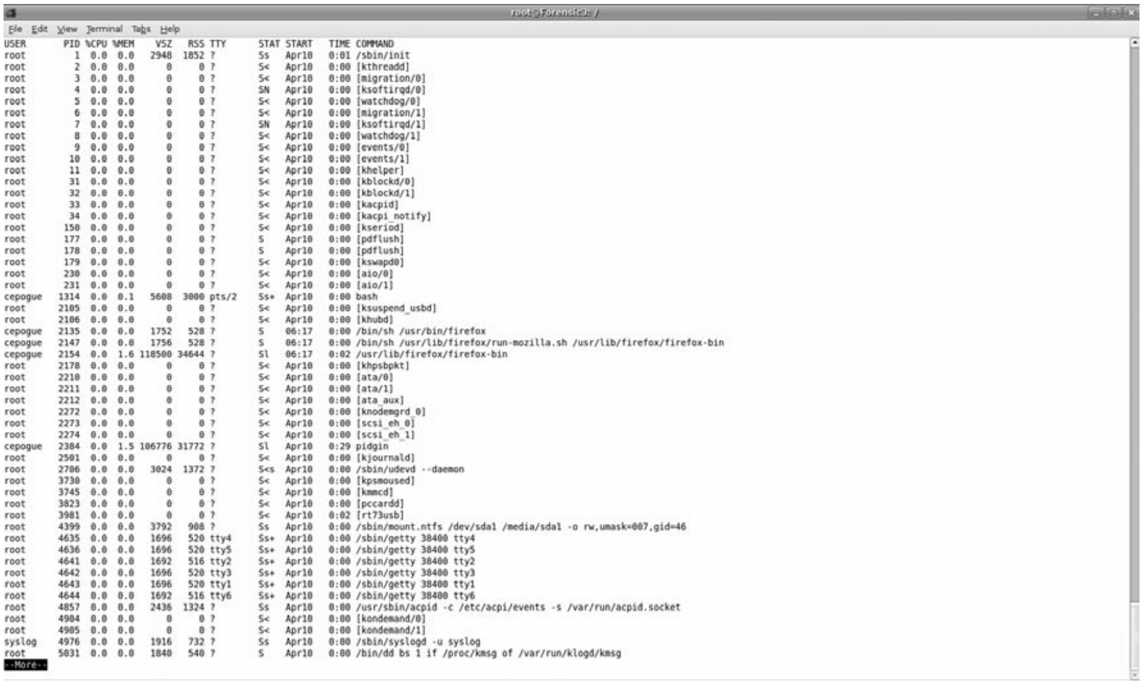
As you can see from Figure 4.10, this command yields eight columns. The first two are pretty easy to decipher, where the route is headed and which gateway is being used. In the event that no gateway is used, an asterix (*) will appear in that column. The next column shows the "generality" of the route, or in other words, the network mask for that particular route. The next column displays the flags that can be set. The flag breakdown is:

- **G** Gateway
- **U** The interface being used is **Up**
- **H** Only a single **Host** can be reached through the route, like the loopback
- **D** The route has been **Dynamically** created
- **M Modified** by an Internet Control Message Protocol (ICMP) redirect
- **!** The route is a reject, and the packets will be dropped

Running Processes

It is important to know what is running on the host you are analyzing. This can be accomplished by a few different commands. For the purposes of this book, I only cover "ps aux" and "top." (See Figure 4.11.)

Figure 4.11 "ps aux" and "top"



```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   2948  1852 ?        Ss   Apr10   0:01 /sbin/init
root         2  0.0  0.0      0     0 ?        S<   Apr10   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S<   Apr10   0:00 [migration/0]
root         4  0.0  0.0      0     0 ?        SN   Apr10   0:00 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   Apr10   0:00 [watchdog/0]
root         6  0.0  0.0      0     0 ?        S<   Apr10   0:00 [migration/1]
root         7  0.0  0.0      0     0 ?        SN   Apr10   0:00 [ksoftirqd/1]
root         8  0.0  0.0      0     0 ?        S<   Apr10   0:00 [watchdog/1]
root         9  0.0  0.0      0     0 ?        S<   Apr10   0:00 [events/0]
root        10  0.0  0.0      0     0 ?        S<   Apr10   0:00 [events/1]
root        11  0.0  0.0      0     0 ?        S<   Apr10   0:00 [khelper]
root        31  0.0  0.0      0     0 ?        S<   Apr10   0:00 [kblockd/0]
root        32  0.0  0.0      0     0 ?        S<   Apr10   0:00 [kblockd/1]
root        33  0.0  0.0      0     0 ?        S<   Apr10   0:00 [kacpid]
root        34  0.0  0.0      0     0 ?        S<   Apr10   0:00 [kacpi_notify]
root       150  0.0  0.0      0     0 ?        S<   Apr10   0:00 [kseriod]
root       177  0.0  0.0      0     0 ?        S    Apr10   0:00 [pdflush]
root       178  0.0  0.0      0     0 ?        S    Apr10   0:00 [pdflush]
root       179  0.0  0.0      0     0 ?        S<   Apr10   0:00 [kswapd0]
root       230  0.0  0.0      0     0 ?        S<   Apr10   0:00 [aio/0]
root       231  0.0  0.0      0     0 ?        S<   Apr10   0:00 [aio/1]
cepogue   1314  0.0  0.1   5608  3000 pts/2    Ss+  Apr10   0:00 bash
root      2105  0.0  0.0      0     0 ?        S<   Apr10   0:00 [ksuspend_usbd]
root      2106  0.0  0.0      0     0 ?        S<   Apr10   0:00 [khubd]
cepogue   2135  0.0  0.0   1752   528 ?        S    06:17   0:00 /bin/sh /usr/bin/firefox
cepogue   2147  0.0  0.0   1756   528 ?        S    06:17   0:00 /bin/sh /usr/lib/firefox/run-mozilla.sh /usr/lib/firefox/firefox-bin
cepogue   2154  0.0  1.6 118500 34644 ?        Sl   06:17   0:02 /usr/lib/firefox/firefox-bin
root      2178  0.0  0.0      0     0 ?        S<   Apr10   0:00 [khpsbpkt]
root      2210  0.0  0.0      0     0 ?        S<   Apr10   0:00 [ata/0]
root      2211  0.0  0.0      0     0 ?        S<   Apr10   0:00 [ata/1]
root      2212  0.0  0.0      0     0 ?        S<   Apr10   0:00 [ata_aux]
root      2272  0.0  0.0      0     0 ?        S<   Apr10   0:00 [knodemgrd_0]
root      2273  0.0  0.0      0     0 ?        S<   Apr10   0:00 [scsi_eh_0]
root      2274  0.0  0.0      0     0 ?        S<   Apr10   0:00 [scsi_eh_1]
cepogue   2384  0.0  1.5 106776 31772 ?        Sl   Apr10   0:29 pidgin
root      2501  0.0  0.0      0     0 ?        S<   Apr10   0:00 [kjournald]
root      2706  0.0  0.0   3024  1372 ?        S<s  Apr10   0:00 /sbin/udevd --daemon
root      3730  0.0  0.0      0     0 ?        S<   Apr10   0:00 [kpsmoused]
root      3745  0.0  0.0      0     0 ?        S<   Apr10   0:00 [kmmcd]
root      3823  0.0  0.0      0     0 ?        S<   Apr10   0:00 [pccardd]
root      3981  0.0  0.0      0     0 ?        S<   Apr10   0:02 [rt73usb]
root      4399  0.0  0.0   3792   908 ?        Ss   Apr10   0:00 /sbin/mount.ntfs /dev/sda1 /media/sda1 -o rw,umask=007,gid=46
root      4635  0.0  0.0   1696   520 tty4     Ss+  Apr10   0:00 /sbin/getty 38400 tty4
root      4636  0.0  0.0   1696   520 tty5     Ss+  Apr10   0:00 /sbin/getty 38400 tty5
root      4641  0.0  0.0   1692   516 tty2     Ss+  Apr10   0:00 /sbin/getty 38400 tty2
root      4642  0.0  0.0   1696   520 tty3     Ss+  Apr10   0:00 /sbin/getty 38400 tty3
root      4643  0.0  0.0   1696   520 tty1     Ss+  Apr10   0:00 /sbin/getty 38400 tty1
root      4644  0.0  0.0   1692   516 tty6     Ss+  Apr10   0:00 /sbin/getty 38400 tty6
root      4857  0.0  0.0   2436  1324 ?        Ss   Apr10   0:00 /usr/sbin/acpid -c /etc/acpi/events -s /var/run/acpid.socket
root      4904  0.0  0.0      0     0 ?        S<   Apr10   0:00 [kondemand/0]
root      4905  0.0  0.0      0     0 ?        S<   Apr10   0:00 [kondemand/1]
syslog    4976  0.0  0.0   1916   732 ?        Ss   Apr10   0:00 /sbin/syslogd -u syslog
root      5031  0.0  0.0   1840   540 ?        S    Apr10   0:00 /bin/dd bs 1 if /proc/kmsg of /var/run/klogd/kmsg
--More--
```

The "ps aux" command shows all running processes using the BSD syntax, and as you can see from Figure 4.11, the output is separated into 11 columns. Paramount among these, at least in terms of a forensic investigation, are the "USER", "TTY", "START", "TIME", and "COMMAND." Each will be important for different reasons depending on what you are trying to determine, and can easily be parsed with a text utility (like Textpad[1]) or from the command line using "grep." Whatever the case may be, these entries will tell you who started the process, from where, when, and the command that was used. This information can also be used in conjunction with data from the user shell history and the network log files for correlating certain events. For example, in Figure 4.11, you can see that the user "cepogue" started a process called "Pidgin" which was started on "Apr10," has been running for "0:34" minutes, and was launched with the command, "pidgin." Some items of note within this information, however, are the time and the command.

The time shows that the process has only been running for 34 minutes. What this means is that this column denotes the amount of time the process has been running

[1] www.textpad.com/

on the CPU and not the time that has elapsed since the program was launched, since most programs spend a great deal of time waiting for other stuff to happen before they actually need time on the CPU.

The command column in this example simply shows a single word, "pidgin." This can mean one of two things. Either the binary is in my user path, or I launched it from a graphical user interface (GUI). In the case, the latter is true. Had I ran the binary from the command line using the full path, the entry in the "COMMAND" column would have read */usr/bin/pidgin*.

The process state code, shown as "stat," is not something you will likely use during an investigation. The codes show what state the process is currently in, or at least the time you issued the command. The codes are:

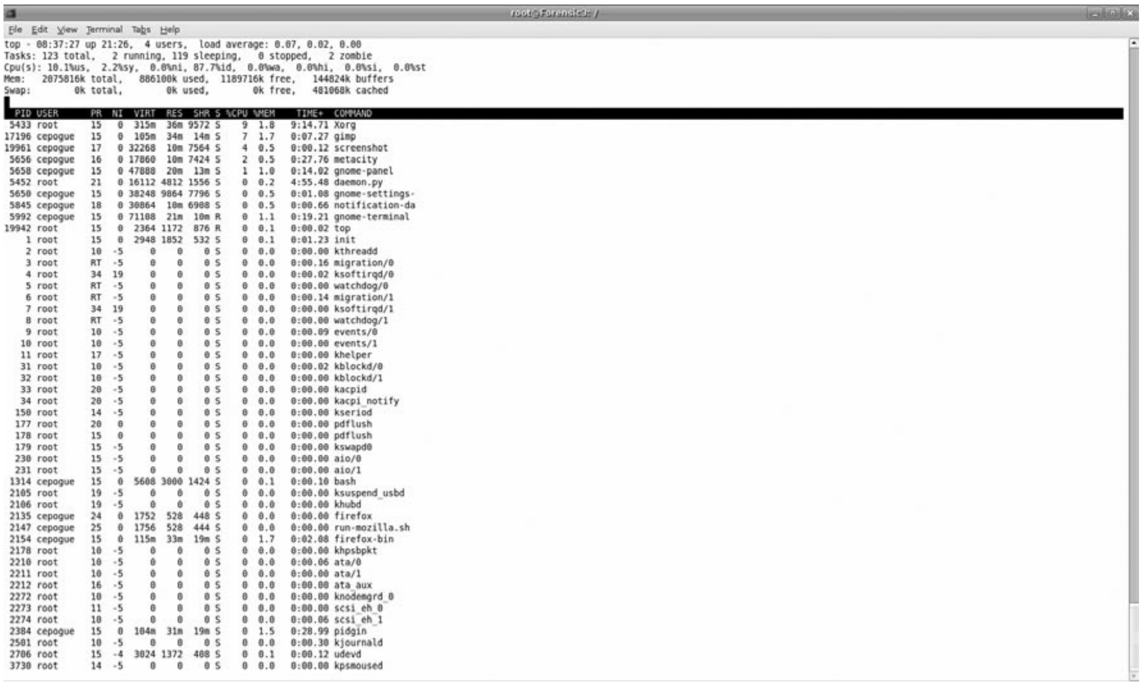
- **D** Uninterruptible sleep (usually IO)
- **R** Running or runnable (on run queue)
- **S** Interruptible sleep (waiting for an event to complete)
- **T** Stopped, either by a job control signal or because it is being traced
- **W** Paging (not valid since the 2.6.xx kernel)
- **X** Dead (should never be seen)
- **Z** Defunct ("zombie") process, terminated but not reaped by its parent

For BSD formats and when the stat keyword is used, additional characters may be displayed as:

- **<** high-priority (not nice to other users)
- **N** low-priority (nice to other users)
- **L** has pages locked into memory (for real-time and custom IO)
- **s** is a session leader
- **l** is multi-threaded (using CLONE_THREAD, like NPTL pthreads do)
- **+** is in the foreground process group

The "top" command shows exactly what you think it would show, the running processes that are using the most CPU. (See Figure 4.12.)

Figure 4.12 Running Processes Using the Most CPU



```
top - 08:37:27 up 21:26,  4 users,  load average: 0.07, 0.02, 0.00
Tasks: 123 total,   2 running, 119 sleeping,   0 stopped,   2 zombie
Cpu(s): 10.1%us,  2.2%sy,  0.0%ni, 87.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   2075816k total,   886100k used,  1189716k free,   144824k buffers
Swap:        0k total,        0k used,        0k free,   481068k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 5433 root      15   0  315m  36m 9572 S    9  1.8   9:14.71 Xorg
17196 cepogue   15   0  105m  34m  14m S    7  1.7   0:07.27 gimp
19961 cepogue   17   0 32268  10m 7564 S    4  0.5   0:00.12 screenshot
 5656 cepogue   16   0 17860  10m 7424 S    2  0.5   0:27.76 metacity
 5658 cepogue   15   0 47888  20m  13m S    1  1.0   0:14.02 gnome-panel
 5452 root      21   0 16112 4812 1556 S    0  0.2   4:55.48 daemon.py
 5650 cepogue   15   0 38248 9864 7796 S    0  0.5   0:01.08 gnome-settings-
 5845 cepogue   18   0 30864  10m 6988 S    0  0.5   0:00.66 notification-da
 5992 cepogue   15   0 71108  21m  10m R    0  1.1   0:19.21 gnome-terminal
19942 root      15   0  2364 1172  876 R    0  0.1   0:00.02 top
    1 root      15   0  2948 1852  532 S    0  0.1   0:01.23 init
    2 root      10  -5     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S    0  0.0   0:00.16 migration/0
    4 root      34  19     0    0    0 S    0  0.0   0:00.02 ksoftirqd/0
    5 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/0
    6 root      RT  -5     0    0    0 S    0  0.0   0:00.14 migration/1
    7 root      34  19     0    0    0 S    0  0.0   0:00.00 ksoftirqd/1
    8 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/1
    9 root      10  -5     0    0    0 S    0  0.0   0:00.09 events/0
   10 root      10  -5     0    0    0 S    0  0.0   0:00.00 events/1
   11 root      17  -5     0    0    0 S    0  0.0   0:00.00 khelper
   31 root      10  -5     0    0    0 S    0  0.0   0:00.02 kblockd/0
   32 root      10  -5     0    0    0 S    0  0.0   0:00.00 kblockd/1
   33 root      20  -5     0    0    0 S    0  0.0   0:00.00 kacpid
   34 root      20  -5     0    0    0 S    0  0.0   0:00.00 kacpi_notify
  150 root      14  -5     0    0    0 S    0  0.0   0:00.00 kseriod
  177 root      20   0     0    0    0 S    0  0.0   0:00.00 pdflush
  178 root      15   0     0    0    0 S    0  0.0   0:00.00 pdflush
  179 root      15  -5     0    0    0 S    0  0.0   0:00.00 kswapd0
  230 root      15  -5     0    0    0 S    0  0.0   0:00.00 aio/0
  231 root      15  -5     0    0    0 S    0  0.0   0:00.00 aio/1
 1314 cepogue   15   0  5608 3000 1424 S    0  0.1   0:00.10 bash
 2105 root      19  -5     0    0    0 S    0  0.0   0:00.00 ksuspend_usbd
 2106 root      19  -5     0    0    0 S    0  0.0   0:00.00 khubd
 2135 cepogue   24   0  1752  528  448 S    0  0.0   0:00.00 firefox
 2147 cepogue   25   0  1756  528  444 S    0  0.0   0:00.00 run-mozilla.sh
 2154 cepogue   15   0  115m  33m  19m S    0  1.7   0:02.08 firefox-bin
 2178 root      10  -5     0    0    0 S    0  0.0   0:00.00 khpsbpkt
 2210 root      10  -5     0    0    0 S    0  0.0   0:00.06 ata/0
 2211 root      10  -5     0    0    0 S    0  0.0   0:00.00 ata/1
 2212 root      16  -5     0    0    0 S    0  0.0   0:00.00 ata_aux
 2272 root      10  -5     0    0    0 S    0  0.0   0:00.00 knodemgrd_0
 2273 root      11  -5     0    0    0 S    0  0.0   0:00.00 scsi_eh_0
 2274 root      10  -5     0    0    0 S    0  0.0   0:00.06 scsi_eh_1
 2384 cepogue   15   0  104m  31m  19m S    0  1.5   0:28.99 pidgin
 2501 root      10  -5     0    0    0 S    0  0.0   0:00.30 kjournald
 2706 root      15  -4  3024 1372  408 S    0  0.1   0:00.12 udevd
 3730 root      14  -5     0    0    0 S    0  0.0   0:00.00 kpsmoused
```

Figure 4.12 shows the output from the "top" command on my local host. As you can see, the results are 12 columns, which don't look entirely unlike the same columns we saw from the "ps" results. Again, the columns of the most interest in an investigation are "PID, "USER," "TIME+," and "COMMAND."

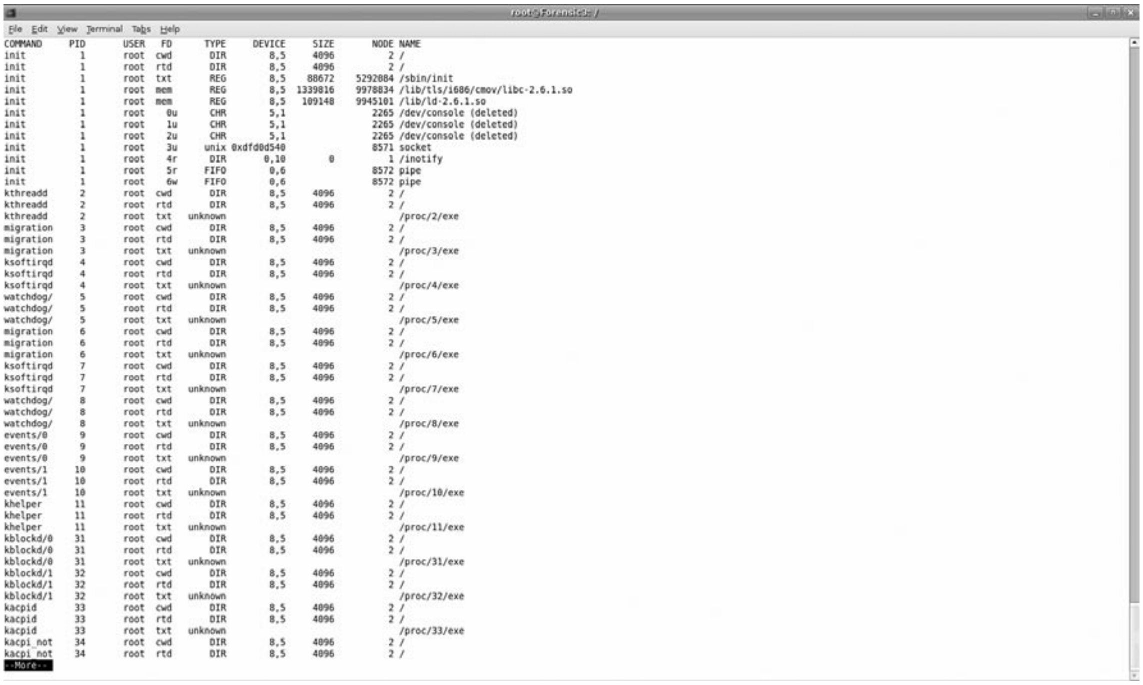
You will see a bit of a difference in the "TIME" column from the "ps" command and the "TIME+" shown in Figure 4.12. "Top" shows more granularity by listing the CPU time down to the hundredth of a second.

There are also two additional columns which again, don't hold any real forensic value, but I will explain so that you know what they are. The "PR" column stands for PRiority, and denotes well the priority of the task, and the "NI" column stands for NIce, and indicates the nice value of the task. With this value, the smaller the number, the less nice it is to the other processes, or the higher priority it has. A zero in this column simply means that the priority will not change when determining the task's dispatchability.

Open File Handlers

The "List Open Files" (lsof) command is used to show which files were and are opened by which processes. (See Figure 4.13.)

Figure 4.13 Open Files



```
COMMAND     PID      USER   FD      TYPE     DEVICE     SIZE       NODE NAME
init          1      root  cwd       DIR        8,5     4096          2 /
init          1      root  rtd       DIR        8,5     4096          2 /
init          1      root  txt       REG        8,5    88672    5292084 /sbin/init
init          1      root  mem       REG        8,5  1339816    9978834 /lib/tls/i686/cmov/libc-2.6.1.so
init          1      root  mem       REG        8,5   109148    9945101 /lib/ld-2.6.1.so
init          1      root    0u      CHR        5,1                2265 /dev/console (deleted)
init          1      root    1u      CHR        5,1                2265 /dev/console (deleted)
init          1      root    2u      CHR        5,1                2265 /dev/console (deleted)
init          1      root    3u     unix 0xdfd0d540                8571 socket
init          1      root    4r      DIR       0,10        0          1 /inotify
init          1      root    5r     FIFO        0,6                8572 pipe
init          1      root    6w     FIFO        0,6                8572 pipe
kthreadd      2      root  cwd       DIR        8,5     4096          2 /
kthreadd      2      root  rtd       DIR        8,5     4096          2 /
kthreadd      2      root  txt   unknown                                /proc/2/exe
migration     3      root  cwd       DIR        8,5     4096          2 /
migration     3      root  rtd       DIR        8,5     4096          2 /
migration     3      root  txt   unknown                                /proc/3/exe
ksoftirqd     4      root  cwd       DIR        8,5     4096          2 /
ksoftirqd     4      root  rtd       DIR        8,5     4096          2 /
ksoftirqd     4      root  txt   unknown                                /proc/4/exe
watchdog/     5      root  cwd       DIR        8,5     4096          2 /
watchdog/     5      root  rtd       DIR        8,5     4096          2 /
watchdog/     5      root  txt   unknown                                /proc/5/exe
migration     6      root  cwd       DIR        8,5     4096          2 /
migration     6      root  rtd       DIR        8,5     4096          2 /
migration     6      root  txt   unknown                                /proc/6/exe
ksoftirqd     7      root  cwd       DIR        8,5     4096          2 /
ksoftirqd     7      root  rtd       DIR        8,5     4096          2 /
ksoftirqd     7      root  txt   unknown                                /proc/7/exe
watchdog/     8      root  cwd       DIR        8,5     4096          2 /
watchdog/     8      root  rtd       DIR        8,5     4096          2 /
watchdog/     8      root  txt   unknown                                /proc/8/exe
events/0      9      root  cwd       DIR        8,5     4096          2 /
events/0      9      root  rtd       DIR        8,5     4096          2 /
events/0      9      root  txt   unknown                                /proc/9/exe
events/1     10      root  cwd       DIR        8,5     4096          2 /
events/1     10      root  rtd       DIR        8,5     4096          2 /
events/1     10      root  txt   unknown                                /proc/10/exe
khelper      11      root  cwd       DIR        8,5     4096          2 /
khelper      11      root  rtd       DIR        8,5     4096          2 /
khelper      11      root  txt   unknown                                /proc/11/exe
kblockd/0    31      root  cwd       DIR        8,5     4096          2 /
kblockd/0    31      root  rtd       DIR        8,5     4096          2 /
kblockd/0    31      root  txt   unknown                                /proc/31/exe
kblockd/1    32      root  cwd       DIR        8,5     4096          2 /
kblockd/1    32      root  rtd       DIR        8,5     4096          2 /
kblockd/1    32      root  txt   unknown                                /proc/32/exe
kacpid       33      root  cwd       DIR        8,5     4096          2 /
kacpid       33      root  rtd       DIR        8,5     4096          2 /
kacpid       33      root  txt   unknown                                /proc/33/exe
kacpi_not    34      root  cwd       DIR        8,5     4096          2 /
kacpi_not    34      root  rtd       DIR        8,5     4096          2 /
--More--
```

As can be seen in Figure 4.13, the output from this command, without any additional switches, is separated into nine columns. You can see that I had to pipe my original "lsof" command to "more" since it yielded so many lines of output. In fact, by sending the output to an outfile called "foo," then cat'ing that file and piping that out to "wc –l," I found that the standard "lsof" command provided me with 404 lines of output. Which is really not all that bad, but that is my local Ubuntu workstation, not a server. A standard Linux server would probably have five times that many lines. So be sure to use a good text parser (again, like textpad) to effectively wade through the data to find what you are looking for.

Some of the switches which I have found useful in narrowing your focus if you can get console access are (from the lsof man pages):

- To list all open Internet, x.25 (HP-UX) and UNIX domain files, use:

  ```
  lsof -i -U
  ```

- To list all open IPv4 network files in use by the process whose PID is 1234, use:

  ```
  lsof -i 4 -a -p 1234
  ```

- Presuming the UNIX dialect supports IPv6, to list only open IPv6 network files, use:

```
lsof -i 6
```

- To list all files using any protocol on ports 513, 514, or 515 of host wonderland.cc.purdue.edu, use:

```
lsof -i @wonderland.cc.purdue.edu:513-515
```

- To list all files using any protocol on any port of mace.cc.purdue.edu (cc.purdue.edu is the default domain), use:

```
lsof -i @mace
```

- To list all open files for login name "abe," or user ID 1234, or process 456, or process 123, or process 789, use:

```
lsof -p 456,123,789 -u 1234,abe
```

- To list all open files on device /dev/hd4, use:

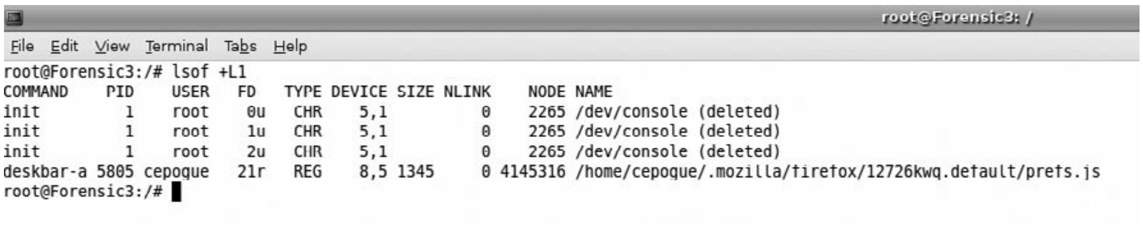
```
lsof /dev/hd4
```

- To find the process that has /u/abe/foo open, use:

```
lsof /u/abe/foo
```

Additionally, I like to use the "+L1" switch to display all of the unlinked (or marked for deletion) files. (See Figure 4.14).

Figure 4.14 +L1 Switch



```
root@Forensic3:/# lsof +L1
COMMAND    PID   USER   FD   TYPE DEVICE SIZE NLINK    NODE NAME
init         1   root   0u   CHR    5,1          0    2265 /dev/console (deleted)
init         1   root   1u   CHR    5,1          0    2265 /dev/console (deleted)
init         1   root   2u   CHR    5,1          0    2265 /dev/console (deleted)
deskbar-a 5805 cepogue  21r  REG    8,5 1345     0 4145316 /home/cepogue/.mozilla/firefox/12726kwq.default/prefs.js
root@Forensic3:/#
```

This command switch has proved itself useful more than once, when an illicit user has tried to cover their tracks by deleting something.

Summary

Gathering the volatile data is one thing, knowing what the heck it means is something else entirely. Hopefully, you now have a decent understanding of the commands that we discussed in Chapter 3, what the output looks like, and why it's important. Remember, each case will be different, so likewise will the information you gather. Be flexible (Semper Gumby!) and be smart.

Simply gathering the data, and even understanding it is only the beginning to an effective analysis. Do not forget that the information you gather needs to be correlated. Compare your volatile data with the shell histories, localhost logs, network logs, and anything else the customer can provide you with. Never look at a single piece of information as the "end" of a trail, but as a piece of a larger puzzle, you just need to figure out where it fits in.

The beauty of Linux is that there is always more than one way to do something and likely more than one thing that is keeping track of those things. If a tool or utility is unfamiliar to you, test it out in your lab. Find out what it does, how it does it, and what it looks like. Often you will find that the work you do in the lab can make or break your case.