The Law Enforcement Introduction to Linux

A Beginner's Guide

Barry J. Grundy
Special Agent
NASA Office of Inspector General
Computer Crimes Division
Code 190 Greenbelt Rd.
Greenbelt, MD 20771
(301)286-3358
bgrundy@imx.hq.nasa.gov

VER 1.8.1

Legalities

Foreword

This purpose of this document is to provide an introduction to the GNU/Linux operating system for Investigators. There are better books written on the subject of GNU/Linux (by better qualified professionals), but my hope here is to provide a single document that allows a user to sit at the shell prompt (command prompt) for the first time and not be overwhelmed by a 700 page book.

In addition, tools available to investigators for forensic analysis are presented. **This is by no means meant to be the definitive "how-to" on forensic methods using GNU/Linux**. Rather, it is a ***starting point*** for those who are interested in pursuing the self education needed to become proficient in the use of Linux as an investigative tool. Not all of the commands offered here will work in all situations, but by describing the ***basic*** commands available to the Investigator I hope to "start the ball rolling". I will present the commands, the reader needs to followup on the more advanced options and uses. Knowing *how* these commands work is every bit as important as knowing what to type at the prompt. If you are even an intermediate Linux user, then much of what is contained in these pages will be review. Still, I hope you find some of it useful.

As always, I am open to suggestions and critique. My contact information is on the front page. If you have ideas, questions, or comments, please don't hesitate to call or e-mail me. Any feedback is welcome.

This document is always being updated. Check for newer versions (numbered on the front page) at:

http://home.columbus.rr.com/bgrundy/linlaw

or in the "resources" section of the Ohio HTCIA website:

http://www.ohiohtcia.org/resource.html

A word about the "GNU" in GNU/Linux

When we talk about the Linux operating system, we are actually talking about the GNU/Linux operating system (OS). Linux itself is *not* an OS. It is just a kernel. The OS is actually a combination of the Linux kernel and the GNU utilities that provide the tools allowing us to interact with the kernel. Which is why the proper name for the OS is "GNU/Linux". We (incorrectly) call it "Linux" for convenience.

I. Installation

First and foremost, *know your hardware*. If your Linux machine is to be a dual boot system with Windows, then use the Windows Device Manager to record all your installed hardware and the settings used by Windows. If you are setting up a standalone Linux system, then gather as much documentation about your system as you can. This has become much less important with the evolution of the Linux install routines. Hardware compatibility and detection has been *greatly* improved over the past couple of years.

- Hard drive
- SCSI adapters and devices
- Sound card
- Video Card (important to know your chipset, memory).
- Monitor timings
  - Horizontal and vertical refresh rates.
- Network card settings
- Network Parameters
  - IP (if not DHCP)
  - Netmask
  - DNS servers
  - Default gateway
- Modem
  - NO WINMODEMS. (support is being worked on – check http://www.linmodems.org. Note that if you have an HSF modem, Conexant has released linux drivers! Find them at http://www.conexant.com/customer/).
- USB support in kernel 2.4. USB is becoming a standard feature of Linux with newer distributions that ship with the 2.4 series kernel.

For Redhat, you can check for hardware compatibility and installation issues at: http://www.redhat.com/support/hardware/

If you cannot find your monitor documentation and need it for Xfree86 (the Linux GUI) setup, then go to:
http://www.monitorworld.com/monitors_home.html

Distributions

1. *RedHat*

The most popular Linux distribution (right now, but losing ground). RedHat works with companies like Dell, IBM and Intel to assist business in the adoption of Linux for enterprise use. Use of RPM and Kickstart began the first "real" user upgrade paths for Linux.

2. *Debian*

Not for beginners. The installation routine is not as polished as some other distributions. A hacker favorite. Just adopted the 2.2 series kernel. The most “non-commercial” Linux distro and true to the spirit of GNU/GPL.

3. *SuSE*

Another distribution with it's own proprietary install program, YaST2. German in origin. SuSE is by far the largest distribution, and comes with six (6!) CD's (or a DVD). There are tons of included applications, most notably Vmware and Real Player.

5. *Slackware*

The original commercial distribution. Slackware has been around for years. Installation is not as easy as others. Good standard Linux. Not over-encumbered by GUI config tools.

6. Others (and many more):

TurboLinux

Mandrake Linux (rapidly gaining on RedHat’s market share).

My suggestion for the absolute beginner would be either the newest version of Mandrake (currently 8.2) or RedHat (currently 7.2). Mandrake is actually a RedHat based distribution with numerous GUI enhancements that make the learning process easier for “newbies”. I suggest RedHat based distributions because the online support is huge, and the installed base is the largest. This is only a suggestion. If you really want to “dive in” and bury yourself, go for Debian or Slackware.

One thing to keep in mind: If you are going to use GNU/Linux in a forensic capacity, then try not to rely on GUI tools too much. Almost all settings and configurations in GNU/Linux are maintained in text files (usually in either your home directory, or in */etc*). By learning to edit the files yourself, you avoid problems when either the X window system is not available, or when the specific GUI tool you rely on is not on the system.

Installation Methods:

- Buy a book! (most come with a distro)
- Download the needed files, create a boot and root disk and read online! (See the "Linux Support" section on page 50)
- Get hold of a distribution CD and boot from it (change your bios to boot from the CD if needed).

If you have access to a bootable installation CDROM (download an ISO image and burn it on a CDR, buy a book that includes a distro, etc.), then this process will be easier. Much of the work is done for you, and relativley safe defaults are provided. As mentioned earlier, hardware detection has gone through some great improvements in the last year or two.

Typical Linux installation is well documented online (check the "how-tos" at www.linuxdoc.org). There are numerous books available on the subject, and most of these are supplied with a Linux distribution ready for install.

Bootable ISO's can downloaded from www.linuxiso.org and burned to a CD. Familiarize yourself with Linux disk and partition naming conventions (covered in Chapter II of this document) and you should be ready to start.

Installation Overview

1. Decide on standalone Linux or dual boot.
   -install Windows first in a dual boot system.
   -do NOT create any extra partitions with Windows fdisk. Just leave the space unallocated. The Linux install will create the partitions.
2. Boot the Linux Media
   -hopefully you have a bootable CDROM (and booting from the CD is supported in your bios.)
   -you can use a boot.img to create a bootable floppy for the install from the installation CD if booting from the CD is not possible.
3. Accepting most defaults works.
   Your hardware will be detected and configured under most (if not all) circumstances. If the install freezes or breaks, try again

in “text” mode or “expert” mode, if available. This is often caused by video card problems.

4. Partition and format for Linux
   -at *least* two partitions.
   -Root (/) as type “Linux Native”.
   -Swap as type “Linux Swap” (use 2x your system memory as a starting point for swap size).
5. Package installation (system)
   -

II. Linux Disks, Partitions and the Filesystem

Disks

Linux treats its devices as files. The special directory where theses "files" are maintained is *"/dev*".

Floppy (a:)	/dev/fd0
Floppy (b:)	/dev/fd1

1st Hard disk	(master, IDE-0)	/dev/hda
2nd Hard disk	(slave, IDE-0)	/dev/hdb
3nd Hard disk	(master, IDE-1)	/dev/hdc etc.
1st SCSI hard disk		/dev/sda
2nd SCSI hard disk		/dev/sdb etc.

Partitions

1st Hard disk (master, IDE-0)	/dev/hda
• 1st Primary partition	/dev/hda1
• 2nd Primary partition	/dev/hda2 etc.
• 1st Logical drive	/dev/hda5
• 2nd Logical drive	/dev/hda6 etc.
2nd Hard disk (slave, IDE-0)	/dev/hdb
• 1st Primary partition	/dev/hdb1 etc.
CDROM (master, IDE-1)	/dev/hdc
CDROM (SCSI)	/dev/scd0
1st SCSI disk	/dev/sda etc.

The pattern described above is fairly easy to follow. If you are using a standard IDE disk, it will be refered to as "hdx" where the "x" is replaced with an "a" if the disk is connected to the primary IDE controller as master and a "b" if the disk is connected to the primary IDE controller as a slave device. In the same way, the IDE disks connected to the secondary IDE controller as master and slave will be referred to as "hdc" and "hdd" respectivley.

This is an example of the output of **fdisk -l /dev/hda** on a dual boot system:

Disk /dev/hda: 255 heads, 63 sectors, 1582 cylinders
*Units = cylinders of 16065 * 512 bytes*

Device	*Boot*	*Start*	*End*	*Blocks*	*Id*	*System*
/dev/hda1		*1*	*255*	*2048256*	*b*	*Win95 FAT32*
/dev/hda2	*	*256*	*638*	*3076447+*	*83*	*Linux*
/dev/hda3		*639*	*649*	*88357+*	*82*	*Linux swap*
/dev/hda4		*650*	*1582*	*7494322+*	*f*	*Win95 Ext'd (LBA)*
/dev/hda5		*650*	*1453*	*6458098+*	*b*	*Win95 FAT32*
/dev/hda6		*1454*	*1582*	*1036161*	*b*	*Win95 FAT*

fdisk –l /dev/hd*x* gives you a list of all the partitons available on a particular drive. Each partition is identifed by its Linux name. The "boot flag" is indicated, and the beginning and ending cylinders for each partition is given. The number of blocks per partiton is displayed. Finally, the partition "Id" and file system type are displayed. To see a list of valid types, run **fdisk** and at the prompt type "l" (the letter “el”). Do not confuse Linux fdisk with DOS fdisk. They are very different. The Linux version of **fdisk** provides for much greater control over partitioning.

Note that if you use a parallel ZIP drive, it will be accessed as */dev/sda* (assuming no other SCSI devices). Support must be compiled into the kernel (most new distros have it already compiled in). A Linux formatted ZIP disk is */dev/sda1* and a DOS (FAT) formatted ZIP disk is */dev/sda4* (no, I don't know why).

BEFORE THESE DEVICES CAN BE USED, THEY MUST BE MOUNTED! Any partitions you define during installation will be mounted automatically every time you boot. We will cover the mounting of devices in the section that deals with Linux commands, after you have some navigation experience.

The Filesystem

Like the Windows file system, the Linux file system is hierarchical. the "top" directory is refered to as "the root" directory and is represented by "/". Note that the following is not a complete list, but provides a introduction to some important directories.

/ (“root” not to be confused with “/root”)
- |_ bin
  - |_ <files> ls, chmod, sort, date, cp, dd
- |_boot
  - |_<files> vmlinuz, system.map
- |_ dev
  - |_<devices> hd*, tty*, sd*, fd*, cdrom
- |_ etc
  - |_X11
    - |_ <files> XF86Config, X
  - |_<files> lilo.conf, fstab, inittab, conf.modules
- |_ home
  - |_barry (your user’s name is in here)
    - |_<files> .bashrc, .bash_profile, personal files
  - |_other users
- |_mnt
  - |_cdrom
  - |_floppy
- |_root
  - |_<root user's home directory>
- |_sbin
  - |_<files> shutdown, cfdisk, fdisk, insmod
- |_usr
  - |_local
  - |_lib
  - |_man

On most Linux distributions, the directory structure is organized in the same manner. Certain configuration files and programs are distribution dependant, but the basic layout is similar to this.

Directory contents can include:

/bin

-Common commands.

/boot

-Files needed at boot time, including the kernel images pointed to by LILO (the LInux LOader). Also includes information needed to load modules during bootup.

/dev

-Files that represent devices on the system. These are actually interface files to allow the kernel to interact with the hardware and the filesystem.

/etc

-Administrative configuration files and scripts.

/home

-Directories for each user on the system. Each user directory can be extended by the respective user and will contain their personal files.

/mnt

-Provides mount points for external, remote and removeable file systems and devices.

/root

-The root user's home directory.

/sbin

-Administrative commands and process control daemons.

/usr

-Contains local software, libraries, documentation, games, etc.

III. The Linux Boot Sequence (*Simplified*)

Booting the kernel

The first step in the boot up sequence for Linux is loading the kernel. The kernel image is usually contained in the */boot* directory. It can go by several different names…

- bzImage
- vmlinuz

Sometimes the kernel image will specify the kernel version contained in the image, i.e. *bzImage-2.2.12*. Very often there is a softlink (like a shortcut) to the most current kernel image in the */boot* directory. It is normally this softlink that is referenced by the LILO. LILO specifies the "root device", along with the kernel version to be booted. This is all controlled by the file */etc/lilo.conf*

more /etc/lilo.conf

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
image=/boot/vmlinuz
        label=linux
        root=/dev/hda3
        read-only
image=/boot/vmlinuz-2.2.9-19mdk
        label=linux_old
        root=/dev/hda3
        read-only
other=/dev/hda1
        label=dos
        table=/dev/hda
```

You can see the kernel messages that "fly" past the screen during bootup with the command **dmesg.** Often, this command can be used to find hardware problems that occur during bootup, or to see how your suspect drive was detected (geometry, etc). The output can be piped through a paging viewer to make it easier to see:

dmesg | less

Initialization

The next step starts with the program */sbin/init.* This program really has two functions:

- initialize the runlevel and startup scripts
- terminal process control (respawn terminals)

In short, the init program is controlled by the file */etc/inittab*. It is this file that controls your runlevel and the global startup scripts for the system.

Runlevel

The runlevel is simply a description of the system state. For our purposes, it is easiest to say that (for *RedHat*, at least – other systems, like *Suse* may differ):

- runlevel 0 = shutdown
- runlevel 1 = single user mode
- runlevel 3 = full multiuser mode / text login
- runlevel 5 = full multiuser / X11 / graphical login
- runlevel 6 = reboot

In the file */etc/inittab* you will see a line similar to:

id:3:initdefault:

It is here that the default runlevel for the system is set. If you want a text login (which I would strongly suggest), set the above value to "*3*". If you want a graphical login, you would edit the above line to contain a "5".

Global Startup Scripts

After the default runlevel has been set, *init* (via */etc/inittab*) then runs the following scripts:

- */etc/rc.d/rc.sysinit* - handles system initialization, filesystem mount and check, PNP devices, etc.
- */etc/rc.d/rc X* - where *X* is the runlevel passed as an argument by *init*. This script then calls the specified script for the runlevel that is being started.
- */etc/rc.d/rc.local* - called from within the specific runlevel scripts, *rc.local* is a general purpose script that can be edited to include commands that you want started at bootup (sort of like *autoexec.bat*).

Again, this is somewhat Redhat / Mandrake specific. Other distributions can differ slightly.

Bash

Bash *(Bourne Again Shell*) is the default command shell for RedHat, Mandrake, and many other Linux distros. It is the program that sets the environment for your command line experience in Linux. The functional equivalent in DOS would be *command.com*. There are a number of shells available, but we will cover bash here.

There are actually quite a few files that can be used to customize a user's Linux experience. We will concentrate on two that will get you started. I am assuming here that you are using the bash shell.

- */home/$USER/.bashrc* - This script is located in each user's home directory (*$USER*) and can be edited by the user, allowing him or her to customize their own environment. It is in this file that you can add aliases to change the way commands respond.
- */etc/bashrc* - This is the global bash initialization file. Edits made to this file will be applied to all users using the bash shell.

The Bash startup sequence is actually more complicated than this, but this should give you a starting point. In addition to the above files, check out *.bash_profile* and */etc/profile.*

IV. DOS / Linux Equivalent Commands

"DOS command" = Linux equivalent

"dir" = **ls**

ls –F	classifies files and directories.
ls –a	show all files (including hidden).
ls –l	detailed file list (long view).
ls –lh	detailed list (long,with "human readable" file sizes).

Output of **ls -l**

```
total 11
drwx------   5 barry    501      1024 Feb  7 15:07 Desktop
drwxr-xr-x   2 barry    users    1024 Feb  9 08:19 Files
drwx------   2 barry    users    1024 Dec 18 15:58 Mail
-rw-r--r--   1 barry    users       0 Feb  9 09:21 Mydata.file.today
drwxr-xr-x  26 barry    users    1024 Jan 23 00:49 Office51
drwxr-xr-x   2 barry    users    1024 Nov  8 16:21 Prog
drwxr-xr-x   2 barry    users    1024 Dec 16 15:42 Programming
drwxr-xr-x   2 barry    users    1024 Feb  9 09:27 Sample
drwxr-xr-x   2 barry    users    1024 Feb  9 08:41 autosave
drwxr-xr-x   2 barry    users    1024 Feb  8 15:50 bin
drwx------   2 barry    501      1024 Oct  9 14:00 nsmail
drwxrwxr-x   3 barry    501      1024 Jan 23 00:42 software
-rw-r--r--   1 barry    users       0 Feb  9 09:20 textfile
```

"cd" = **cd**

cd	back to your home directory.
cd ..	up one directory (note the space between "cd" and "..").
cd -	back to the last directory you were in.
cd /*dirname*	change to the specified directory. Note that the addition of the "/" in front of the directory implies an explicit path, not a relative one. With practice, this will make more sense.

"copy" = **cp**

cp *sourcefile destinationfile* copy a file.

"cls" = **clear**

clears the terminal screen of all text and returns a prompt.

"move" and "ren" = **mv**

mv *sourcefile destination* move or rename a file.

"del" = **rm** (be careful, undelete is not so easy!)

rm *filename* deletes a file.

rm -r recursively deletes all files in directories and subdirectories.

"help" or " /?" = **man**

man *command* displays a "manual" page for the specified command. Use "q" to quit. VERY USEFUL.

Output of **man find:**

FIND(1L) *FIND(1L)*

NAME

find - search for files in a directory hierarchy

SYNOPSIS

find [path...] [expression]

DESCRIPTION

*This manual page documents the GNU version of **find**. **find** searches the directory tree rooted at each given file name by evaluating the given expression from left to right, according to the rules of precedence (see section OPERATORS), until the outcome is known (the left hand side is false for and operations, true for or), at which point **find** moves on to the next file name.*

<*CONTINUES*>

"md" = **mkdir**

mkdir *directoryname* creates a directory.

"type" = **cat** or **more** or **less**

cat *filename* The simplest form of file display, **cat** streams the contents of a file to the standard output. **cat** actually stands for "*concatenate*". This command can also be used to add files together (useful later on…). For example:

cat *file1 file2 > file3*

Takes the contents of *file1* and *file2* and streams the output which is redirected to a single file, *file3*. This effectively adds the two files into one single file (the original files remain unchanged).

more *filename* displays the contents of a file one page at a time. Unlike its DOS counterpart, Linux **more** takes filenames as direct arguments.

less *filename* **less** is a better **more**. Supports scrolling in both directions, and the ability to search. **less** is actually the GNU version of **more**, and on many systems you will find that **more** is actually a link to **less**.

Note that you can string together several options. For example:

ls -aF

will give you a list of files, including hidden files and file/directory classification ("/" for directories, "*" for executables, and "@" for links).

Output of **ls –aF :**

./	*.emacs*	*.kderc*	*.zshrc*	*Sample/*
../	*.gnome/*	*.kpackage/*	*Desktop/*	*autosave/*
.Xauthority	*.gnome_private/*	*.mailcap*	*Files/*	*bin/*
.Xdefaults	*.gqviewrc*	*.maxwellrc*	*Mail/*	*mylink@*
.bash_history	*.gxedit*	*.netscape/*	*Mydata.file.today*	*nsmail/*
.bash_logout	*.gxedit.apps*	*.sversionrc*	*Office51/*	*samp_script.sh**
.bash_profile	*.kaudioserver*	*.user.rdb*	*Prog/*	*snapshot01.gif*
.bashrc	*.kde/*	*.vimrc*	*Programming/*	*software/*
				textfile

Additional useful commands

grep - search for patterns in a file (or in the output of another command).

g**rep *pattern filename***

will look for occurances of *pattern* within the file *filename.* **grep** is an extremely powerful tool. It has hundreds of uses given the large number of options it supports. Check the **man** page for more details.

find -allows you to search for a file (wild cards – actually "regular expressions" permitted). To look for your *XF86Config* file, you might try:

find / -name XF86Config -print

This means "find, starting in the root directory (/), by name, *XF86Config* and print the results to the screen".

pwd -prints the present working directory to the screen.

pwd
/home/barry

file -categorizes files based on what they contain, regardless of the name (or extension, if one exists). Compares the file header to the "magic" file in an attempt to ID the file type. For example:

file snapshot01.gif
snapshot01.gif: GIF image data, version 87a, 800 x 600

ps -list of current processes. Gives the process ID number (PID), and the terminal on which the process is running.

ps -ax shows all processes (**-a**), and all processes without an associated terminal (**-x**).

Output (partial) of **ps -ax** on my system as it is running right now:

PID	*TTY*	*STAT*	*TIME*	*COMMAND*
1	*?*	*S*	*0:04*	*init*
2	*?*	*SW*	*0:00*	*[kflushd]*
3	*?*	*SW*	*0:00*	*[kupdate]*
4	*?*	*SW*	*0:00*	*[kpiod]*
5	*?*	*SW*	*0:00*	*[kswapd]*
191	*?*	*S*	*0:01*	*/sbin/pump -i eth0*
232	*?*	*S*	*0:00*	*syslogd -m 0*
243	*?*	*S*	*0:00*	*klogd*
328	*tty1*	*SW*	*0:00*	*[login]*
329	*tty2*	*S*	*0:00*	*login -- root*
330	*tty3*	*S*	*0:00*	*/sbin/mingetty tty3*
331	*tty4*	*S*	*0:00*	*/sbin/mingetty tty4*
332	*tty5*	*S*	*0:00*	*/sbin/mingetty tty5*
333	*tty6*	*S*	*0:00*	*/sbin/mingetty tty6*
340	*tty1*	*SW*	*0:00*	*[bash]*
353	*tty1*	*S*	*0:00*	*sh /usr/X11R6/bin/startx*
360	*tty1*	*S*	*0:00*	*xinit /etc/X11/xinit/xinitrc -- :0 -auth /home/barry/*
361	*?*	*R*	*2:04*	*/etc/X11/X :0 -auth /home/barry/.Xauthority*
365	*tty1*	*S*	*0:05*	*kwm*
368	*tty1*	*S*	*0:00*	*kbgndwm*

jobs -gives a list of the "jobs" that are currently running. A good indicator of the command line muli-tasking of Linux. Each job is assigned a "job number" which can then be used to "kill" the jobs or run them in the foreground or background.

strings -prints out the readable characters from a file. Will print out strings from a file that are at least four characters long (by default). Useful for looking at data files without the originating program, and searching executables for useful strings, etc.

chmod -changes the permissions on a file.

First, a short description of permissions:
Files in Linux have certain specified file permissions. These permissions can be viewed by running the ls -l command on a directory or on a particular file. For example:

ls –l q2.script

-rwxr-xr-x 1 barry user 1643 Jan 19 23:23 q2.script

If you look close at the first 10 characters, you have a dash (-) followed by 9 more characters. The first character describes the type of file. A dash (-) indicates a regular file. A "d" would indicate a directory, and "b" a special block device, etc.

The next 9 characters indicate the file permissions. These are given in groups of three

User	Group	Others
rwx	rwx	rwx

The characters indicate

r = read
w = write
x = execute

So in the above *q2.script* we have

-rwxr-xr-x

This give the user (file owner) read, write and execute permissions, but restricts other members of the users group and users outside that group to only read and execute the file.

Now back to the chmod command. There are a number of ways to use this command, including explicitly assigning r, w, or x to the file. We will cover the octal method here because the syntax is easiest to remember. In this method, the syntax is as follows

chmod *octal filename*

octal is a three digit numerical value in which the first digitrepresents the user, the second digit represents the group, and the third digit represents others outside the user's group. Each digit is calculated by assigning a value to each permission:

read (r) = 4
write (w) = 2
execute (x) = 1

For example, the file *q2.script* in our original example has an octal permission value of 755. If you wanted to change the file so that the owner and the group had read, write and execute permissions, but others would only be allowed to read the file, you would issue the command:

chmod 774 q2.script

A new long list of the file would show:

-rwxrwxr-- 1 barry user 1643 Jan 19 23:23 q2.script

chown -changes the owner of a file in much the same way as chmod changes the permissions.

chown ralph q2.script

-rwxrwxr-- 1 ralph user 1643 Jan 19 23:23 q2.script

chgrp - changes a file's group attribute. Works the same as chown, but affects the group instead of the owner.

shutdown -this command MUST be used to shutdown the machine and cleanly exit the system. This is not DOS. Turning off the machine at the prompt is not allowed and can damage your filesystem. You can run several different options here (check the man page for many more):

shutdown -r now	will reboot the system now.
shutdown -h now	will halt the system. Ready for power down.

Metacharacters

Linux also supports wildcards (metacharacters)

- * for multiple characters (including ".").
- ? for single characters.
- [ ] for groups of characters or a range of characters or numbers.

This is a complicated and *very* powerful subject, and *will* require further reading… Refer to "regular expressions" in your favorite Linux text, along with "globbing" or "shell expansion".

Command Hints

1. Linux supports command line editing.
2. Linux has a History List of previously used commands.
   -use the keyboard arrows to scroll through commands you've already typed.
3. Linux commands and filenames are CASE SENSITIVE.
4. Learn output redirection for stdout and stderr.
5. Linux uses "/" for directories, DOS uses "\".
6. Linux uses "-" for command options, DOS uses "/"
7. To execute commands in the current directory (if the current directory is not in your PATH), use the syntax "*./command*". This tells Linux to look in the present directory for the command.

Pipes and Redirection

Like DOS, Linux allows you to redirect the output of a command from the standard output (usually the display or "console") to another device or file. This is useful for creating files that contain lists of files on a mounted volume, or in a directory. For example:

ls -al > filelist.txt

would output a long list of all the files in the current directory. Instead of outputting the list to the console, a new file called "*filelist.txt*" will be created that will contain the list. If the file "*filelist.txt*" already existed, then it will be overwritten. Use the following command to append the output of the command to the existing file, instead of over-writing it:

ls -al >> filelist.txt

Another useful tool similar to that available on DOS is the command pipe. The command pipe takes the output of one command and "pipes" it straight to the input of another command. This is an extremely powerful tool for the command line.

Look at the following process list:

ps -ax

```
 PID TTY      STAT   TIME COMMAND
   1 ?        S      0:04 init
 232 ?        S      0:00 syslogd -m 0
 271 ?        S      0:00 inetd
 328 tty1     SW     0:00 [login]
 329 tty2     S      0:00 login -- root
 330 tty3     SW     0:00 [mingetty]
 331 tty4     SW     0:00 [mingetty]
 340 tty1     SW     0:00 [bash]
 353 tty1     SW     0:00 [startx]
 360 tty1     SW     0:00 [xinit]
 361 ?        R      2:41 /etc/X11/X :0 -auth /home/barry/.Xauthority
 519 pts/0    S      0:00 bash
 2451 pts/0   S      0:00 -bash
2490 tty2     S      0:00 -bash
```

2727 pts/1 R 0:00 ps -ax

What if all you wanted to see were those processess ID's that indicated a bash shell? You could "pipe" the output of **ps** to the input of **grep**, specifying "bash" as the pattern for grep to search. The result would give you only those lines of the output from ps that contained references to "bash".

ps -ax | grep bash

340 tty1 SW 0:00 [bash]
519 pts/0 S 0:00 bash
2451 pts/0 S 0:00 -bash
2490 tty2 S 0:00 -bash

The SuperUser

If Linux gives you an error message "Permission denied", then in all likelihood you need to be "root" to execute the command or edit the file, etc. You don't have to log out and then log back in as "root" to do this. Just use the "su" command to give yourself root powers (assuming you know root's password).

su -

Then enter the password when prompted. You now have root privileges (the system prompt will reflect this). Note that the "**-**" after "**su**" allows Linux to apply root's path to your "**su**" login. So you don't have to enter the full path of a command.

When you are finished using your "**su**" login, return to your own self by typing "**exit**".

A word of caution. Be VERY judicious in your use of the root login. It can be destructive. For simple tasks that require root permission, use "**su**" and use it sparingly.

V. Editing with Vi

There are a number of terminal mode editors available in Linux, including *emacs* and *vi*. You could always use one of the available GUI text editors in Xwindow, but what if you are unable to start X? The benefit of learning *vi* or *emacs* is your ability to use them from an xterm, a character terminal, or a **telnet** (use **ssh** instead!) session, etc. We will discuss *vi* here. (I don't do *emacs* :-)). *vi* in particular is useful, because you will find it on all versions of Unix. Learn *vi* and you should be able to edit a file on any Unix system.

Using Vi

You can start *vi* either by simply typing **vi** at the command prompt, or you can specify the file you want to edit with **vi *filename***. If the file does not already exist, it will be created for you.

vi consists of two operating modes, *command* mode and *insert* mode. When you first enter *vi* you will be in command mode. Command mode

Vi command summary

Insert Mode:

a	=	append text (after the cursor)
i	=	insert text (directly under the cursor)
o (the letter “oh”)	=	open a new line under the current line
O (capital “oh”)	=	open a new line above the current line

Command Mode:

0 (zero)	=	Move cursor to beginning of current line.
$	=	Move cursor to the end of current line.
x	=	delete the character under the cursor
X	=	delete the character before the cursor
dd	=	delete the entire line the cursor is on
:w	=	save and continue editing
:wq	=	save and quit (can use ZZ as well)
:q!	=	quit and discard changes
:w *filename*	=	save a copy to *filename* (save as)

The best way to save yourself from a messed up edit is to hit **<ESC>** followed by **:q!**

Another useful feature that can be used in command mode is the string search. To search for a particular string in a file, make sure you are in command mode and type

/string

Where *string* is your search target. After issuing the command, you can move on to the next hit by typing "**n**".

vi is an extremely powerful editor. There are a huge number of commands and capabilities that are outside the scope of this guide. See **man vi** for more details. Keep in mind there are chapters in books devoted to this editor. There are even a couple of books devoted to *vi* alone.

VI. Mounting File Systems and Disks

There is a long list of file systems that can be accessed through Linux. You do this by using the **mount** command. Linux has a special directory used to **mount** file systems to the existing Linux directory tree. This directory is called */mnt.* It is here that you can dynamically attach new filesystems from external (or internal) storage devices that were not mounted at boot time. Actually you can **mount** files anywhere (not just on */mnt*), but it's better for organization. Here is a brief overview.

Any time you specify a mount point in */mnt*, you must first make sure that you have a directory under */mnt* to use. After all, suppose we want to have a CDROM and a floppy mounted at the same time? They can't both be mounted under */mnt* (you would be trying to access 2 filesystems through one directory!). So we create directories for each device under the parent directory */mnt*. You decide what you want to call the directories, but make them easy to remember. Keep in mind that until you learn to manipulate the file */etc/fstab* (covered later), only root can mount and unmount disks and filesystems.

mkdir /mnt/floppy
mkdir /mnt/cdrom

Newer distributions usually create these mountpoints for you, but you might want to add others for yourself (mountpoints for subject disks or copies, etc.)

The Mount Command

The "**mount**" command uses the following syntax:

mount -t <filesystem> -o <options> <device> <mountpoint>

Example: Reading a DOS / Windows floppy

- Insert the floppy and type:

mount -t vfat /dev/fd0 /mnt/floppy

- Now change to the newly mounted filesystem:

cd /mnt/floppy

- You should now be able to navigate the floppy as usual.
- When you are finished, EXIT OUT OF THE */mnt/floppy* directory, and unmount the file system with:

umount /mnt/floppy

- Note the proper command is ***u*mount**, not ***un*mount**. This cleanly unmounts the disk. DO NOT remove the disk OR SWAP the disk until it is unmounted.
- If you get an error message that says the filesystem cannot be unmounted because it is busy, then you most likely have a file open from that directory, or are using that directory from another terminal. Check all you xterms and virtual terminals.

Example: Reading a CDROM

- Insert the CDROM and type:

mount -t iso9660 /dev/cdrom /mnt/cdrom

- Now change to the newly mounted filesystem:

cd /mnt/cdrom

- You should now be able to navigate the CD as usual.
- When you are finished, EXIT OUT OF THE */mnt/cdrom* directory, and unmount the file system with:

umount /mnt/cdrom

If you want to see a list of which filesystems are currently mounted, just use the **mount** command without any arguments or parameters. It will list the mount point and filesystem type of each device on system, along with

the mount options used (if any). This is actually read from a file called */proc/mounts*, part of a virtual filesystem that keeps an up to date "snapshot" of the current system configuration. Try the following two commands:

mount
more /proc/mounts

The ability to mount and unmount filesystems is an important skill in Linux. There are a large number of options that can be used with **mount** (some we will cover later), and a number of ways the mounting can be done easily and automatically. Refer to the **mount** info or man pages for more information.

The file system table (/etc/fstab)

It might seem like "**mount -t iso9660 /dev/cdrom /mnt/cdrom**" is alot to type every time you want to mount a CD or a disk. One way around this is to edit the file */etc/fstab*. This file allows you to provide defaults for your mountable devices, thereby shortenting the commands required to mount them. My */etc/fstab* looks like this:

```
/dev/hda2   /           ext2      defaults              1 1
/dev/hda5   /mnt/apps   vfat      user,noauto,defaults      0 0
/dev/hda6   /mnt/data   vfat      user,noauto,defaults      0 0
/dev/hda3   swap        swap      defaults              0 0
/dev/fd0    /mnt/floppy vfat      user,noauto           0 0
/dev/hdc    /mnt/cdrom  iso9660   user,noauto,ro            0 0
/dev/sda4   /mnt/zip    vfat      user,noauto,defaults      0 0
none        /proc       proc      defaults              0 0
```

The columns are:
<device> <mount point> <filesystem> <default options>

With this */etc/fstab*, I can mount a floppy or CD by simply typing:

mount /mnt/floppy
mount /mnt/cdrom

The above **mount** commands look incomplete. When not enough

information is given, the **mount** command will look to */etc/fstab* to fill in the blanks. If it finds the required info, it will go ahead with the mount.

Note the "user" entry in the options column for some devices. This allows non-root users to mount the devices. Very useful. To find out more about available options for */etc/fstab*, enter **info fstab** at the command prompt.

Also keep in mind that default Linux installations will often create */mnt/floppy* and */mnt/cdrom* for you already. After installing a new Linux system, have a look at */etc/fstab* to see what is available for you. If what you need isn't there, add it.

VII. Linux and Forensics

Linux comes with a number of simple utilities that make imaging and basic analysis of suspect disks and drives comparitively easy. These tools include:

- **dd or cp** -command used to copy from an input file or device to an output file or device. Simple bitstream imaging.
- **cfdisk and fdisk** -used to determine the disk structure.
- **grep** -search files (or multiple files) for instances of an expression or pattern.
- **chmod** -change the default permissions on a file. Used to turn off the write permissions of an image file.
- **The loop device** -allows you to mount an image without having to rewrite the image to a disk.
- **sha1sum** -create and store an SHA hash of a file or list of files (including devices).
- **file** -reads a file's header information in an attempt to ascertain it's type, regardless of name or extension.
- **xxd** - command line hexdump tool. For veiwing a file in hex mode.
- **ghex and khexedit** -the Gnome and KDE (X Window interfaces) hex editors. Both have primitive search and byte selection capabilities.

Following is a *very* simple series of steps to allow you to perform an easy practice analysis using the simple Linux tools mentioned above. All of the commands can be further explored through the output of "**man *command***". For simplicity we are going to use a floppy from a DOS machine.

Analysis organization

Most of the work you will do here can be applied to actual casework. The tools are standard Linux tools, and although the example shown here is *very* simple, it can be extended with some practice and a little (ok, a lot) of reading.

The output of various commands and the amount of searching we will do here is limited by the scope of this example and the amount of data on a floppy. When you actually do an analysis on larger media, you will want to have it organized. Note that when you issue a command that results in an output file, that file will end up in your current directory, unless you specify a path for it in the command.

One way of organizing your data would be to create a directory in your "home" directory for evidence and then a subdirectory for different cases.

mkdir ~/evidence

The tilde (~) in front of the directory name is shorthand for "home directory", so when I type **~/evidence**, Linux interprets it as **/home/barry/evidence**. If I am logged in as root, then the directory will be created as **/root/evidence**. Note that if you are already in your home directory, then you don't need to type **~/**. Simply using **mkdir evidence** will work just fine. We are being explicit for instructional purposes.

Directing all of our analysis output to this directory will keep our output files separated from everything else and maintain case organization.

For the purposes of this exercise, we will be logged in as "root". I have mentioned already that this is generally a bad idea, and that you can make a mess of your system if you are not careful. Many of the commands we are utilizing here require root acess (permissions should not be changed to allow otherwise, IMHO). So the output files that we create and the images we make will be found under **/root/evidence/**

An additonal step you might want to take is to create a special mount point for all disk analysis. This is another way of separating common system use with evidence processing. Note that since we will be directly accessing hardware and the root filesystem, we will have to **su** to *root* to perform this and the next steps of this analysis.

mkdir /mnt/analysis

Determining the structure of the disk

There are two simple tools available for determining the structure of a disk attached to your system. The first, **fdisk**, we discussed eariler using the **-l** option. Replace the "x" with the letter of the drive that corresponds to the subject drive.

fdisk –l /dev/hdx

Disk /dev/hda: 255 heads, 63 sectors, 1582 cylinders
*Units = cylinders of 16065 * 512 bytes*

Device	*Boot*	*Start*	*End*	*Blocks*	*Id*	*System*
/dev/hda1		*1*	*255*	*2048256*	*b*	*Win95 FAT32*
/dev/hda2	*	*256*	*638*	*3076447+*	*83*	*Linux*
/dev/hda3		*639*	*649*	*88357+*	*82*	*Linux swap*
/dev/hda4		*650*	*1582*	*7494322+*	*f*	*Win95 Ext'd (LBA)*
/dev/hda5		*650*	*1453*	*6458098+*	*b*	*Win95 FAT32*
/dev/hda6		*1454*	*1582*	*1036161*	*b*	*Win95 FAT*

We can redirect the output of this command to a file for later use by issuing the command as:

fdisk –l /dev/hdx > fdisk.disk1

A couple of things to note here: The name of the output file (*fdisk.disk1*) is completely arbitrary. There are no rules for extensions. Name the file anything you want. I would suggest you stick to a convention and make it descriptive. Also note that since we did not define an explicit path for the file name, *fdisk.disk1* will be created in our current directory (for instance, */root/evidence/*).

The second tool you might consider is a "graphical" tool called **cfdisk.** This tool serves the same function as fdisk, but is easier to navigate and is menu driven, allowing for eaiser partition manipulation (when required).

cfdisk /dev/hdx

Keep in mind that although **cfdisk** is easier to use, it's output is not formatted for redirection to a text file.

Note that you can expect to see strange output if you use this command on a floppy disk. Be aware of that if you attempt **cfdisk** or **fdisk** on the practice floppy. Try it on your harddrive instead.

Creating an image of the suspect disk

Make an image of the practice disk. This is your standard backup of a suspect disk.

dd if=/dev/fd0 of=image.disk1 bs=512

This takes your floppy device *(/dev/fd0*) as the input file (*if*) and writes the output file (*of*) called *image.disk1* in the current directory. The **bs** option specifies the blocksize. This is really not needed for most block devices (hard drives, etc.) as the actual block size is handled by the Linux kernel.

For the sake of saftey, change the read-write permissions of your image to read-only.

chmod 444 image.disk1

The **444** gives all users read-only access. If you are real picky, you could use **400**. Note that the owner of the file is the user that created it.

Now that you have created an image file, you can restore the image to another disk for analysis and viewing. Put another (blank) floppy in and type:

dd if=image.disk1 of=/dev/fd0 bs=512

This is the same as the first **dd** command, only in reverse. Now you are taking your image and writing it to another disk to be used as a backup or as a working copy for the actual analysis.

Mounting a restored image

Mount the restored disk and view the contents. Remember, we are assuming this is a DOS formatted disk from a Win 98/95 machine.

mount -t vfat -o ro,noexec /dev/fd0 /mnt/analysis

This will mount your restored disk on "*/mnt/analysis*". The "**–o ro,noexec**" specifies the options **ro** (read-only) and **noexec** (prevents the execution of binaries from the mount point) in order to protect the disk from you, and your system (and mountpoint) from the contents of the disk.

Now **cd** to the mount point (*/mnt/analysis*)and browse the contents.

Mounting the image using the loopback device

Another way to view the contents of the image without having to use another disk is to mount using the *loop* interface. The *loop* device driver must first be installed by "inserting" it's module (Linux "modular" driver) into the running kernel. The various modules available on your system are located in */lib/modules/$KERNEL-VERSION/*. They are object files that contain the required driver code for the supported device or option (filesystem support under Linux is often loaded as a module). Note that the current kernel version can be found using the command **uname -r**.

Using modules

Modules are installed and removed from the system "on the fly" using the following commands (as root):

insmod -to insert the module
rmmod -to remove the module
lsmod -to get a list of currently installed modules

We install the *loop* module (*loop.o*) with:

insmod /lib/modules/2.2.12-20/block/loop.o
^ *your kernel version goes here*

If you are using a newer 2.4 kernel based distribution (most probably are, use **uname –r** to determine your kernel version), then the module directories are structured a little different and the command would be:

insmod /lib/modules/2.2.12-20/kernel/drivers/block/loop.o
^ *your kernel version goes here*

Now check to see if the module has been correctly loaded with:

lsmod

The output should be:

Module	*Size*	*Used by*
loop	*7744*	*0 (unused)*

note that the output will include any other drivers loaded as modules

by the system as well (some at boot up).

Modules on Newer systems

On newer Linux systems (like the one you are probably using now) there an automatic *kernel daemon* that handles the loading and unloading of modules automatically. When you issue a command that requires a module that is not yet loaded, the kernel will detect your request and load the applicable module. The module "autoloader" is useful and ends the need to install modules by hand using insmod. It is likely that your system follows this convention.

We are now ready to mount the image using the loopback device. We use the same mount command and the same options, but this time we include the option "**loop**" to indicate that we want to use the *loop* device to mount the image. Change to the directory where you created the image and type:

mount -t vfat -o ro,noexec,loop image.disk1 /mnt/analysis

Now you can change to */mnt/analysis* and browse the image as if it were a mounted disk! Keep in mind that this sort of analysis is much harder to initiate as any user other than root. The insertion and deletion of the *loop* module (if required), as well as the mounting of the image using the *loop* device must be done as root (the mounting can be done by a user with the proper */etc/fstab* editing, but it's not versitile). Use the **mount** command to double check the mounted options.

File Hash

One important step in any analysis is verifying the integrity of your data both before after the analysis is complete. You can get a hash (CRC, MD5, or SHA) of each file in a number of different ways. We will use the SHA hash. SHA is a hash signature generator that supplies a 160 bit "fingerprint" of a file or disk. It is not feasible for someone to computationally recreate a file based on the SHA hash. This means that matching SHA signatures mean identical files.

We can get an SHA sum of a disk by changing to our evidence directory (i.e. */root/evidence*) and doing:

sha1sum /dev/fd0 > SHA.disk1

The redirection allows us to store the signature in a file and use it for

verification later on. In addition we can get a hash of each file on the disk using the **find** command and an option that allows us to execute a command on each file found. We can get a very useful list of SHA hashes for every file on a disk (once it is mounted) by changing to the */mnt/analysis* directory:

cd /mnt/analysis

and issuing the command:

find . -type f -exec sha1sum {} \; > /root/evidence/SHA.filelist

This command says "**find**, starting in the *current* directory (signified by the "**.**"), any regular file (**-type f**) and execute (**-exec**) the command **sha1sum** on all files found (**{}**). Redirect the output to *SHA.filelist* in the /root/evidence directory (where we are storing all of our evidence files). The "**\;**" is an escape sequence that ends the command.

You can also use Linux to do your verification for you. To verify that nothing has been changed on the original floppy, you can use the -c option with **sha1sum**. If the disk was not altered, the command will return "ok". Make sure the floppy is in the drive and type:

sha1sum -c /root/evidence/SHA.disk1

If the SHA hashes match from the floppy and the original SHA output file, then the command will return "OK" for */dev/fd0*. The same can be done with the list of file SHAs. Mount the floppy on */mnt/analysis,* change to that directory and issue the command:

sha1sum -c /root/evidence/SHA.filelist

Again, the SHA hashes in the file will be compared with SHA sums taken from the floppy (at the mountpoint). If anything has changed, the program will give a "failed" message. Unchanged files will be marked "OK".

The analysis

You can now view the contents of the restored disk or mounted image. If you are running the X window system, then you can use your favorite file

browser to look through the disk. In most (if not all) cases, you will find the command line more useful and powerful in order to allow file redirection and permanent record of your analysis. We will use the command line here.

We are also assuming that you are issuing the following commands from the proper mount point (*/mnt/analysis/*). If you want to save a copy of each command's output, be sure to direct the output file to your evidence directory (*/root/evidence/)*

Navigate through the directories and see what you can find. Use the **ls** command to view the contents of the disk. The command in the following form might be useful:

ls –al

This will show all the hidden files (**-a),** give the list in long format to identify permission, date, etc. (**-l**). You can also use the **–R** option to list recursivley through directories. You might want to pipe that through **less**.

ls –alR | less

Making a list of all files

Get creative. Take the above command and redirect the output to your evidence directory. With that you will have a list of all the files and their owners, permissions and time stamps on the suspect disk. This is a very important command. Check the **man** page for various uses and options. For example, you could use the **–i** option to include the inode in the list, the **–u** option can be used include and sort by access time (when used with the **–t** option).

ls –laiRtu > /root/evidence/file.list

You could also get a list of the files, one per line, using the find command and redirecting the output to another list file:

find . -type f -print > /root/evidence/filelist.list.2

Have a look at the above two commands, and compare their output. Which do you like better? Remember the above syntax assumes you are issuing the command from the */mnt/analysis* directory (use **pwd** if you don't

know where you are).

Now use the **grep** command on either of the file lists for whatever strings or extensions you want to look for.

grep -i jpg filelist.list

This command looks for the pattern “jpg” in the list of files.

Making a list of file types

What if you are looking for JPEG’s but the name of the file has been changed, or the extension is wrong? You can also run the command **file** on each file and see what it might contain.

file *filename*

If there are a large number of files without extensions, or where the extensions have changed, you might want to run the **file** command on *all* 0 Tc -0.6.536 -1[7.2

file

Viewing files

For text files and data files, you might want to use **cat**, **more** or **less** to view the contents.

cat *filename*
more *filename*
less *filename*

Be aware that if the output is not standart text, then you might corrupt the terminal output (type "*reset*" at the prompt and it should clear up). It is best to run these commands in a terminal window in X so that you can simply close out a corrupted terminal and start another. Using the **file** command will give you a good idea of which files will be viewable.

Perhaps a better alternative for viewing files would be to use the **strings** command. This command can be used to parse regular text out of any file. It's good for formatted documents (MS Word or StarOffice), data files (Excel, etc.) and even binaries (i.e. unidentified executables) might have interesting text strings hidden in them. It might be best to pipe the output through **less.**

strings *filename* | less

Have a look at the contents of the practice disk on */mnt/analysis*. There is a file called *arp.exe.* What does this file do? We can't execute it, and from using the **file** command we know that it's an i386 executable. Run the following command (again, assuming you are in the */mnt/analysis* directory) and scroll through the output.

strings arp.exe | less

Did you find anything of interest (hint: like a usage message for the executable)?

If you are currently running the X window system, you can use any of the graphics tools that come standard with whichever Linux Distribution you are using. **gqview** is one graphics tool for the **Gnome** desktop that will display thumbnails of all the recognized graphic files in a directory.

Experiment a little.

Once you are finished exploring, be sure to unmount the floppy (or image). Again, make sure you are not anywhere in the mountpoint when you try to unmount, or you will get the "busy" error.

umount /mnt/analysis

Searching unallocated and slack space for text

Now let's go back to the original image. The restored disk (or loop mounted image) allowed you to check all the files and directories. What about unallocated and slack space? We will now analyze the image itself, since it was a byte for byte copy and includes data in the unallocated areas of the disk, as well as file slack space.

Lets assume that we have seized this disk from a former employee of a large corporation. The would-be cracker sent a letter to the corporation threatening to unleash a virus in their network. The suspect denies sending the letter. This is a simple matter of finding the text from a deleted file (unallocated space).

First, change back to the directory in which you created the image, whether it was the root's home directory, or a special one you created.

cd /root/evidence

Now we will use the **grep** command to search the image for any instance of an expression or pattern. We will use a number of options to make the output of **grep** more useful. The syntax of **grep** is normally:

grep –options <pattern> <search_range>

The first thing we will do is create a list of keywords to search for. Open a text editor and make a list of terms you want to search for. For our example, lets use "ransom", "$50,000" (the ransom amount), and "unleash a virus". These are some keywords and a phrase that we have decided to use from the original letter received by the corporation. Make the list of keywords (using **vi**) and save it as */root/evidence/searchlist.txt*. Ensure that each string you want to search for is on a different line.

$50,000
ransom
unleash a virus

Make sure there are NO BLANK LINES IN THE LIST OR AT THE END OF THE LIST!! Now we run the **grep** command on our image:

grep –aibf searchlist.txt image.disk1 > hits.txt

Looking at the **grep** command we see that we are asking **grep** to use the list we created in "*searchlist.txt*" for the patterns we are looking for. This is specified with the "**-f *listfile***" option. We are telling **grep** to search *image.disk1* for these patterns, and we are redirecting the output to a file called *hits.txt*, so we can record the output and view them at our leisure. The option **–a** tells **grep** to output all the lines where there are hits, not just the name of the file where there are hits. The option **-i** tells **grep** to ignore upper and lower case. And the **-b** option tells **grep** to give us the byte offset of each hit so we can find the line in **xxd** or one of the graphical hex editors, like GHex.

Once you run the command above, you should have a new file in your current directory called *hits.txt*. View this file with **less** or **more** or any text viewer. Keep in mind that **strings** might be best for the job. Again, if you use **more** or **less**, you run the risk of corrupting your terminal. We will simply use **cat** to stream the entire contents of the file to the standard output. The file *hits.txt* should give you a list of lines that contain the words in your *searchlist.txt* file. In front of each line is a number that represents the byte offset for that line in the image file.

cat hits.txt

75441:you and your entire business ransom.
75500:I have had enough of your mindless corporate piracy and will no longer stand for it. (…)
75767:Don't try anything, and dont contact the cops. If you do, I will unleash a virus that will bring down your whole network and destroy your consumer's confidence.

Now open GHex. Find it on the KDE or Gnome menus, or simply type **ghex &** in a terminal window. It is a standard hex editor. Open the image file, and click on <Edit> and then <Goto Byte>. Type in the byte offset given in your hits.txt file and it should take you to that byte in the hex screen. the ASCII equivalent is displayed on the right. Do this for each offset in the list of hits. This should yield some interesting results. If you want to stay in the command line, you can use **xxd** to display the data found at each byte offset.

xxd -s *offset* image.disk1 | less

Handling large disks

The example used in this text utilizes a filesystem on a floppy disk. What happens when you are dealing with larger hard disks? When you create an image of a disk drive with the **dd** command there are a number of componets to the image. These components can include a boot sector, partition table, and the various partitions (if defined).

When you attempt to mount a larger image with the loop device, you find that the mount command is unable to find the filesystem on the disk, because it does not know how to "recognize" the partition table. The easy way around this (although it is not very efficient for large disks) would be to create separate images for each disk partition that you want to analyze. For a simple hard drive with a single large parition, you would create two images. These are "logical" images of each partition.

Assuming your suspect disk is attached as the master device on the secondary IDE channel:

dd if=/dev/hdc of=image.disk bs=4096 (gets the entire disk)
dd if=/dev/hdc1 of=image.part1 bs=4096 (gets the first partition)

The first command gets you a full image of the entire disk for backup purposes, including the boot sector and partition table. The second command gets you the partition. The resulting image from the second command can be mounted via the loop device.

image with the loop device) is to send the mount command a message to skip trying to mount the first 63 sectors of the image. These sectors are used to contain information (like the MBR) that is not part of a normal data partition. We know that each sector is 512 bytes, and that there are 63 of them. This gives us an offset of 32256 bytes from the start of our image to the first partition we want to mount. This is then passed to the mount command as an option:

mount –t vfat –o ro,noexec,loop,offset=32256 image.disk /mnt/analysis

This effectivley "jumps over" the first 63 sectors of the image and goes straight to the "boot sector" of the first partition, allowing the mount command to work properly.

When you are dealing with larger disks (over 2GB), you must also concern yourself with the size of your image files. If your Linux distribution relies on the 2.2.x kernel then you will encounter a file size limit of 2GB (on x86 systems). The Linux 2.4.x kernel solves this problem. You can either compile the 2.4.x kernel on your current system, or wait for a distribution that includes the 2.4.x kernel in it's default installation (they are on the shelf now). In order for the new 2.4.x kernel to work properly (if you upgrade it yourself), you will also need to upgrade *modutils* and *glibc*. Note that some 2.2.x kernel based distributions provide a fix for this "out of the box", but it is an exception rather than a rule.

Now that we know about the issues surrounding creating large images from whole disks, what do we do if we run into an error? Suppose you are creating a disk image with **dd** and the command exits halfway through the process with a read error? We can instruct **dd** to *attempt* to read past the errors using the **conv=noerror** option. In basic terms, this is telling the **dd** command to ignore the errors that it finds, and attempt to read past them. When we specify the **noerror** option it is a good idea to include the **sync** option along with it. This will "pad" the **dd** output wherever errors are found and ensure that the output will be "syncronized" with the original disk. This may allow filesystem access where errors are not fatal. The command will look something like:

dd if=/dev/hdx of=image.disk1 conv=noerror,sync

In addition to the structure of the images and the issues of image sizes,

we also have to concearn ourselves with memory usage and our tools. You might find that **grep**, when used as illustrated in our floppy analysis example, might not work as expected with larger images and could exit with an error similar to:

grep: memory exhausted

The most apparent cause for this is that **grep** does it's searches line by line. When you are "grepping" a large disk image, you might find that you have a huge number of bits to read through before **grep** comes across a newline character. What if **grep** had to read 200MB of data before coming across a newline? It would "exhaust" itself (the input buffer fills up).

What if we could force-feed **grep** some newlines? In our example analysis we are "grepping" for text. We are not concearned with non-text characters at all. If we could take the input stream to **grep** and change the non-text characters to newlines, grep would have no problem. Note that changing the input stream to **grep** does *not* change the image itself. Also, remember that we are still looking for a byte offset. Luckily, the character sizes remain the same, and so the offset does not change as we feed newlines into the stream (simply replacing one "character" with another).

Let's say we want to take all of the control characters streaming into **grep** from the disk image and change them to newlines. We can use the *translate* command, **tr**, to accomplish this. Check out **man tr** for more information about this powerful command:

tr '[:cntrl:]' '\n' < image.disk1 | grep -abif searchlist.txt > hits.txt

This command would read: "Translate all the characters contained in the set of *control characters ([:cntrl:])* to *newlines (\n)*. Take the input to **tr** from *image.disk1* and pipe the output to **grep,** sending the results to *hits.txt*. This effectivley changes the stream before it gets to **grep.**

This is only one of many possible problems you could come across. My point here is that when issues such as these arise, you need to be familiar enough with the tools Linux provides to be able to understand ***why*** such errors might have been produced, and how you can get around them. Remember, the shell tools and the GNU software that accompany a Linux distribution are extrememly powerful, and are capable of tackling nearly any

task. Where the standard shell fails, you might look at *perl* or *python* as options. These subjects are outside of the scope of the current presentation, but are introduced as fodder for further experimentation.

Preparing a disk for the suspect image

One common practice in forensic disk analysis is to "wipe" a disk prior to restoring a forensic image to it. This ensures that any data found on the restored disk is *from* the image and not from "residual" data. That is, data left behind from a previous case or image.

We can use a special device that is used as a source of zeros. This can be used to create empty files and wipe portions of disks. You can write zeros to an entire disk using the following command:

dd if=/dev/zero of=/dev/hdx bs=4096

This starts at the beginning of the drive and writes zeros in every sector in 4096 byte chunks. Specifying larger block sizes can speed the writing process. Experiment with different block sizes and see what effect it has on the writing speed (i.e. 64k). I've wiped 60GB disks in under an hour and a half on a fast IDE controller with the proper drive parameters. Specific drive parameters can be set using the **hdparm** command. Check **hdparm's** man page for available options**.** For instance, setting dma on a drive can dramatically speed things up.

Another function we might find useful would be the ability to split images up into usable chunks, either for archiving or for use in another program. For examble, you might have a 10GB image that you want to split into 640MB parts so they can be written to CD-R media. Or, if you use a program such as *Ilook Investigator* (*www.ilook.fsnet.co.uk/ilook/ilook.htm*) and need files no larger than 2GB (for a fat32 partition), you might want to split the image into 2GB pieces. For this we use the **split** command.

split normally works on lines of input (i.e. from a text file). But if we use the **–b** option, we force split to treat the file as *binary* input and lines are ignored. We can specify the size of the files we want along with the prefix we want for the output files. The command looks like:

split –b XXm <file to be split> <prefix of output files>

where **XX** is the size of the resulting files. For example, if we have a 6GB image called *image.disk1.dd*, we can split it into 2GB files using the following command:

split –b 2000m image.disk1.dd image.split.

This would result in 3 files (2GB in size) each named with the prefix "image.split." as specified in the command, followed by "aa", "ab", "ac", and so on:

image.split.aa
image.split.ab
image.split.ac

The process can be reversed. If we want to reassemble the image from the split parts (from CD-R, etc.), we can use the **cat** command and redirect the output to a new file. Remember **cat** simpley streams the specified files to standard output. If you redirect this output, the files are assembled into one.

cat image.split.aa image.split.ab image.split.ac > image.new
or
cat image.split.a* > image.new

The **sha1sum** of *image.new* will match that of *image.disk1.dd*. Test the above procedure on a floppy disk split into 360k parts.

Conclusion

Note that examples presented to you here are very simple. There are quicker ways of doing this, and more powerful ways. The steps above allow you to use common Linux tools and utilities that are helpful to the beginner. Once you become comfortable with Linux, you can extend the commands to encompass many more options. Start learning how to automate these tasks with shell scripts (shell scripts are your friend!). Eventually you can start applying this to the ***ext2*** filesystem used by Linux itself.

VIII. Linux Support

Web sites to check for support:

Look here first: The Linux Documentation Project (LDP):
http://www.linuxdoc.org

The Linux How-to Index
http://metalab.unc.edu/mdw/HOWTO/HOWTO-INDEX.htm

RedHat Software
http://www.redhat.com

Linux.com -Sponsored by VA Linux
http://www.linux.com

The XFree86 (X Window) Homepage
http://www.xfree86.org

The Official page of the Linux Kernel
http://www.kernel.org

The Linux Information Headquarters
http://www.linuxhq.com

Slashdot. News for Nerds. A must read, at least twice a day...
http://www.slashdot.org