# Unknown Binary Analysis, Debugfs Validation as a Forensics Tool, and Forensics Legal Issues

GIAC Certified Forensic Analyst (GCFA)
Practical Assignment

Version 1.4 (July 21, 2003)

By
Michael Harvey

August 19, 2003

## Table of Contents

## Summary

This document addresses the requirements of the GIAC Certified Forensic Analyst (GCFA) practical assignment version 1.4 dated 12 July 2003. The document is divided into three parts.

In Part 1, an unknown binary file was analyzed to determine its function relating to a case of an employee using employer computers to distribute illegal copies of copyrighted material to publicly available systems. A floppy disk image was provided for analysis containing an unknown file named "prog", which was the binary file of interest. Forensic techniques and tools were used to analyze the image to determine the type of filesystem, what files were present, and what was the purpose of the suspicious binary file "prog". Some of the tools that were used in the investigation included standard utilities such as "strings" (to display printable characters in binary files), "Autopsy" (a forensics tool providing sophisticated filesystem analysis capabilities), and "strace" (a Linux utility that traces system calls during a program's execution). Several suspicious files were found on the image, including a potentially incriminating message in a Word document, and the netcat utility. The unknown binary file was finally identified to be the "bmap" utility that is used to store files covertly in filesystem slack space.

In Part 2, the Linux utility "debugfs" was tested in order to validate its use as a forensics tool. Specifically, the ability of debugfs to display and recover deleted files was the focus of the testing. Debugfs was tested under seven different scenarios involving the recovery of files of varying size and number. Each scenario iwass performed using a fresh loopback filesystem stored in a single file on the test platform. The use of a loopback filesystem for testing allowed for the easy creation and manipulation of a fresh filesystem for every test run. The debugfs tool was found to be able to reliably identify and restore deleted files as long as the inode and data blocks from the deleted file had not been overwritten by filesystem activity.

In Part 3, the legal issues of incident handling were explored based on the scenario from Part 1. The laws relating to the illegal copying and distribution of copyrighted material were presented and their application to the given scenario was discussed. Also presented was a discussion of some of the steps that should be taken to preserve the evidence in an investigation and the impact that the presence of child pornography would have on the incident handling process.

# Part 1 - Analyze an Unknown Binary

In Part 1, an unknown binary program was analyzed as part of a forensics investigation based on a scenario provided by GIAC. The primary task was to determine what the binary program does and how it may have been used in illegal activity. The accuracy and detail of this information is critical to the investigation. The data obtained during this investigation provides the primary means of finding out if inappropriate activity has taken place, and the information from the investigation will be potentially used in a court of law if legal action is taken.

## 1.0 The scenario

The following scenario is provided to set the stage for the analysis of the unknown binary:

> An employee, John Price has been suspended from his place of employment when an audit discovered that he was using the organizations computing resources to illegally distribute copyrighted material. Unfortunately Mr. Price was able to wipe the hard disk of his office PC before investigators could be deployed. However, a single 3.5 inch floppy disk (the floppy disk image that you must use for this assignment can be **downloaded here**) was found in the drive of the PC. Although Mr. Price has subsequently denied that the floppy belonged to him, it was seized and entered into evidence:
>
> - Tag# fl-160703-jp1
> - 3.5 inch TDK floppy disk
> - MD5: 4b680767a2aed974cec5fbcbf84cc97a
> - fl-160703-jp1.dd.gz
>
> The floppy disk contains a number of files, including an unknown binary named 'prog'. Your primary task is to analyze this binary to establish its purpose, and how it might have been used by Mr. Price in the course of his alleged illegal activities. You should also examine the disk for any other evidence relating to this case. It is suspected that Mr. Price may have had access to other computers in the workplace.

Note that the personnel who performed the on-site investigation took great care to document and preserve the evidence. It is instructive to discuss what has been provided for the analysis. There is a formal tag number assigned to the floppy disk and the type and brand of disk is also noted. This information allows a formal chain of custody to be established so investigators will always be able to positively identify the disk and identify who has handled the disk, assuming the disk is placed in a controlled environment.

The binary image of the floppy disk's entire contents is stored in the file named fl-160703-jp1.dd.gz. The ".gz" extension implies that the file has been compressed with the gzip utility, commonly used on UNIX and Linux systems. The ".dd" part of the extension implies that the "dd" command was used to dump the entire contents of the floppy disk, bit by bit, to the file. The dd command is found installed by default on most UNIX and Linux operating systems. Thus the extensions of the file name imply that the floppy image was generated using a UNIX or Linux system. But note that the operating system used by Mr. Price's PC is not specified, so at this point in the investigation, it is not known what type of filesystem was on the floppy disk or what type of operating system the suspect binary will run on.

Lastly, note that the MD5 digest value of the image file is provided. MD5 is a cryptographic checksum program that reads in data of any length and produces a fixed-length128-bit digest value. Due to the nature of the cryptographic algorithm, the digest value is unique for every input data. If the input data changes at all, a different digest value will result. Thus, the MD5 checksum digest is a reliable way to check if data (including files) has changed. In the given scenario, when a copy of the .gz file is made, an md5 digest can be generated for the copy and compared to the md5 digest of the original file. If they match, it is guaranteed that the copy is an exact duplicate of the original. This is very useful in proving that an analysis was performed on an exact copy of the original evidence and that the results are the same as if the original evidence were used.

The md5 program source code can be downloaded from ftp://coast.cs.purdue.edu/pub/tools/unix/crypto/md5/. The MD5.README file at this site gives some more details about md5.

## 1.1 The Big Picture: Incident Response

Before proceeding with the analysis of the floppy disk image, it is useful to briefly discuss the steps of computer incident response and where forensics analysis fits in. There are six general steps to the forensics analysis process as described here.

1. **Preparation:** The preparation step involves having in place all of the resources that will be needed when an incident response is necessary. These resources include things like policies, people, analysis software and hardware, supplies, communication plan, and clear procedures.
2. **Identification:** In the identification step, a potential incident has been identified and the incident response team must determine it is real or if it is a false alarm. The incident response team investigates all available sources of information to determine the nature of the problem and to assess if the problem is adverse or malicious in nature. Note that to qualify as an incident, harm or the potential to cause harm must be present.
3. **Containment:** The containment step focuses on limiting the damage caused by the incident. Depending on the nature of the incident, this may involve pulling affected computers off of the network, implementing additional network or host security controls, or simply observing activity while the system remains operational. The nature of the containment response depends on the information gathered in the identification step.

4. **Eradication:** In the eradication step, the cause of the incident is determined and removed, and defenses are improved to reduce the risk of the incident occurring again. This step relies on the information gathered in the previous two steps.
5. **Recovery:** The recovery step involves bringing the system or systems back online and ensuring that the cause of the incident has been truly eliminated.
6. **Lessons Learned:** In the lessons learned phase, a formal report on the incident is written and the overall incident response is reviewed to identify weaknesses and ways to improve responses in the future.

Forensic analysis is essentially step two, the identification phase of incident response, but forensics techniques and the information gained from them are used throughout steps two, three, four, and five. For the scenario presented here, the primary job of the forensic analyst is to identify evidence of Mr. Price's wrongdoing. The fact that an incident has occurred has already been established by the audit mentioned in the scenario, and suspending Mr. Price has contained the incident and hopefully eradicated the cause. The goal of the forensic analysis is to provide solid evidence of Mr. Price's illicit activities, identify the exact methods used to carry out those activities, and to provide information on how to detect the use of those same methods in the future.

## 1.2 Analysis Platform Description

The platform used to analyze the floppy image and the unknown binary is a Dell Inspiron 5100 Laptop computer running the Windows XP SP1 operating system with the latest Microsoft security fixes as of 8/1/03 installed from http://windowsupdate.microsoft.com.

The VMware Workstation 3.2.0 application is installed to provide a virtual Linux machine and a virtual network within which the binary can be safely analyzed. VMware emulates the hardware and BIOS (Basic Input Output System) of a computer for each virtual computer that it runs. VMware also emulates a network within the host computer (the Windows XP system in this case) so the different virtual machines can communicate with each other. This is an ideal situation for forensics analysis because the virtual machines and network are not exposed to external systems where the potentially malicious software being run might do harm. And since each virtual machine is simply a set of files on the host machine's file system, it is very easy to make copies of a virtual machine at different points in time and to easily revert back to one of those copies.

The virtual machine used for this analysis was running the Red Hat 8 Linux operating system with the latest security updates applied. The updates were downloaded from the Red Hat mirror site ftp://ftp.dulug.duke.edu/pub/redhat/linux/updates/8.0/en/os/i386/, written to CDROM, and copied into the directory /var/tmp/updates on the virtual Linux machine. The updates were installed by executing the following commands from the Linux shell (command line) within the VMware virtual Linux machine:

```
# cd /var/tmp/updates
# rpm -Fv *
```

Note that it is important to keep the host operating system (Windows XP in this case) updated with the latest security fixes and good security practices to minimize the risk of the host operating system being affected by the testing of suspicious programs within the VMware virtual machines. This is a possibility because VMware creates a virtual network interface on the host operating system that interfaces with the VMware virtual network and the virtual machines.

The Linux virtual machine has the forensics tools TASK 1.6 and Autopsy 1.70 installed. More information on the TASK tool can be found at http://www.sleuthkit.org/sleuthkit/index.php and more information on the Autopsy tool can be found at http://www.sleuthkit.org/autopsy/index.php.

More information on the VMware product, including full documentation, can be found at the VMware Web site http://www.vmware.com.

## 1.3 Move Image to Analysis Platform

The first step in the forensics analysis will be to analyze the image of the floppy disk found in Mr. Price's computer. Since Mr. Price was able to wipe the hard disk of his PC, the floppy disk is the only available evidence from the incident.

The first step is to load the binary into the VMware virtual machine for analysis. To accomplish this, the compressed image file provided, fl-160703-jp1.dd.gz, was copied to a standard Windows floppy disk with a FAT filesystem from the Windows computer on which it was downloaded from the network. This floppy was then mounted under the Linux virtual machine and the binary copied off into the directory /root/GCFA which was created to store the information that will be generated during this investigation. The GCFA directory was empty before this binary was copied in, ensuring that no pre-existing files could be confused with the evidence being gathered. Here are the steps taken to accomplish the copy, done as the root user on the Linux virtual machine.

- Make the VMware virtual machine see the floppy drive by choosing the VMware menu selection Devices -> floppy0 -> Connect. The floppy becomes the device /dev/fd0 on the Linux system.
- Mount the floppy using the Linux disk manager tool from the Red Hat menu, System Tools -> Disk management. Select the /dev/fd0 device and select Mount. The floppy is mounted to the Linux filesystem as /mnt/floppy.
- Copy the file with the following command:

```
cp /mnt/floppy/fl-160703-jp1.dd.gz
```

- Verify the integrity of the file to ensure it was not changed in any way in the copy process. Generate the md5 digest value of the file and compare it with the value given with the original file. Here are the commands and output from the Linux machine. The “#” is the command prompt.

```
# cd /root/GCFA
# ls -l
-r-xr-xr-x   1   root   root   474162  Aug  2  14:34  fl-
 160703-jp1.dd.gz
# md5sum fl-160703-jp1.dd.gz
  4b680767a2aed974cec5fbcbf84cc97a  fl-160703-jp1.dd.fl
```

In the output above, the file’s attributes are first shown using the “ls -l” command to provide a “long” listing of the file. The output reveals that the file is owned by the root user, is in the root group, and has a size of 474162 bytes. Note that in the output above, the file name wrapped around the screen and the output of “ls --l should be one long line. For more information on the meaning of the “ls -l”

The integrity is verified by the MD5 digest for the file, which is given in the last line of output above. The 32-character digest value matches the value provided with the original binary, proving they are identical files and that no corruption occurred during the copy process.

The next step is to uncompress the image using the gunzip program. Following are the commands used to do this, and the output of “ls -l” after uncompressing.

```
# cd /root/GCFA
# gunzip fl-160703-jp1.dd.gz
# ls -l
-r-xr-xr-x   1  root   root   1474560  Aug  3  14:34
fl-160703-jp1.dd
```

Note that the size of the file is now 1.474MB, an appropriate size for an image of a 1.44MB floppy disk. Also note that there is no write permission for this file, helping to minimize the chance of accidentally introducing unwanted changes. For reference, generate an MD5 digest value for this file as follows.

```
# md5 fl-160703-jp1.dd
20be7bc13a5cb8d77232659c52a3ba65  fl-160703-jp1.dd
```

## 1.4 Analyze the Raw Image

The raw binary image of the floppy disk is now on the Linux analysis system. The first step is to see if Linux can tell what kind of file it is. Recall that at this point, it’s not certain if the floppy contained a Windows FAT filesystem or a Linux filesystem. The “file” command will try to identify the type of file.

```
# file fl-160703-jp1.dd
fl-160703-jp1.dd: Linux rev 1.0 ext2 filesystem data
```

As the output above shows, the image is that of a Linux ext2 filesystem. Thus Mr. Price's PC must have been running the Linux operating system since Windows operating systems cannot read the Linux ext2 filesystem natively.

Next, take a look at the printable characters within the image, which may give a hint as to what was on the floppy disk. The Linux "strings" command prints all printable characters inside of a binary file. Some of the output from the strings command is meaningless binary data whose bits happen to match those of a printable character, but other output can be very useful. In this case, within the image file the strings command should be able to show the contents of text files, the contents of directories (including file names), and any other printable characters that were stored on the floppy disk. Even the contents of deleted files will show up if they haven't been overwritten. Note that the strings command's output is not particularly well formatted and that on a disk image such as the one here, it will not be clear what text is associated with directory entries vs. file contents, or with what file name any particular text is associated with. The strings output is just for a "quick look" at what's on the disk.

As shown in the command below, the strings command is executed on the image file and the results are stored in the file strings_of_image.txt. The "-t x" option to strings causes the hexadecimal offset with the file to be printed for each string. This command is executed from the /root/GCFA directory.

```
# strings –t x fl-160703-jp1.dd > strings_of_image.txt
```

The file strings_of_image.txt can then be viewed in a standard text editor. In this case, the Nautilus text editor was used to view the file on the Linux analysis system. The numbers on the left of each line are the hexadecimal offset into the file for the string on that line. Following are some of the more interesting items in the output.

Following are the first several lines of the output.

```
   7020 lost+found
   7034 John
   7040 progt
   7050 May03
   7060 Docs
   706c nc-1.10-16.i386.rpm..rpm
   708c prog
   7098 .~5456g.tmp
   a420 sect-num.gif
   a434 sectors.gif
   a602 UU a
   a648 vmware
   a688 cd ..
   a738 vmware-config.pl
   a758 vmware
```

At least some of these lines could be file or directory names. "lost+found" is a standard directory present in ext2 filesystems. It is known from the information provided that the suspicious program is named "prog", and prog shows up on line seven of the output above. The potential file or directory names of John, progt, and Docs are also seen.

"nc-1.10-16.i386.rpm" is the name of the netcat program packages for installation on a Red Hat Linux system using the Red Hat package manager (rpm). Netcat is a program used to open up a simple network connection between computers. The connection can be used for creating an interactive login session, copying files, and many other creative things.

The presence of "vmware" could indicate that VMware is involved in this incident somehow. A search using the Google Internet search engine (http://www.google.com) found the reference http://www.vmware.com/support/ws2/doc/prebuilt_modules_ws_linux.html, which reveals that "vmware-config.pl" is a Perl script which is run to configure the kernel for VMware services after VMware is installed on a Linux system.

Following is some more output from the strings command.

```
   a820 xmms-mpg123-1.2.7-13.i386.rpm..rpmUU
   ac20 ebay300.jpg
   b020 Letter.doc
   b034 Mikemsg.doc
   b048 DVD-Playing-HOWTO-html.tar.gz
   b070 Kernel-HOWTO-html.tar.gz
   b090 MP3-HOWTO-html.tar.gz
   b0b0 Sound-HOWTO-html.tar.gz
   b0d0 DVD-Playing-HOWTO-html.tar
```

Doing another Google search for xmms reveals that xmms is an mp3 sound file player for Linux. The file xmms-mpg123-1.2.7-13.i386.rpm is the name of the Red Hat RPM package for xmms. One of several Web sites that offered the rpm file for download was http://havardk.xmms.org/dist/xmms-1.2.7-rh8-rh9-rpm/.

The ebay300.jpg could be the name of a JPEG image file, but more importantly it could indicate that the use of ebay, the online auction Web site, is involved in this incident, but more information is needed before any conclusions can be made. An analyst at this point must avoid pure speculation, but must be able to take hints for things to watch out for as more information is obtained. A text search of the strings output using the text editor's Find feature revealed no other occurrences of "ebay".

There are two .doc files, the standard file extension for Microsoft Word documents. The other file names in the output above (assuming they are file names at this point) provide further indications that multimedia applications are involved.

This output is particularly interesting.

```
b034 Mikemsg.doc
13200 Hey Mike,
1320b I received the latest batch of files last night and I
13241 m ready to rock-n-roll (ha-ha).
13262 I have some advance orders for the next run. Call me
      soon.
14ec8 Hey Mike,
14ee8 John Price
14f08 Normal
14f18 John Price
14f38 Microsoft Word 8.0
162bc CCNOU
16310 Hey Mike,
1632a Title
16364 _PID_GUID
17a20 Microsoft Word Document
17a3c MSWordDoc
17a4a Word.Document.8
```

Taken together, this appears to be a Microsoft Word document containing a message from Mr. Price to someone named Mike. The first line (b034) is from early in the file near where the other suspected filenames are located. The rest of the output was later in the file and was together sequentially. This could be very valuable evidence to present to company management or law enforcement to show the activities that Mr. Price may have been involved in. But the forensics analyst cannot jump to any conclusions; the analyst's job is to record and present evidence. Note that where a line does not have the leading offset hex number, its contents are wrapped from the previous line. This was done manually to make the output more readable for this report. The original file strings_of_image.txt would be used as the official unaltered evidence.

Here is some more output from the strings command.

```
3fcaf <LI> Get the LiViD utilities from CVS, so you can update
      it as it is being
3fcfb updated by the developers. (recommended)</LI>
3fd29 </UL>
3fd33 <PRE>
3fd3b        The following commands will retrieve them:
3fd71        # mkdir ~/livid
3fd88        # cd ~/livid
3fd9c        # export CVSROOT=:pserver:anonymous@cvs.
             linuxvideo.org:/cvs/livid
3fde5        # cvs login
3fdf8        (Logging in to anonymous@cvs.linuxvideo.org)
3fe2c        CVS password:
3fe41
3fe4a        There is no password for anonymous, just press
             enter.
3fe87
3fe91        # cvs -z3 co -P ac3dec oms mpeg2dec mgadvd
3fec4        They should download into their respective
             directories.
3ff03 </PRE>
```

The text above was taken from the middle of a larger portion of text that seems to be instructions for how to download code for the linuxvideo project. The "<PRE>" and "</UL>" are HTML tags, implying that this is part of an HTML document. The Web site http://www.linuxvideo.org/ describes the Linux video project as follows:

> *The LiViD Project is a collection of video and dvd related sub-projects. The idea is to provide one central location for users to find information and support for video hardware and software. The other major advantage for organizing all the video related projects is the reduction of the amount of duplicated code, and a better collaboration on the design of interfaces.*

Just before the listing of device file names, there is a section that appears to be text from a program. The work "prog" appears in it, so it may be of interest to the investigation. Here is selected output from the text of the strings output.

```
a0e00 test for fragmentation (returns 0 if file is fragmented)
a0e39 checkfrag
a0e60 display fragmentation information for the file
a0e8f frag
a0ea0 wipe the file from the raw device
a0ee0 print number of bytes available
a0f03 test (returns 0 if exist)
a0f21 wipe
a0f28 place data
a0f35 display data
a0f60 extract a copy from the raw device
a0f85 list sector numbers
a0f9b operation to perform on files
a0fb9 mode
a10c0 prog
a10c5 main
a10db 07/15/03
a10e6 invalid option: %s
a10f9 try '--help' for help.
a1110 how did we get here?
a1140 no filename. try '--help' for help.
a1164 target filename: %s
a1178 Unable to stat file: %s
a1191 %s is not a regular file.
a11ab %s has multiple links.
a11c2 Unable to open file: %s
a11da Unable to determine blocksize
a11f8 target file block size: %d
a13c5 stuffing block %d
a13d7 %s has slack
a13e4 %s does not have slack
a13fb %s has fragmentation
a1420 %s does not have fragmentation
a14d5 NULL value for slack_block
a15c9 bmap_get_block_count
a1600 unable to determine filesystem blocksize
a1678 stat reports %d blocks: %d
a1693 bmap_get_block_size
a16c0 nul block while mapping block %d.
a16e2 bmap_raw_open
a16f0 NULL filename supplied
a1707 Unable to stat file: %s
a171f %s is not a regular file.
a1740 unable to determine raw device of %s
a179f unable to open raw device %s
a17bc raw fd is %d
a17c9 bmap_raw_close
a17d8 /.../image
a17e3 bogowipe
a17ec write error
a1800 /dev/xdb9
a180a /dev/xdb8
```

The listing above could be part of a program file. If it is, some of the elements suggest a function relating to raw filesystem access. The mention of “filename”, “fragmentation”, "bogowipe" and the listing of raw devices that follows may suggest manipulation of raw disk devices to destroy files. The text towards the beginning could be a menu presenting options to be selected. The line "wipe the file from the raw device" reinforces the idea that this program involves wiping data. At the end of the output, there is a long section listing device files, such as "/dev/xdb8" and "/dev/sdy7". The listing above shows only the beginning of the device list.

Speculation: maybe this is a portion of the program Mr. Price used to wipe his PC?

The rest of the file seems to be either binary or text portions of Linux libraries or programs. The strings analysis of the floppy image file has given some interesting clues as to what Mr. Price may have been up to. The apparent interest in multimedia, especially mp3 and DVD, is consistent with the reasoning behind Mr. Price's suspension from the company.

## 1.5 Mount the Image

For further analysis, the floppy image will be mounted on the Linux analysis system so that the image can be interacted with as a real directory structure.

On the Linux analysis system as the root user, execute the following commands to mount the floppy image. Mounting the floppy image uses the loopback device, which allows an image file such as we have here to be mounted just as an actual disk device can be mounted. The image will be mounted under the directory /root/GCFA/floppy_mount. The image file is at /root/GCFA/fl-160703-jp1.dd.

```
cd /root/GCFA
mount –o loop,ro fl-160703-jp1.dd /root/GCFA/floppy_mount
```

The "-o" option specifies the options to apply to the mount. Here, the "loop" option specifies to use a loopback mount. The "ro" option means to mount the image read-only so there is no chance of corrupting the image by writing to it.

Now the floppy filesystem can be viewed by changing directory into the mounted filesystem and listing the contents with the following command. The "-laR" option to the ls command causes a recursive listing of all files and directories present along with the "long" listing of information associated with each entry. For lines where the listing wrapped around the screen, the wrapped part of line was manually indented for better clarity.

The column headings were manually added in bold for clarity.

```
# cd /root/GCFA
# ls -laR floppy_mount
floppy_mount:
total 560

PERMISSIONS     OWNER    GROUP         SIZE  DATE  TIME  NAME
drwxr-xr-x    6 root     root          1024 Jul 16 02:03 .
drwxr-xr-x    5 root     root          4096 Aug  5 21:37 ..
-rw-r--r--    1 root     root          2592 Jul 14 10:13 .~5456g.tmp
drwxr-xr-x    2 502      502           1024 Jul 14 10:22 Docs
drwxr-xr-x    2 502      502           1024 Feb  3  2003 John
drwx------    2 root     root         12288 Jul 14 10:08 lost+found
drwxr-xr-x    2 502      502           1024 May  3 06:10 May03
-rwxr-xr-x    1 502      502          56950 Jul 14 10:12 nc-1.10-
 16.i386.rpm..rpm
-rwxr-xr-x    1 502      502         487476 Jul 14 10:24 prog

floppy_mount/Docs:
total 171
drwxr-xr-x    2 502      502           1024 Jul 14 10:22 .
drwxr-xr-x    6 root     root          1024 Jul 16 02:03 ..
-rwxr-xr-x    1 502      502          29184 May 21 06:09 DVD-Playing-
 HOWTO-html.tar
-rwxr-xr-x    1 502      502          27430 May 21 06:09 Kernel-
 HOWTO-html.tar.gz
-rw-------    1 502      502          29696 Jun 11 09:09 Letter.doc
-rw-------    1 502      502          19456 Jul 14 10:48 Mikemsg.doc
-rwxr-xr-x    1 502      502          32661 May 21 06:12 MP3-HOWTO-
html.tar.gz
-rwxr-xr-x    1 502      502          26843 Jul 14 10:11 Sound-HOWTO-
 html.tar.gz

floppy_mount/John:
total 44
drwxr-xr-x    2 502      502           1024 Feb  3  2003 .
drwxr-xr-x    6 root     root          1024 Jul 16 02:03 ..
-rwxr-xr-x    1 502      502          19088 Jan 28  2003 sect-num.gif
-rwxr-xr-x    1 502      502          20680 Jan 28  2003 sectors.gif

floppy_mount/lost+found:
total 13
drwx------    2 root     root         12288 Jul 14 10:08 .
drwxr-xr-x    6 root     root          1024 Jul 16 02:03 ..

floppy_mount/May03:
total 17
drwxr-xr-x    2 502      502           1024 May  3 06:10 .
drwxr-xr-x    6 root     root          1024 Jul 16 02:03 ..
-rwxr-xr-x    1 502      502          13487 Jul 14 10:12 ebay300.jpg
```

The listing verifies that some of the names that were suspected as being file and directories from the strings output were indeed the names of files and directories on the floppy's filesystem (Docs, John, prog, etc.). The owner and group of each file are either root (user id 0 on any Linux system) or 502. Since the Linux analysis system has no password file name entry for user id 502, the user id number is printed.

Some of the files were examined with “strings” to see what was in them. The Docs/Mikemsg.doc file contains the text seen in the “strings” of the raw image and seems to be a Microsoft Word document. Here is the message in the file.

```
I received the latest batch of files last night and I
m ready to rock-n-roll (ha-ha).
I have some advance orders for the next run. Call me soon.
```

The file “.~5456g.tmp” has an odd name, but contains no printable characters. It is a pure binary file. The “file” command identifies it as data.

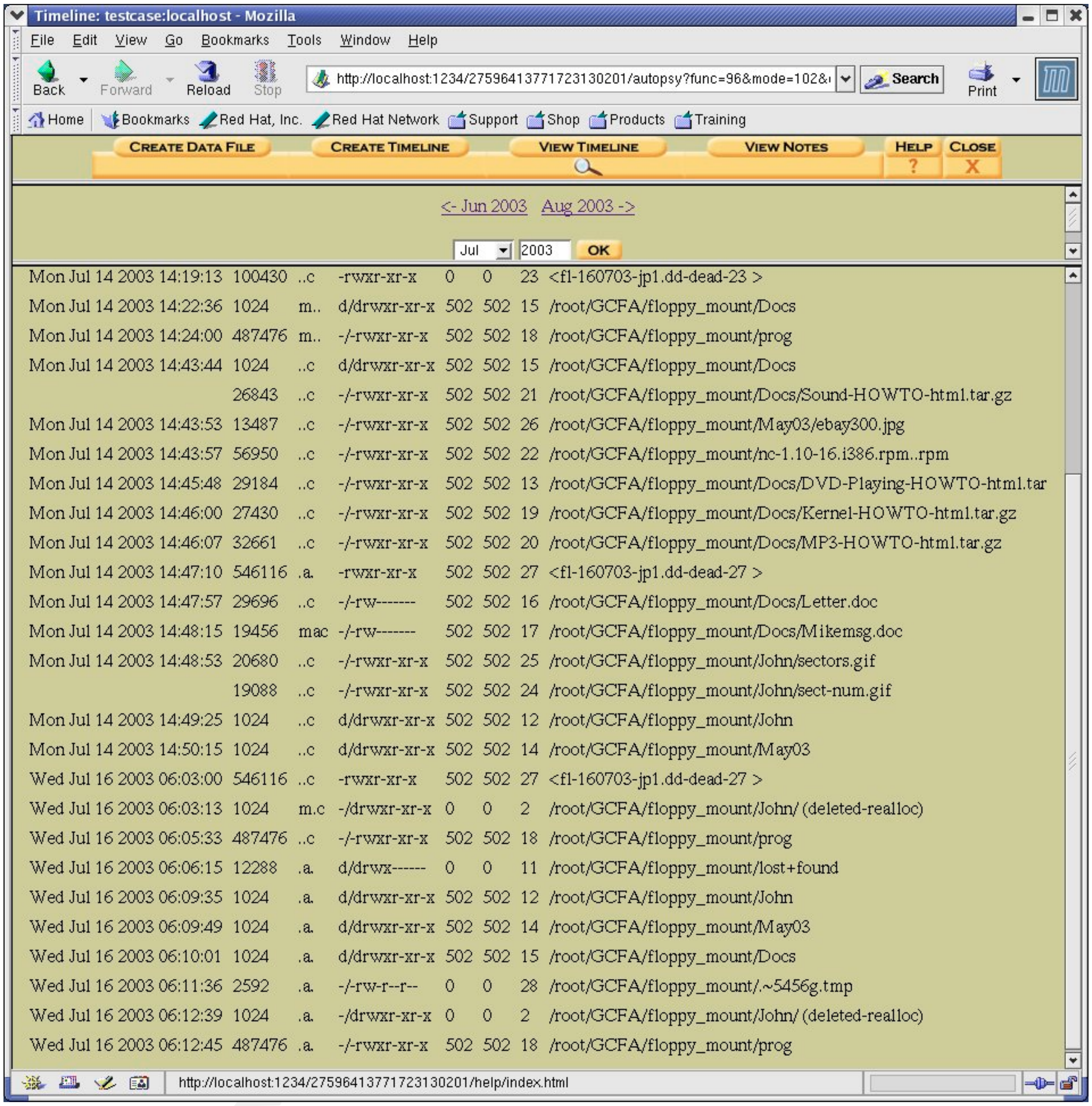
## 1.6 Load image into the Autopsy Tool

Autopsy is a Web front-end to the TASK (The @stake Sleuth Kit) set of system forensics tools. The tools within TASK allow for the analysis if filesystem images to extract information on files and access deleted files, as well as investigate the lower-level filesystem constructs such as inodes and data blocks. Autopsy was used here to verify the MAC (Modification, Access, Change) times of the “prog” executable and to check if there were any interesting deleted files on the floppy image. The details of using Autopsy won’t be discussed here in order to apply more time to the direct analysis of the binary program, only the results from Autopsy will be presented.

The image file was loaded into the Autopsy forensic analysis tool in order to gather further information on the filesystem. A new case was created in the Autopsy tool and a timeline was generated. The modification, access, and change times of the prog file were examined. The times matched those shown in Figure 1-1, except for being four hours ahead. The time offset is due to the time GMT time zone setting chosen for Autopsy vs. the EDT setting on the analysis system. The times shown by Autopsy were as follows:

**Modification Time:** July 14 14:24:00
**Change Time:** July 16 06:05:33
**Access Time:** July 16 06:12:45

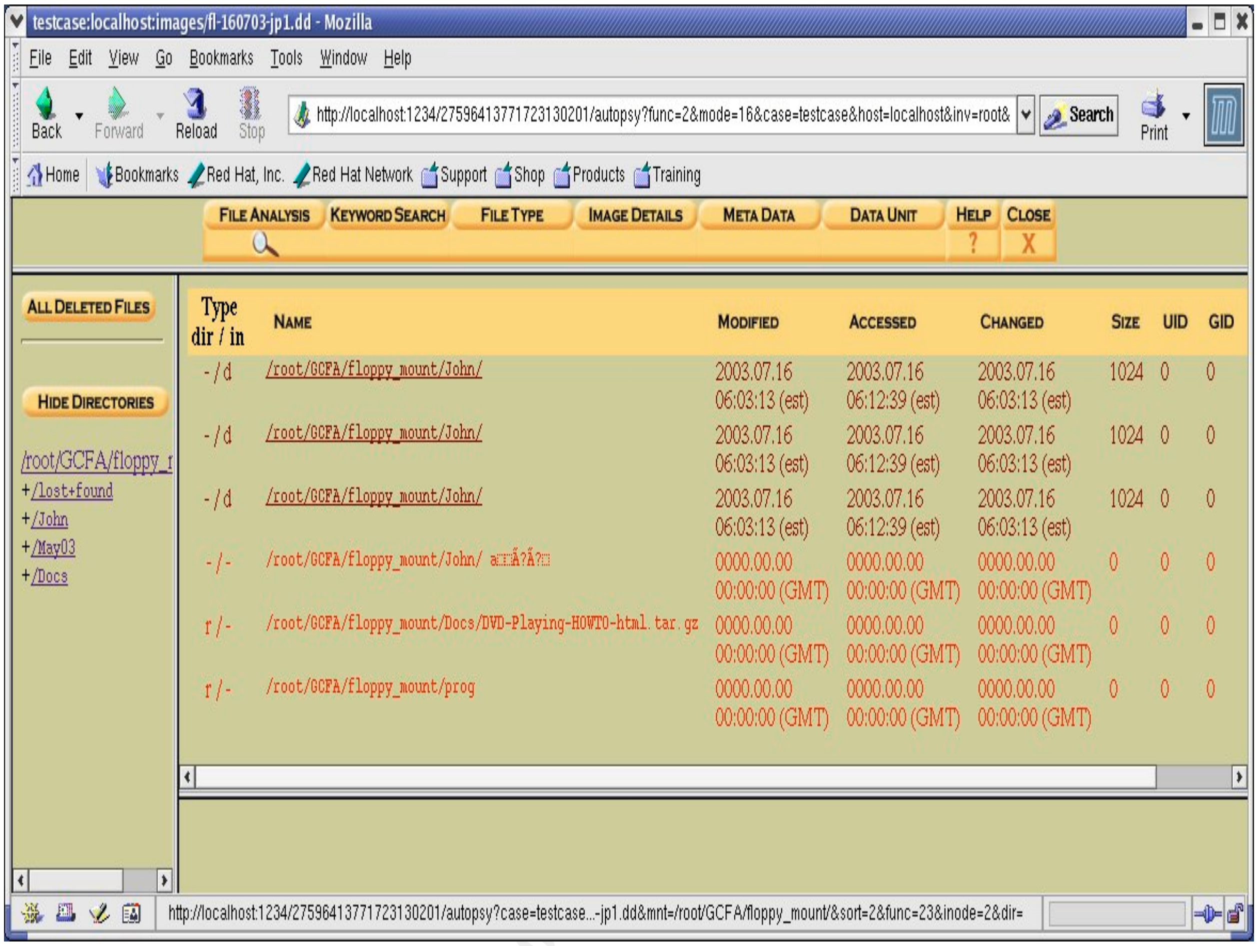
Figure 1-1 shows the screen shot of the tail end of the filesystem timeline containing the activity relating to the prog file.


Timeline: testcase:localhost - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://localhost:1234/27596413771723130201/autopsy?func=96&mode=102&
Search Print
Home Bookmarks Red Hat, Inc. Red Hat Network Support Shop Products Training
CREATE DATA FILE CREATE TIMELINE VIEW TIMELINE VIEW NOTES HELP CLOSE
<- Jun 2003 Aug 2003 ->
Jul 2003 OK
http://localhost:1234/27596413771723130201/help/index.html

```
Mon Jul 14 2003 14:19:13 100430 ..c -rwxr-xr-x   0   0   23 <fl-160703-jp1.dd-dead-23 >
Mon Jul 14 2003 14:22:36 1024   m.. d/drwxr-xr-x 502 502 15 /root/GCFA/floppy_mount/Docs
Mon Jul 14 2003 14:24:00 487476 m.. -/-rwxr-xr-x 502 502 18 /root/GCFA/floppy_mount/prog
Mon Jul 14 2003 14:43:44 1024   ..c d/drwxr-xr-x 502 502 15 /root/GCFA/floppy_mount/Docs
                         26843  ..c -/-rwxr-xr-x 502 502 21 /root/GCFA/floppy_mount/Docs/Sound-HOWTO-html.tar.gz
Mon Jul 14 2003 14:43:53 13487  ..c -/-rwxr-xr-x 502 502 26 /root/GCFA/floppy_mount/May03/ebay300.jpg
Mon Jul 14 2003 14:43:57 56950  ..c -/-rwxr-xr-x 502 502 22 /root/GCFA/floppy_mount/nc-1.10-16.i386.rpm..rpm
Mon Jul 14 2003 14:45:48 29184  ..c -/-rwxr-xr-x 502 502 13 /root/GCFA/floppy_mount/Docs/DVD-Playing-HOWTO-html.tar
Mon Jul 14 2003 14:46:00 27430  ..c -/-rwxr-xr-x 502 502 19 /root/GCFA/floppy_mount/Docs/Kernel-HOWTO-html.tar.gz
Mon Jul 14 2003 14:46:07 32661  ..c -/-rwxr-xr-x 502 502 20 /root/GCFA/floppy_mount/Docs/MP3-HOWTO-html.tar.gz
Mon Jul 14 2003 14:47:10 546116 .a. -rwxr-xr-x   502 502 27 <fl-160703-jp1.dd-dead-27 >
Mon Jul 14 2003 14:47:57 29696  ..c -/-rw-------  502 502 16 /root/GCFA/floppy_mount/Docs/Letter.doc
Mon Jul 14 2003 14:48:15 19456  mac -/-rw-------  502 502 17 /root/GCFA/floppy_mount/Docs/Mikemsg.doc
Mon Jul 14 2003 14:48:53 20680  ..c -/-rwxr-xr-x 502 502 25 /root/GCFA/floppy_mount/John/sectors.gif
                         19088  ..c -/-rwxr-xr-x 502 502 24 /root/GCFA/floppy_mount/John/sect-num.gif
Mon Jul 14 2003 14:49:25 1024   ..c d/drwxr-xr-x 502 502 12 /root/GCFA/floppy_mount/John
Mon Jul 14 2003 14:50:15 1024   ..c d/drwxr-xr-x 502 502 14 /root/GCFA/floppy_mount/May03
Wed Jul 16 2003 06:03:00 546116 ..c -rwxr-xr-x   502 502 27 <fl-160703-jp1.dd-dead-27 >
Wed Jul 16 2003 06:03:13 1024   m.c -/drwxr-xr-x 0   0   2  /root/GCFA/floppy_mount/John/ (deleted-realloc)
Wed Jul 16 2003 06:05:33 487476 ..c -/-rwxr-xr-x 502 502 18 /root/GCFA/floppy_mount/prog
Wed Jul 16 2003 06:06:15 12288  .a. d/drwx------  0   0   11 /root/GCFA/floppy_mount/lost+found
Wed Jul 16 2003 06:09:35 1024   .a. d/drwxr-xr-x 502 502 12 /root/GCFA/floppy_mount/John
Wed Jul 16 2003 06:09:49 1024   .a. d/drwxr-xr-x 502 502 14 /root/GCFA/floppy_mount/May03
Wed Jul 16 2003 06:10:01 1024   .a. d/drwxr-xr-x 502 502 15 /root/GCFA/floppy_mount/Docs
Wed Jul 16 2003 06:11:36 2592   .a. -/-rw-r--r--  0   0   28 /root/GCFA/floppy_mount/.~5456g.tmp
Wed Jul 16 2003 06:12:39 1024   .a. -/drwxr-xr-x 0   0   2  /root/GCFA/floppy_mount/John/ (deleted-realloc)
Wed Jul 16 2003 06:12:45 487476 .a. -/-rwxr-xr-x 502 502 18 /root/GCFA/floppy_mount/prog
```

**Figure 1-1: Autopsy Timeline**

Autopsy was next used to check for deleted files on the image. After selecting the Host Manager menu and the floppy image, select OK. At the next screen, select File Analysis. Then choose the “Expand Directories” and “All Deleted Files” options. Figure 1-2 is the screenshot of the resulting output.


testcase:localhost:images/fl-160703-jp1.dd - Mozilla

http://localhost:1234/27596413771723130201/autopsy?func=2&mode=16&case=testcase&host=localhost&inv=root&

| Type dir / in | Name | Modified | Accessed | Changed | Size | UID | GID |
|---|---|---|---|---|---|---|---|
| - / d | /root/GCFA/floppy_mount/John/ | 2003.07.16 06:03:13 (est) | 2003.07.16 06:12:39 (est) | 2003.07.16 06:03:13 (est) | 1024 | 0 | 0 |
| - / d | /root/GCFA/floppy_mount/John/ | 2003.07.16 06:03:13 (est) | 2003.07.16 06:12:39 (est) | 2003.07.16 06:03:13 (est) | 1024 | 0 | 0 |
| - / d | /root/GCFA/floppy_mount/John/ | 2003.07.16 06:03:13 (est) | 2003.07.16 06:12:39 (est) | 2003.07.16 06:03:13 (est) | 1024 | 0 | 0 |
| - / - | /root/GCFA/floppy_mount/John/ a□□Å?Å?□ | 0000.00.00 00:00:00 (GMT) | 0000.00.00 00:00:00 (GMT) | 0000.00.00 00:00:00 (GMT) | 0 | 0 | 0 |
| r / - | /root/GCFA/floppy_mount/Docs/DVD-Playing-HOWTO-html.tar.gz | 0000.00.00 00:00:00 (GMT) | 0000.00.00 00:00:00 (GMT) | 0000.00.00 00:00:00 (GMT) | 0 | 0 | 0 |
| r / - | /root/GCFA/floppy_mount/prog | 0000.00.00 00:00:00 (GMT) | 0000.00.00 00:00:00 (GMT) | 0000.00.00 00:00:00 (GMT) | 0 | 0 | 0 |

/root/GCFA/floppy_r
+/lost+found
+/John
+/May03
+/Docs

http://localhost:1234/27596413771723130201/autopsy?case=testcase...-jp1.dd&mnt=/root/GCFA/floppy_mount/&sort=2&func=23&inode=2&dir=

File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop Search Print
Home Bookmarks Red Hat, Inc. Red Hat Network Support Shop Products Training
File Analysis Keyword Search File Type Image Details Meta Data Data Unit Help Close
All Deleted Files
Hide Directories

**Figure 1-2: Autopsy Deleted Files Display**

The deleted file prog (last in the list) might have been a previous version of the "prog" binary that was deleted. Autopsy cannot gather any further information from this deleted file (the MAC times and other info are all "0"), so there's probably not much more to learn.

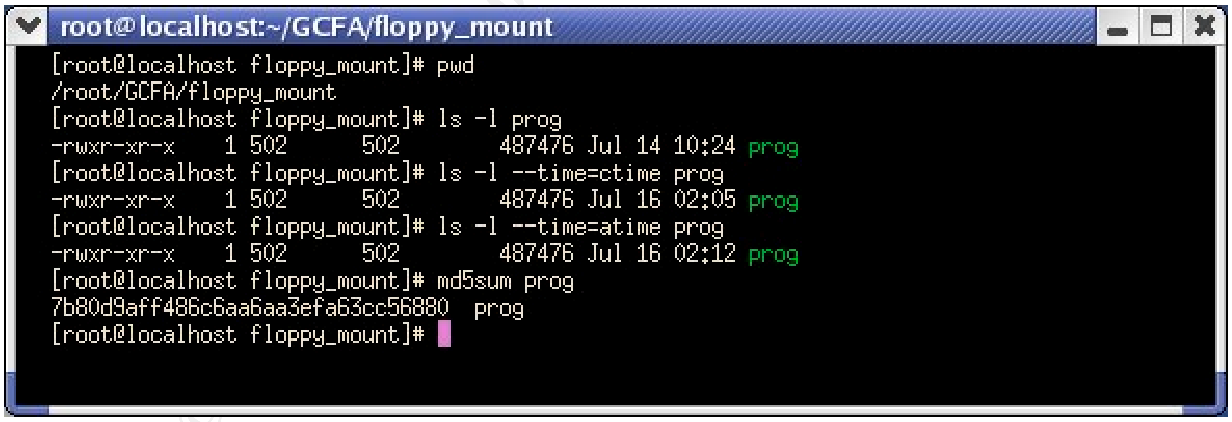
## 1.7 Binary Details of the prog File

The primary mission in Part 1 is to analyze the suspicious binary "prog", so it is necessary now to concentrate the limited analysis time available on this file, even if it is at the expense of examining the other data further. The forensic analyst must make wise decisions on where to spend valuable analysis time so that the most important information is gathered with the least cost in time.

From the mounted image, it is easy to gather most the initial information required by the practical assignment.

Figure 1-3 below is a screen shot of the shell used to display information on the prog file. The following information can be gathered from this output.

- **True name of the program/file found on the system.** The name of the executable file in the floppy image is “prog”. Upon further analysis as described in the following sections, the program was identified as the “bmap” program, which is a tool to covertly store data in filesystem slack space.
- **File/MAC Time information.** The “ls -l” command displays the file’s modification time, which is 10:24 July 14 here (based on the timezone of EDT on the analysis system). The “ls -l --time=ctime” command displays the time of last inode change, which is 02:05 July 16. The “ls -l --time=atime” displays the time of last access of the file’s data, which here is 02:12 July 16.
- **File owner/group.** As seen in Figure 1-3, the owner and group of the prog file are both 502. The numerical user id and group id is displayed because the analysis system has no password or group file entry for 502.
- **File size.** The size of the file is shown in the listing in Figure 1-3 as 487476 bytes.
- **MD5 hash of the file.** The MD5 digest value of the file was generated with the md5sum command and is shown in Figure 1-3 as 7b80d9aff486c6aa6aa3efa63cc56880.
- **Key words found that are associated with the program/file.** This was discussed above in section 1.4 and verified by performing a “strings” on the file prog. The phrases shown in section 1.4 were verified to be from the prog file and include mention of “bogowipe”, “wipe the file from the device”, and device file names among other things.



```
[root@localhost floppy_mount]# pwd
/root/GCFA/floppy_mount
[root@localhost floppy_mount]# ls -l prog
-rwxr-xr-x    1 502      502        487476 Jul 14 10:24 prog
[root@localhost floppy_mount]# ls -l --time=ctime prog
-rwxr-xr-x    1 502      502        487476 Jul 16 02:05 prog
[root@localhost floppy_mount]# ls -l --time=atime prog
-rwxr-xr-x    1 502      502        487476 Jul 16 02:12 prog
[root@localhost floppy_mount]# md5sum prog
7b80d9aff486c6aa6aa3efa63cc56880  prog
[root@localhost floppy_mount]#
```

**Figure 1-3: Screen Shot of information for prog File**

The “file “ command identifies the details of an executable. Running the file command on the “prog” executable returns the following.

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

Thus the prog program is a Linux executable. A statically linked executable means that all libraries that the program needs are embedded within the executable and no linking to external libraries on the system is required. The ldd command cannot be used to list the libraries used by a statically linked executable. A stripped executable means that symbol information has been removed from the object files. To the forensics investigator, this basically means there is less information to be gleaned from looking at the executable file.

Looking at the prog executable with the objdump command shows object information about the executable. Perhaps the most interesting output is from the object file headers.

```
# objdump –f prog

prog:    file format elf32-i386
architecture: i386, flags 0x00000102:
EXEC_P, D_PAGED
Start address 0x080480e0
```

This reinforces the fact that prog is an executable for the Intel architecture. The start address is consistent with the normal start addresses of Linux programs, 0x0804.

The readelf command looks as the ELF (Executable and Linkable Format) information in the executable file. In this case, readelf does not return anything that seems to be obviously abnormal. Following is the output of running “readelf -h” on the prog executable file.

```
   # readelf -h prog

   ELF Header:
     Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
     Class:                             ELF32
     Data:                              2's complement, little endian
     Version:                           1 (current)
     OS/ABI:                            UNIX - System V
     ABI Version:                       0
     Type:                              EXEC (Executable file)
     Machine:                           Intel 80386
     Version:                           0x1
     Entry point address:               0x80480e0
     Start of program headers:          52 (bytes into file)
     Start of section headers:          486796 (bytes into file)
     Flags:                             0x0
     Size of this header:               52 (bytes)
     Size of program headers:           32 (bytes)
     Number of program headers:         3
     Size of section headers:           40 (bytes)
     Number of section headers:         17
     Section header string table index: 16
```

## 1.8 Execute the "prog" Executable

At this point, about as much information has been gathered about the "prog" executable as possible without actually executing the program.  Now prog will be executed and monitored to learn more about what it does.

Two tools will be used to monitor the prog executable, tcpdump and strace.  Tcpdump monitors and records network activity and strace records all system call activity of a program.

The tcpdump program was started as the root user with the following command.

```
   # tcpdump -s 0 -w tcpdump.out
```

The "-s 0" option instructs tcpdump to capture the entire contents of packets instead of just header information.  The "-w tcpdump.out" option tells tcpdump to write its output to the file tcpdump.out.  Tcpdump monitors the primary network interface, which is eth0 on the test virtual Linux machine.  The IP address of the virtual machine, as assigned by the VMware server, is 192.168.126.129.  The IP address of the virtual interface on the VMware host OS, Windows XP, is 192.168.126.1.  These are the only two active IP addresses on the virtual network.

The program was then executed as the regular system user "test" and traced with the strace utility.  The "$" in the shell output below represents the shell prompt of the test account. It was decided to first execute the program as an unprivileged user to minimize any problems it may cause. The test user has the user id of 100, a default group of 100 (users), a home directory of /home/test, and a login shell of /bin/bash. From the root login, the test account was accessed with the “su - test” command, which switches user id to the test account and configures the standard login settings for the test account.

```
# su – test
cd /root/GCFA/floppy_mount
$ strace -f -o ../strace.out ./prog
```

The “-f” option instructs strace to trace any child processes spawned by the executable. The “-o strace.out” option causes strace to store its output in the file strace.out. The standard out and standard in file descriptors for the program will not be affected.

The executable returns the following.

```
$ strace -f -o ../strace.out ./prog
no filename. Try ‘--help’ for help.
```

The prog program requires a filename argument, but is very nice to provide instructions on how to get help. Looking at the strace.out file shows no unusual activity other than the program executing and printing the stated output to the terminal. Tcpdump showed no network activity during the run. The output from strace was as follows.

```
5389  execve("./prog", ["./prog"], [/* 21 vars */]) = 0
5389  fcntl64(0, F_GETFD)               = 0
5389  fcntl64(1, F_GETFD)               = 0
5389  fcntl64(2, F_GETFD)               = 0
5389  uname({sys="Linux", node="localhost.localdomain", ...}) = 0
5389  geteuid32()                       = 100
5389  getuid32()                        = 100
5389  getegid32()                       = 100
5389  getgid32()                        = 100
5389  brk(0)                            = 0x80bedec
5389  brk(0x80bee0c)                    = 0x80bee0c
5389  brk(0x80bf000)                     = 0x80bf000
5389  brk(0x80c0000)                     = 0x80c0000
5389  write(2, "no filename. try \'--help\' for he"..., 36) = 36
5389  _exit(2)                          = ?
```

The first column of the output is the process id of the “prog” process. The output shows the program being executed (the execve system call) and obtaining the name of the current system (the uname system call). The uname call seems to be performed by all commands executed on the system, so it is probably a standard operation that the system does when starting any program. The last thing the program does before exiting is to write the help message out to the screen. The “2” just inside of the parenthesis for the write call is the file descriptor, which is the I/O channel that the system will perform the write action on. In this case, file descriptor 2 is standard error. File descriptor 0 is standard in, and file descriptor 1 is standard out.

The program was executed again, but the --help option was specified to see what is reported as the proper usage of the program. Hopefully this will give a better clue as to exactly what it does. The following listing shows what the program returned.

```
$ strace -f -o ../strace1.out ./prog --help
prog:1.0.20 (07/15/03) newt
Usage: prog [OPTION]... [<target-filename>]
use block-list knowledge to perform special operations on files

--doc VALUE
  where VALUE is one of:
  version  display version and exit
  help  display options and exit
  man  generate man page and exit
  sgml  generate SGML invocation info
--mode VALUE
  where VALUE is one of:
  m  list sector numbers
  c  extract a copy from the raw device
  s  display data
  p  place data
  w  wipe
  chk  test (returns 0 if exist)
  sb  print number of bytes available
  wipe  wipe the file from the raw device
  frag  display fragmentation information for the file
  checkfrag  test for fragmentation (returns 0 if file is fragmented)
--outfile <filename> write output to ...
--label                 useless bogus option
--name                  useless bogus option
--verbose                    be verbose
--log-thresh <none | fatal | error | info | branch | progress | entryexit>
  logging threshold ...
--target <filename> operate on ...
```

From the output above, prog is described as “use block-list knowledge to perform special operations on files”. Searching for this phrase using the Google Internet search engine found the Web site http://lwn.net/2000/0420/announce.php3, which listed the program bmap with the exact description contained in the “prog” program. The site lists bmap as version 1.0.17.

Searching further with Google using the keywords “bmap wipe” (with wipe being one of the options from the bmap help output) found the Web page http://www.linuxsecurity.com/feature_stories/data-hiding-forensics.html. This is a linuxsecurity.com feature article by Anton Chuvakin, Ph. D. dated 3/10/2002 11:28 and titled “Linux Data Hiding and Recovery”. The article presents the following discussion of the bmap tool.

> The obscure tool bmap exists to jam data in slack space, take it out and also wipe the slack space, if needed. Some of the examples follow:
>
> ```
> # echo "evil data is here" | bmap --mode putslack /etc/passwd
> ```
>
> puts the data in slack space produced by /etc/passwd file
>
> ```
> # bmap --mode slack /etc/passwd
> getting from block 887048
> file size was: 9428
> slack size: 2860
> block size: 4096
> evil data is here
> ```
>
> shows the data:
>
> ```
> # bmap --mode wipeslack /etc/passwd
> ```
>
> cleans the slack space.
>
> Hiding data in slack space can be used to store secrets, plant evidence (forensics software will find it, but the suspect probably will not) and maybe hide tools from integrity checkers (if automated splitting of the larger file into slack-sized chunks is implemented).

The “prog” program would now seem to really be the “bmap” program, used to hide files in filesystem slack space. A file-hiding tool like this would seem consistent with the activities that Mr. Price has been accused of. With this tool, illicit files could be stored on a computer system and would be invisible to anyone unless they used the bmap tool or picked through the raw image of the filesystem.

One difference between Anton Chuvakin’s article and the “prog” executable is that the prog executable does not want the entire word as an option, only the first letter. For example, the article says to show the data, use “--mode slack”, but the help page for prog specifies “--mode s”.

Also, the link to bmap in the article points to an ftp site with the bmap executable and source code available. This Web site is ftp://ftp.scyld.com/pub/forensic_computing/bmap.

An attempt was then made to execute prog on a file to show data (--mode s) on a file. The file /var/tmp/testfile was created by the test account containing the one line “This is a test file for testing the use of slack space for storing data.” The file’s attributes are as follows.

```
  -rw-r--r--  1 test  users  73 Aug  7 18:14 /var/tmp/testfile
```

The program was executed as the test user with the following results.

```
% strace -f -o ../strace_s.out ./prog --mode s /var/tmp/testfile
unable to open raw device /dev/sda2
unable to raw open /var/tmp/testfile
```

The output of strace shows a failed open of the raw disk device file /dev/sda2.

```
open("/dev/sda2", O_RDONLYIO_LARGEFILE) = -1 EACCESS (Permission denied)
```

*This means that the program must be run as the root user so that the raw disk device file can be accessed.*

The program was run again, but this time as the root user.

```
# strace –f –o ../strace_s1.out ./prog –mode s /var/tmp/testfile
getting from block 629816
file size was: 73
slack size: 4023
block size: 4096
```

The program reports the file size (the actual size of the file's data, 73 bytes) and the filesystem block size (4096 bytes).  The filesystem block size is the minimum size of disk space that the filesystem can allocate, even if a file is smaller than 4096 bytes. The slack space is the difference between the filesystem block size and the actual file size.  Here, there is 4023 byes of slack space in which bmap could hide data without disturbing the file that's occupying the first 73 byes of the block.

The output from strace stored in the file strace_s1.out shows no suspicious activity beyond access the /var/tmp/testfile file and the /dev/sda2 device file.  The tcpdump output shows no network activity associated with running bmap with the "--mode s" option.

Next, the program was used to actually store data in the slack space of the testfile file.

```
# echo "HIDDEN DATA" | strace –f –o ../strace_p.out ./prog --mode p
 /var/tmp/testfile
stuffing block 629816
file size was: 73
slack size: 4023
block size: 4096
```

The strace output stored in the file strace_p.out shows no file activity other than to the /var/tmp/testfile file and the /dev/sda2 device file. This portion of the strace output shows the data actually being written to the file.

```
1937 open("/dev/sda2", O_WRONLY|O_LARGEFILE) = 4
1937  _llseek(4, 2579726409, [2579726409], SEEK_SET) = 0
1937  read(0, "HIDDEN DATA\n", 4023)    = 12
1937  write(4, "HIDDEN DATA\n", 12)     = 12
```

The "open" call opens the raw disk device /dev/sda2, on which the /var/tmp/testfile resides. File descriptor 4 is assigned for the write access to /dev/sda2. The _llseek call applied to file descriptor 4 is assumed to be skipping over the 73 bytes of real file data to position for writing the hidden data in the slack space. The read call reads in from standard input the text from the "echo" command. The last line shown writes the data out to file descriptor 4, which is the raw disk device. The data is written just after the real file data.

The rest of the strace output shows access only to the files /dev/sda2, /var/tmp/testfile, and read access to a few standard system libraries. There was no network activity while the file was executed.

To verify that the data was written, bmap's "slack" mode was used to print any hidden data in the file.

```
# ./prog --mode s /var/tmp/testfile
getting from block 629816
file size was: 73
slack size: 4023
block size: 4096
HIDDEN DATA
```

This verifies that indeed hidden data was written into the file's slack space.

Here is the file listing of /var/tmp/testfile after placing the hidden data into the file.

```
-rw-r--r--  1 test  users  73 Aug  7 18:14 /var/tmp/testfile
```

The file's listing has not changed at all from before the data was hidden in the file. In particular, the file's size and modification time were not changes. There is no outward indication that data was hidden in the file's slack space.

Looking at testfile's inode change time and last access times shows they have not changed since the file was created.

```
# ls -l --time=ctime /var/tmp/testfile
-rw-r—r--  1 test  users  73 Aug  7 18:14 /var/tmp/testfile
# ls -l --time=atime /var/tmp/testfile
-rw-r—r--  1 test  users  73 Aug  7 18:14 /var/tmp/testfile
```

Since the bmap program works by access the raw disk device directly and bypassing the filesystem layer, no filesystem metadata is changed when data is written or read from the slack space of a file. No command that shows the attributes of the filesystem or files within the filesystem would be able to indicate that data is stored in filesystem slack space.

To obtain the version of bmap being used, the following command was executed.

```
# ./prog --doc version
prog:1.0.20 (07/15/03) newt
```

So this is version 1.0.20 of bmap.

Prog (bmap) was executed with the "--mode s" option on all files contained in the floppy image to see if any contained any "hidden" information. No files did. One file, "Sound-HOWTO-html.tar.gz" returned binary garbage as slack data, but all other files returned nothing as slack data.

## 1.9 Examine and Compile bmap Source Code

The source code for the bmap executable was downloaded from the FTP site given in the linuxsecurity.com article cited in section 1.8, ftp://ftp.scyld.com/pub/forensic_computing/bmap. The site contains four versions of bmap, versions 1.0.16, 1.0.17, 1.0.18, and 1.0.20, as well as Red Hat Linux RPM packages. Since it is known that the version of bmap used here is 1.0.20, this is the version the analysis will concentrate on.

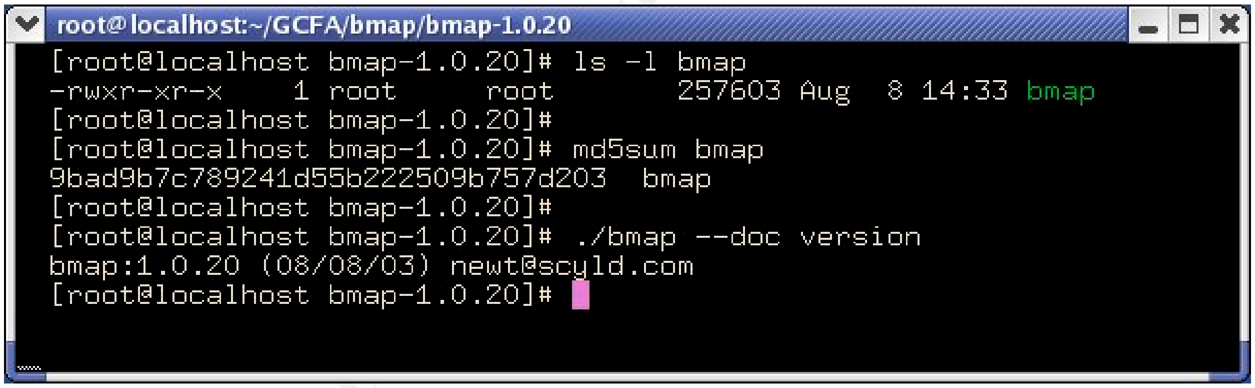
The source code file and its signature file were downloaded onto a computer connected to the Internet, placed on floppy disk, and copied onto the Linux analysis virtual machine. The two files copied were named "bmap-1.0.20.tar.gz" and "bmap-1.0.20.tar.gz.sig". The ".sig" file is a PGP digital signature of the ".gz" file. For reference, the MD5 digest value of the ".gz" file is df716d23d5966826fe6bad9d0a65cdd6.

The source code files were unpacked with the command "tar zxf bmap-1.0.20.tar.gz". The "z" option to tar uncompresses the file automatically, the "X" option extracts the files, and the "f" option specifies the source tar file. The files extracted into a directory called bmap-1.0.20.

According to the README and LICENSE files present in the directory, the bmap program is maintained by Daniel Ridge in support of Scyld Computer Corporation and is protected by the GNU Public License.

Looking at the file Makefile, which contains the instructions for compiling the program, it is seen that the option "binaries" will compile just the binary program and not try to install it on the system. The binary was compiled from the bmap-1.0.20 directory with the command "make binaries" as the user root. This generated two binaries, bmap and bclump. The README file describes bclump as a tool for combining and sorting bmap output.

Running the compiled bmap binary to show its version and to generate the MD5 digest is shown in Figure 1-4.


root@localhost:~/GCFA/bmap/bmap-1.0.20

```
[root@localhost bmap-1.0.20]# ls -l bmap
-rwxr-xr-x    1 root     root       257603 Aug  8 14:33 bmap
[root@localhost bmap-1.0.20]#
[root@localhost bmap-1.0.20]# md5sum bmap
9bad9b7c789241d55b222509b757d203  bmap
[root@localhost bmap-1.0.20]#
[root@localhost bmap-1.0.20]# ./bmap --doc version
bmap:1.0.20 (08/08/03) newt@scyld.com
[root@localhost bmap-1.0.20]# 
```

**Figure 1-4: bmap File Info**

Note that the version reported by bmap is slightly different from the version reported by the "prog" bmap.

"prog" bmap version: 1.0.20 (07/15/03) newt
downloaded bmap version: 1.0.20 (08/08/03) newt@sculd.com

Since the bmap that was downloaded and compiled was compiled on 8/8/03, it would follow that the compile places the current date into the executable file to be reported when the version is queried. This would suggest that the "prog" program was compiled on 07/15/03. Also, the bmap compiled above was configured to compile dynamically and not statically like the "prog" program. This, and the different version string, would explain why the MD5 digest is different between the two programs (7b80d9aff486c6aa6aa3efa63cc56880 for the original, 9bad9b7c789241d55b222509b757d203 for the 08/08/03 version). Perhaps by applying some programming expertise the downloaded bmap source could be forced to compile statically and the version string could be manipulated to match that of the "prog" program. This might make the MD5 digest values match, but in order to keep the investigation moving, this option was not pursued. Instead, the investigation relied on the matching portions of the "strings" output and the functional testing to show that the "prog" program was really bmap.

The strings command was run on each executable and the results were compared. Strings output was generated with the command "strings prog > /var/tmp/strings_prog.txt; strings bmap > /var/tmp/strings_bmap.txt". The results show a lot of difference at the beginning, probably due to the static vs. dynamic compilation issue (the 07/15/03 version starts off with garbage characters indicating binary data, the 08/08/03 version starts of with printable text). However, there is a large area of similarity between the two in the area that seems to contain the text of warning and error messages. The output files generated here were preserved as evidence on the Linux virtual machine and the MD5 digest values are listed below.

MD5 digest of strings_prog.txt: c494534cefbb33104edb4ac27803cvc2
MD5 digest of strings_bmap.txt: 692fa5fc18a29d5899bef323d28bb094

Further search of the Internet found a few other sites that contained the bmap source code. They seem to be mirror sites of each other. Here are the other sites found.

- http://linux4u.jinr.ru/LinuxArchive/Ftp/SCYld/forensic_computing/bmap/bmap-1.0.20.tar.gz.sig
- http://www.scyld.com/pub/forensic_computing/bmap/
- http://ftp.cfu.net/mirrors/garchive.cs.uni.edu/garchive/bmap-1.0.20/

## 1.10 GIAC Required Information

This section is organized to correspond to the specific questions asked in the GCFA practical assignment. References to information in other sections of this document are included where appropriate.

### 1.10.1 Binary Details

The information requested for this section was presented above in section 1.7.

### 1.10.2 Program Description

- **What type of program is it?** Investigation revealed that the "prog" program was in reality the bmap program, which is a Linux compiled executable program. Details are discussed in sections 1.7 and 1.8.
- **What is it used for?** The prog (alias bmap) program's job is to covertly store data in filesystem slack space, take the data back out of slack space, and wipe data from slack space. This is discussed in the linuxsecurity.com article in section 1.8.
- **When was the last time it was used?** The last time the program was used is likely the" last access" time of the program, which was found to be 7/16/03 at 06:12:45, based on the analysis performed using the Autopsy tool in section 1.6.
- **Step-by-step analysis of the program.** This was performed in sections 1.6, 1.7, and 1.8 above.

### 1.10.3 Forensic Details

- **What are the forensic footprints of the program?** The program is a statically compiled stripped executable as discussed in sections 1.7 and 1.8. A statically compiled executable does not link to external libraries, and a stripped executable has the symbol information removed from the object files, reducing the amount of information in the executable. Thus the "prog" program has a very small footprint. When executed, the program accesses only the target file with slack space and the raw disk device on which the target file lives. Following are some ways that the presence of the program can be identified.
    - The MD5 digest value of the program can be generated for each file on a system and compared to the digest value of the known bmap executable. However, it was shown in section 1.9 that some information (namely the date in the version information) can change when the program is compiled, causing the MD5 digest value to change as well.
    - A copy of the bmap program itself can be used to check files on the filesystem for information stored in slack space. The "--mode s" argument to bmap will show if there is information stored in the slack space of a given file. If information is found, it is possible that bmap has been used on the system.
    - Bmap requires root privilege to access the raw disk device. A non-root user cannot run bmap successfully. Strange programs running as the root user could indicate the presence of an unauthorized program.
    - Running the strings program against a file will return the printable text within the file. Bmap has certain known strings present in its binary file. To locate the bmap program, a script might be written to run strings against every file on the system and inspect the output for certain strings matching strings from the known bmap executable.
    - Since bmap only accesses the raw disk device when manipulating slack space, there are no changes to the filesystem or the specified file whose

slack space is being used. Thus filesystem commands are useless in identifying if a file's slack space is populated with data.

- **What other files are used when the program is executed or implemented?** As discussed in the last bullet, the bmap program only accesses the given file whose slack space is to be used, and the raw disk device on which that file lives. Since the bmap program access the raw disk device, the target file with slack space is not accessed as such, only the data areas of its slack space are accessed. This means that there are no changes to the file or the filesystem when bmap is used.
- **How is the filesystem affected by the execution of the program?** As discussed above, the filesystem is not affected by the bmap executable since bmap uses the raw disk device to access the slack space.
- **Does the program use, manipulate, or reference any other system files?** As discussed above, bmap only accesses the target file with slack space and the raw disk device. No system files were touched by bmap in the testing described in section 1.8.
- **Are their any "leads" that could be pulled out of the file for further investigation (e.g. IP address, user information, etc.)?** The file seemed to be a generic version of the bmap program available for download from the Internet from the sites discussed in section 1.9. There were no obvious customizations such as the inclusion of names or IP addresses within the bmap executable. However, as seen in sections 1.4 and 1.5, other files on the floppy image did contain interesting information, such as the Mikemsg.doc file which discussed "advance orders" and "batch of files". With further investigation, the raw disk image and the files it contains may yield more clues. Also, the presence of the netcat package on the floppy image suggests that Mr. Price may have been using netcat to move files over the network. Firewall logs or system logs from Mr. Price's system (assuming the logs were stored on another system or are available on backup tapes) may yield information on network usage and file transfers. Also, if a backup of the system is available, it could be used to recreate Mr. Price's system for further investigation.

### 1.10.4 Program Identification

The requirements of this section are described in section 1.9.

### 1.10.5 Legal Implications

This section briefly discusses the available proof that the "prog" program was actually executed on Mr. Price's system and any legal or policy implications of the program being run or being present on the system.

- **Is there evidence that the "prog" program (bmap) was run on Mr. Price's computer system?** Since Mr. Price wiped the hard disk on his computer system before any evidence could be gathered, there is no direct evidence that the program was executed on that computer. The only evidence that points may imply that the program was executed was the MAC time information obtained from the floppy disk image during the forensic analysis. When a program is executed, the program's access time us updated because the program file must be read to be executed. Accessing a file does not change the file's modification or inode change time, so an access time later than the change or modification time would be consistent with a program file being executed. It would also be consistent with the file being viewed or having the contents listed, such as with the "strings" command. The last access time of the "prog" file from the floppy image was shown in the timeline generated by the Autopsy tool to be July 16 06:12:45 and from the Linux "ls -l –time=atime" as July 16 02:12:45. Note that the time generated from Autopsy was GMT and the time from the analysis system's "ls" command was EST. GMT is four hours ahead of EDT during the summer.

July 16 is assumed to be the date that the floppy was entered into evidence, based on the "160703" portion of the floppy disk's evidence tag number fl-160703-jp1.

The last access time indicates that the "prog" contents of the "prog" file were last accessed (read) on July 16 at 02:12:45 EDT, which is later than both the modification and inode change times. As pointed out above, it's not possible to say with absolute certainty if the file's contents were read as part of the program being executed, or as part of the contents being read by a program such as "strings", or even as part of copying the file to a different location.

So, in summary, there are implications that the file could have been executed on or copied to Mr. Price's computer, but there is no way to tell with complete certainty that the program was executed.

- **How may the use of the program violate the organization's internal policies?** No direct evidence of illegally copied copyrighted material was found on the floppy disk image. The "howto" files for doing multimedia on Linux and the presence of the netcat program may imply an interest on Mr. Price's part in distributing video or music files over the network, but there is no direct evidence that such files actually were copied or distributed.

  However, the presence and use of the file on the employer's system would probably violate internal policy on the personal use of employer computers and networks, and possibly violate rules on installing unauthorized software. For example, the following is an excerpt from the Web site http://www.devzonedevelopment.com/securitypolicytemplate.html and is a sample computer security policy template. Mr. Price's use of bmap may be considered a violation of these two rules.

  **"** Users shall not make unauthorized copies of copyrighted software, except as permitted by law or by the owner of the copyright."

”Users shall not download, install, or run security programs or utilities that could potentially reveal weaknesses in the security of a system. For example, [COMPANY NAME] users shall not run password cracking, key logging, or any other potentially malicious programs on [COMPANY NAME] Systems Division computing systems.”

Another possible rule forbidding personal software might read something like this one from the Web site http://www.mercynet.edu/IT/ethics/modules/CorporateSoftwarePolicy.htm.

“BSA employees shall not download or upload unauthorized software over the Internet or otherwise place software onto company computers.”

### 1.10.6 Interview Questions

This section addresses the scenario, “Assume that you have the opportunity to interview the person who installed and executed the program. List the questions that you could use to prove that the subject was in fact the one who installed it and executed it on the victim system”.

1. Did you write the letter stored in the file Mikemsg.doc, and if so what is meant by the phrases “latest batch of files”, “advance orders”, and “next run”?

   The file Mikemsg.doc and all other files on the floppy image are owned by user id 502, including the prog program file. If it can be established who generated one of the files on the floppy, then the implication is that the same person created all of the files. Of course, there is always the possibility that someone else was using the account or someone had root access and manually changed the ownership of the files to a desired account. Note that root access is required to run the bmap program. This also helps to establish the link between the user and the “copying copyrighted material” charge.

2. Did you download the “HOWTO” files on the floppy, such as the “DVD-Playing-HOWTO-html.tar” file? If so, how does this relate to your job?

   Again, this question can establish a link between the user id 502 and all of the files on the system. It also helps to firm up the link between the user and the “copying copyrighted material” charge.

3. Did you have root access to your computer and what programs have you executed as the root user?

   This question relates to the fact that bmap needs to be run as the root user, so whoever ran it had to have access to the root account.

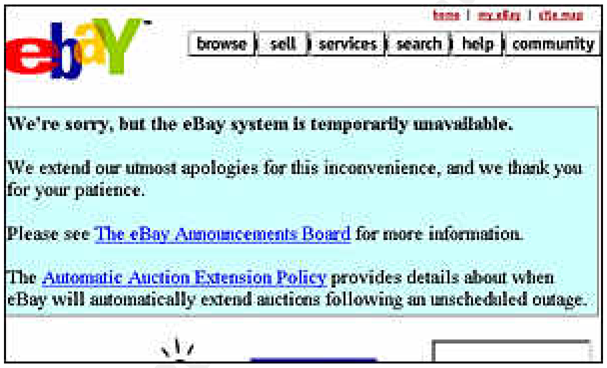
4. The netcat program’s installation package file was found on the floppy disk (filename nc-1.10.16.i386.rpm..rpm). Did you install this program on your

computer, and if so, what did you use it for?

Netcat can be used to transfer files over the network using any chosen network port, and it can also be used to facilitate remote logins or execution of commands.  This relates to the charge of copying copyrighted material.

5. There are three graphics files on the floppy image.  The files sect-num.gif and sectors.gif are diagrams of the track-sector layout of a disk.  The file ebay300.jpg seems to be a shot of a browser window pointed to eBay.  What do you know about these files?  Did you access ebay from your work computer?

   The ebay300.jpg image is a screen shot of a browser window apparently pointed to eBay with a message indicating eBay was temporarily down.  It would be instructive to learn why someone would store this image and what he or she was using eBay for.  Figure 1-5 shows the eBay image.



We're sorry, but the eBay system is temporarily unavailable.

We extend our utmost apologies for this inconvenience, and we thank you for your patience.

Please see The eBay Announcements Board for more information.

The Automatic Auction Extension Policy provides details about when eBay will automatically extend auctions following an unscheduled outage.

**Figure 1-5: ebay300.jpg**

6. What do you know about the file named "prog" on the floppy image?  What was it used for?

   After establishing a relationship between the user and the other files, the user should then have to explain the presence of the prog file and what it was used for.

### 1.10.7 Case Information

This section relies on the detailed analysis described earlier to answer several key questions.

- **What advice can you provide to the System Administrators to help them detect whether this binary is in use, or has been used on other machines?** The potential means that could be used to identify this program were discussed in section 1.10.3. The difficulty with detecting the presence of data in slack space is that there is no way to access slack space using standard filesystem commands. When data is stored in slack space, there is no change in file size or filesystem characteristics. The only way to identify data in slack space would be to use a tool like bmap that accesses the slack space via the raw disk device and dump the contents of the slack space. Perhaps the best thing for system administrators to do is to look for signs of the bmap program file or its variations (prog) on machines or look for these program names in system accounting logs.

  Also keep in mind that slack space is associated with an existing file. If a file containing slack space were to be deleted or changed in size, the slack space would probably be overwritten. Thus, it would be most desirable to use the slack space of files that are least likely to change or be deleted.

  Since slack space can only be access via the raw disk device, root privileges are required. System administrators can pay particular attention to system accounting logs or process listings to identify suspicious activity performed by the root user.

- **What, if anything, did you find that would lead you to believe that John Price was using the organizations computing resources to distribute copyrighted material?** Several suspicious pieces of information were found that could connect Mr. Price with the charge of distributing copyrighted material. The following provides a summary of the information with references to the details elsewhere in this document.
  - The presence of the “prog” program (which was really bmap) implies that files were being hidden on the system. There is no use for bmap other than to covertly store files on a computer system. The use of bmap is consistent with someone who was trying to hide files from system administrators or investigators.

- The presence of the netcat program implies that some sort of network communication was being used outside of that provided by the default system configuration. Netcat would be a good tool to use to covertly transfer files over the network since it can easily use any port numbers desired to potentially bypass firewall or filtering protection on the network. It also can be used to allow the local machine to open a "server" port for other machines to connect to and download files or execute commands. Depending on what Mr. Price's job was, netcat was probably not a tool he needed for official work purposes.
- The Mikemsg.doc file containing the suspicious message with the phrases "latest batch of files", "advance orders", and "next run" was found on the floppy image as described in section 1.5. This could imply a relationship with someone named John with whom Mr. Price was trading the illegal material and taking orders from other parties for more illegal material.
- The presence of the Linux multimedia HOWTO files and the information relating to the LiVid project indicate an interest on the part of Mr. Price in processing or manipulating multimedia files such as DVD video or MP3 music. This type of information is frequently copyrighted.
- The fact that Mr. Price wiped the hard drive on his computer implies that he had something to hide.

### 1.10.8 Additional Information

This section provides a summary of references used in the investigation.

- **Bmap**
    - Chuvakin, Anton, Ph. D. "Linux Data Hiding and Recovery." 10 March 2002. URL: http://www.linuxsecurity.com/feature_stories/data-hiding-forensics.html (accessed 8/2/2003)
    - Source code download site. URL: ftp://ftp.scyld.com/pub/forensic_computing/bmap
- **Linux Multimedia**
    - Xmms Red Hat package download site. URL: http://havardk.xmms.org/dist/xmms-1.2.7-rh8-rh9-rpm/ (accessed 8/2/03)
    - Linux Video Project Web site. URL: http://www.linuxvideo.org/ (accessed 8/3/03)
- **Computer Use Policy Examples**
    - Devzone Web. "Security Policy Template." URL: http://www.devzonedevelopment.com/securitypolicytemplate.html (accessed 8/8/03)
    - "BSA Corporate Software Policy regarding the use of personal computer software." URL: http://www.mercynet.edu/IT/ethics/modules/CorporateSoftwarePolicy.htm (accessed 8/8/03)

## Part 2 – Option2: Perform Forensic Tool Validation

In part 2 of this paper, the Linux ext2 file system debugger, debugfs, will be tested and validated as a “quick and dirty” means to recover deleted files from a Linux filesystem. The testing will validate if debugfs can recover files from a Linux ext2 filesystem in a verifiable and repeatable manner so that the use of the tool can be supported or refuted in a court of law. The testing will also explore the limitations of debugfs in restoring deleted files.

### 2.1 Scope

The debugfs program is a utility for debugging ext2 filesystems. The utility is supplied as part of the e2fsprogs package, which was part of the default installation of the Red Hat 8.0 test system. The e2fsprogs package includes several tools and libraries that support the debugging and low-level manipulation of ext2 filesystems. The home page for the ext2progs utilities, http://sourceforge.net/projects/e2fsprogs, describes the Ext2 Filesystem utilities as containing, “all of the standard utilities for creating, fixing, configuring, and debugging ext2 filesystems.” Of interest here are the functions available in the debugfs utility to display and restore deleted files.

The debugfs utility has many functions that manipulate an ext2 filesystem, including viewing and changing inode information and filesystem parameters. The options available can be displayed by using the “help” command within debugfs. The debugfs functions involved with restoring deleted files include the following.

- **lsdel:** List deleted inodes
- **dump:** Dump an inode out to a file
- **stat:** Show inode information
- **undelete:** Undelete file

The ability to restore deleted files is dependent on the inode and data block information from the deleted file having not been overwritten, and therefore the chances for a successful restoration vary greatly from case to case. The testing performed here will evaluate debugfs’s ability to restore deleted files under several different conditions using the lsdel function to list deleted inodes, and the dump function to move the contents of the deleted inode (the data blocks) to another file.

The undelete function will not be used in testing since its job is to modify the deleted inode itself in order to mark the inode as not deleted, thus modifying the filesystem. Using undelete might also leave the filesystem in an inconsistent state. Since in a forensic investigation it is critical not to disturb evidence, the undelete function should not be the first choice in recovering a deleted file. The dump function only requires read access to the filesystem, and is thus preferable for use in a forensic investigation. Successful restorations will be judged by comparing the restored file’s MD5 digest value with that of the original file before deletion.  In the case of partial restorations, the portion of the file that could be restored will be compared to the original file's contents for validation.

The conditions under which the debugfs tool will be tested include the following.

- Small file deleted (less than 10KB in size), no other filesystem activity
- Large file deleted (approx. 3MB in size), no other filesystem activity
- Small file deleted (less than 10KB in size), filesystem activity after deletion
- Large file deleted (approx. 3MB is size), filesystem activity after deletion
- Multiple files deleted (varying sizes), no other filesystem activity
- Multiple files deleted (varying sizes), filesystem activity after deletion
- Multiple files deleted (varying sizes), one big file then takes all data blocks

## 2.2 Tool Description

This section discusses the details and background of the debugfs tool.

### 2.2.1 Debugfs Version

As described above, the debugfs tool is designed to perform low-level manipulation of an ext2 filesystem.  As described in the debugfs man page on the Linux test platform,

> The debugfs program is an interactive filesystem debugger.  It can be used to examine and change the state of an ext2 file system.

A version of the debugfs man page can be found online at http://www.die.net/doc/linux/man/man8/debugfs.8.html.  Note that out of the four debugfs commands to be used (lsdel, dump, stat, and undelete) only dump and stat are mentioned in the man page.  The lsdel and undelete commands only seem to be documented on the system in the help menu within debugfs itself.  Review of the references in the "Ext2 Filesystem" section of the References portion of this document were necessary to gain an understanding of how all of the commands can be used together to restored deleted files.

The version of debugfs used for evaluation on the Red Hat 8.0 test machine was obtained with the command "debugfs -V" and was as follows.

```
debugfs 1.33 (21-Apr-2003)
Using EXT2FS Library version 1.33
```

The debugfs tool version 1.27 was present as part of the default Red Hat 8.0 installation and was located in the /sbin directory.  To have a more current version of debugfs, for a more interesting test in case the file restoration capabilities had been enhanced, version1.33 of the e2fsprogs package was downloaded from http://rpmfind.net as the file e2fsprogs-1.33-1.i386.rpm. The updated version was installed on the test system with the command "rpm -Fvh e2fsprogs-1.33-1.i386.rpm". This updates all of the e2fsprogs tools and libraries, including debugfs, to version 1.33. After the upgrade, the debugfs binary file had the following MD5 digest value as reported by the md5sum command.

```
# md5sum /sbin/debugfs
ecc2ed1d0f9517f619846834290b5238 /sbin/debugfs
```

The general issues involved with restoring deleted files are described well by Aaron Crane in his paper "Linux Ext2fs Undeletion mini-HOWTO", v1.3, 2 February 1999, available at http://pobox.com/~aaronc/tech/e2-undel/howto.txt. Aaron Crane's paper describes the use of debugfs and his tool, fsgrab, in recovering files. For the purposes of this paper, the use of debugfs is of primary interest.

### 2.2.2 Inode Basics

Some discussion of the role played by inodes in the ext2 filesystem is useful to help understand what is involved with restoring a deleted file.

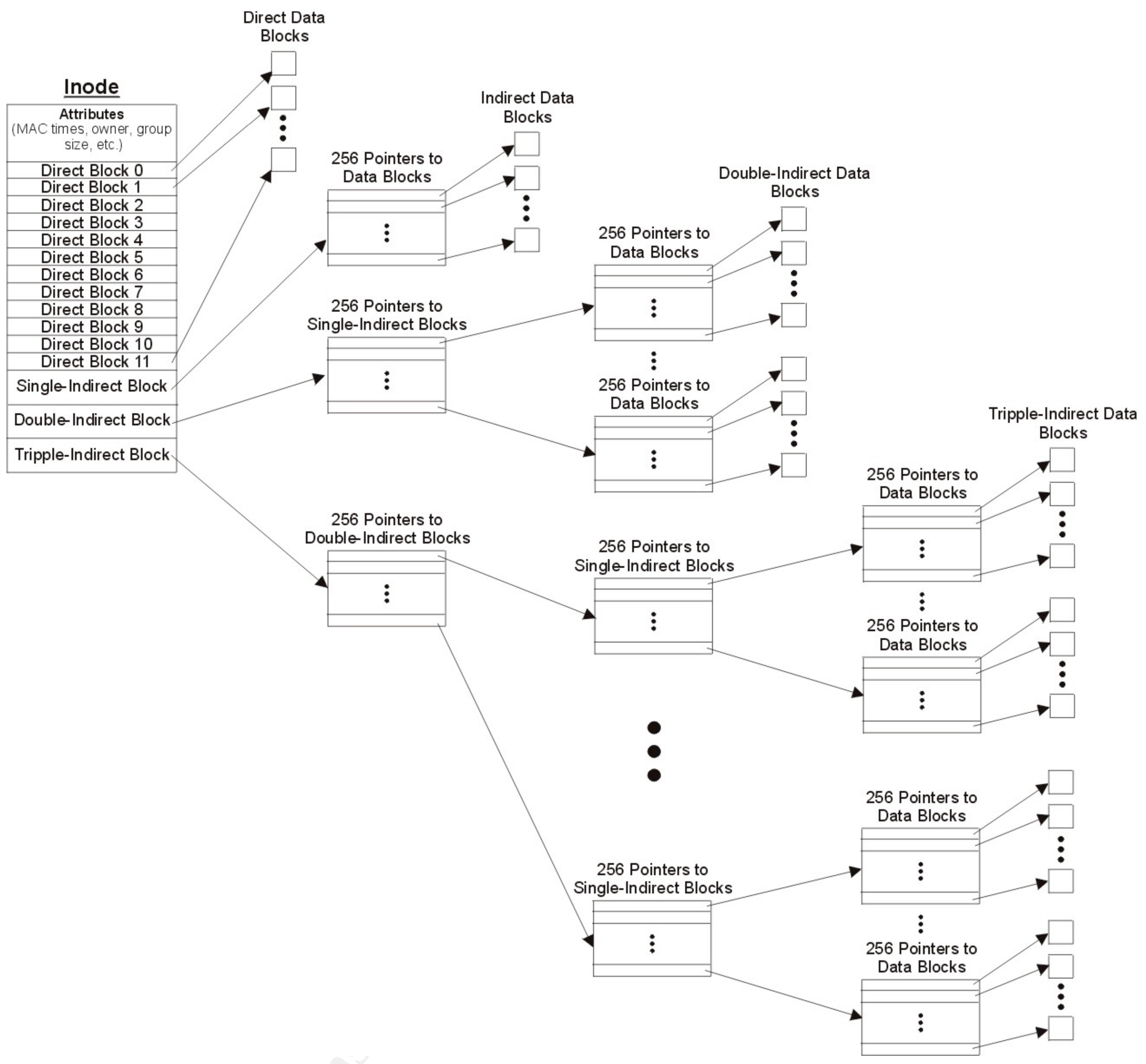
As with most filesystems, the Linux ext2 filesystem stores data in blocks on disk. The blocks are organized into files and directories by meta-data constructs called inodes. An inode contains the attributes of a file such as permissions, MAC (Modification, Access, Change) times, owner, group, and size. The file's inode also contains a list of the data blocks associated with the file. The only major piece of information that the inode does not contain is the file's name. The file's name is contained only in a directory, which is itself defined by an inode.

The ability to undelete or extract deleted files in the Linux ext2 filesystem relies on the fact that Linux does not clear the contents of an inode when the file or directory it points to is deleted. When deleting a file, Linux simply changes the inode's "deletion time" attribute from zero to the time of deletion, and it changes the "link count" attribute (which defines how many references there are to the file from elsewhere in the filesystem) to zero. All other information, including the data block list, is left in tact. At this point, the inode and data blocks are free to be re-used and overwritten.

Once the inode information or the data blocks are reallocated and overwritten, there is no way to recover the original file. However, the file can be recovered if the inode information and data blocks have not been overwritten. The "lsdel" command in debugfs lists the inodes in a filesystem that have the deletion attributes set as described above. Since the inode does not contain the file's name, the proper file can only be identified by information stored in the inode, such as MAC time, owner, group, permissions, and size. Thus, if many files have been deleted, some knowledge of the attributes of the file that was deleted must be known in order to figure out which deleted inode is likely the one to be restored.

The names of delete file names can be obtained from a directory by using the debugfs "ls -d" command on the directory's inode. This will display the names of deleted files (preceded with "<0>"), but there is no indication of what inode number the associated inode used.

The last piece of important information about inodes is how the data blocks are stored. An inode has four levels of specifying access to data blocks; Direct, Single-Indirect, Double-Indirect, and Triple-Indirect. As shown in figure 2-1, direct data blocks are pointed to directly from the inode. The Single-Indirect pointer in the inode points to a block with 256 pointers to data blocks. A Double-Indirect pointer points to 256 Single-Indirect pointer blocks. And finally, a Triple-Indirect pointer points to 256 Double-Indirect blocks.



Direct Data Blocks
Inode
Attributes (MAC times, owner, group size, etc.)
Direct Block 0
Direct Block 1
Direct Block 2
Direct Block 3
Direct Block 4
Direct Block 5
Direct Block 6
Direct Block 7
Direct Block 8
Direct Block 9
Direct Block 10
Direct Block 11
Single-Indirect Block
Double-Indirect Block
Tripple-Indirect Block
Indirect Data Blocks
256 Pointers to Data Blocks
256 Pointers to Single-Indirect Blocks
Double-Indirect Data Blocks
256 Pointers to Data Blocks
256 Pointers to Data Blocks
Tripple-Indirect Data Blocks
256 Pointers to Double-Indirect Blocks
256 Pointers to Single-Indirect Blocks
256 Pointers to Data Blocks
256 Pointers to Data Blocks
256 Pointers to Single-Indirect Blocks
256 Pointers to Data Blocks
256 Pointers to Data Blocks

**Figure 2-1: Inode Layout**

As might be inferred from Figure 2-1, it is easier to recover data from the Direct Data Blocks since the inode contains pointers directly to the blocks. To get to the data in the Single, Double, and Triple-Indirect blocks, the layers of indirection must be traversed, which can be quite complicated and time consuming in practice. Since the default block size is usually 1024 bytes (1KB), files of size 12KB or smaller are stored in data blocks pointed to by the Direct Block pointers within the inode. Files larger than 12KB require the use of the indirect pointers. Thus the 12KB boundary is useful for testing how reliable file restoration is. If a tool used for file restoration can't follow the indirect pointers, it would not be able to effectively recover files larger than 12KB.

A more in-depth explanation of inodes and their role in file recovery is presented in (Crane) and (Card; Ts'o; Tweedie).

The ability to recover deleted files is of great interest to the forensic analyst. Not only might a user or intruder delete files that may provide evidence of wrongdoing, but files created and deleted as part of tasks like compiling programs can give important clues as to what activity was taking place on a system and what programs were used on the system. For example, an intruder might download and compile a program to exploit a vulnerability and gain access as the root user. The intruder may compile the program, run the executable, and then delete the source and object code to hide the evidence. If the source code and object files can be extracted and examined, the investigator may be able to figure out more quickly what has happened on the computer system and what to do about it.

### 2.2.3 Debugfs Details

The debugfs program is part of the e2fsprogs utilities, which include various tools to manipulate and debug an ext2 filesystem. The Web site titled "Ext2fs Home Page" at the URL http://e2fsprogs.sourceforge.net/ext2.html contains links to the e2fsprogs tools and other information about the e2fs filesystem. According to the Ext2fs Home Page, the e2fsprogs tools consist of the individual tools "e2fsck, mke2fs, debugfs, dumpe2fs, tune2fs, and most of the core ext2fs filesystem utilities". The e2fsprogs tools are installed by default with Red Hat Linux 8.0, which will be used on the test platform. The program is dynamically compiled and requires access to several dynamic libraries, as shown below from the ldd command run on the debugfs executable.

```
# ldd /sbin/debugfs
libext2fs.so.2 => /lib/libext2fs.so.2
libe2p.so.2 => /lib/libe2p.so.2
libss.so.2 => /lib/libss.so.2
libcom_err.so.2 => /lib/libcom_err.so.2
libblkid.so.1 => /lib/libblkid.so.1
libuuid.so.1 => /lib/libuuid.so.1
libdl.so.2 => /lib/libdl.so.2
libc.so.6 => /lib/libc.so.6
/lib/ld-linux.so.2 => /lib/ld-linux.so.2
```

Debugfs requires at least read access to the raw disk device on which the deleted files were stored, and write access if any filesystem or inode properties are to be changed. The undelete command within debugfs requires write access to the filesystem being manipulated. This implies that debugfs must be run as the root user, or as a special user with access to the raw disk device.

On a potentially compromised system, no commands or libraries can be completely trusted. An attacker or attack tool may have altered commands or libraries to not report things that reveal the attacker's activities. For example, an attacker might install a Trojan version of the "ls" program that does not list specific file names used by the attacker or a Trojan version of the "ps" program that does not show the processes being used by the attacker. Thus, to ensure that evidence in an investigation is gathered in a sound manner, the tools used in the investigation must not have to rely on system commands or libraries unless the commands and libraries are shown to be unaltered.

One way to ensure that the analysis tools used by the investigator are not relying on compromised programs or libraries is to compile the tools statically. A statically compiled binary program contains all of the libraries it needs within its program and does not rely on external libraries on the system. Another way to ensure that libraries are safe is to place all of the necessary libraries on removable media and make sure that the libraries on the removable media are used before the system libraries. For example, if a program wants to use the dynamic library libc.so.6 which is usually found in the directory /lib, the libc.so.6 library file can be copied from /lib on a known-good machine onto removable media such as a CDROM along with the executable that requires it. To run the program on a potentially compromised system, the CDROM is mounted and the environment variable "LD_LIBRARY_PATH" is set to look at the CDROM mount point first before looking in /lib on the local system. For example, if the CDROM were mounted at /mnt/cdrom, then this command would point executables to the libraries on the CDROM. This assumes that a bash shell is being run.

```
# set LD_LIBRARY_PATH=/mnt/cdrom; export LD_LIBRARY_PATH
```

Then programs can be executed from the CDROM using the trusted libraries.

The debugfs executable on the test machine, as discussed above, is dynamically linked. This means that it requires external libraries in order to execute. In order to place debugfs on removable media in a self-contained form, either the libraries on which debugfs depends would have to be placed on the removable media along with the executable and the LD_LIBRARY_PATH variable set as discussed above, or a statically compiled version of debugfs would have to be available.

As for the latter case, an Internet search was done to see if there is a static version of debugfs available. The closest thing to a static version of debugfs that was found was a package of static libraries for the e2fsprogrogs package. This was found at the Web site http://rpmfind.net. The package name for the static libraries was e2fsprogs-static-1.33-1. This package depends on the e2fsprogs development package, e2fsprogs-devel-1.33-1. An actual static version of debugfs is not included in these packages. Based on the information available, these packages seem to be just the static libraries needed to compile static version of the e2fsprogs tools. It was assumed that if the source code for debugfs were found, that code could be compiled statically using the e2fsprogs-static libraries given the right compiler options. This was not pursued in order to maximize the effort of testing debugfs, but it seems possible to do if a static version of debugfs is desired.

It is possible to place the dynamically compiled debugfs executable on a CDROM or floppy disk, along with the dynamic libraries on which it depends. Then the LD_LIBRARY_PATH variable could be set to point to the removable media mount point for debugfs to find its libraries. This was tested using a CDROM disk and it worked for the all but one of the libraries used by debugfs. The ld-linux.so.2 library seems to be hard-coded by the linker to point to the absolute path /lib/ld-linux.so.2. Even if the LD_LIBRARY_PATH variable is changed to point to the CDROM mount point /mnt/cdrom as described above, the program still insists on looking for this library at the absolute path /lib/ld-linux.so.2 on the local system. Some searching was done on the Internet on this issue, but no easy way was found to make this program not use the absolute path to the ld-linux.so.2 library.

Thus, to use debugfs in an evidentiary sound way, the ld-linux.so.2 library would have to be trusted on the local system, or perhaps a link pointing to ld-linux.so.2 on the CDROM could be created in /lib on the system being examined (/lib/ld-linux.so.2 is really a link to /lib/ld-2.3.2.so) to force the copy of the library on the CDROM to be used. However, in a forensic investigation it is most desirable to minimize any changes made to a system under investigation.

For the purposes of this test, the libraries and programs on the test Linux platform are from a fresh installation of Red Hat 8.0 downloaded from http://www.redhat.com and are considered to be trusted.

## 2.3 Test Apparatus

The platform is the same used in Part 1 to analyze the floppy image and the unknown binary. The system is a Dell Inspiron 5100 Laptop computer running the Windows XP SP1 operating system with the latest Microsoft security fixes as of 8/1/03 installed from http://windowsupdate.microsoft.com.

The VMware Workstation 3.2.0 application is installed to provide a virtual Linux machine and a virtual network within which the binary can be safely analyzed. VMware emulates the hardware and BIOS (Basic Input Output System) of a computer for each virtual computer that it runs. VMware also emulates a network within the host computer (the Windows XP system in this case) so the different virtual machines can communicate with each other. This is an ideal situation for forensics analysis because the virtual machines and network are not exposed to external systems where the potentially malicious software being run might do harm. And since each virtual machine is simply a set of files on the host machine's file system, it is very easy to make copies of a virtual machine at different points in time and to easily revert back to one of those copies.

The virtual machine used for this analysis was a fresh copy of the one used in Part 1. The virtual machine was running the Red Hat 8 Linux operating system (kernel 2.4.20-18.8, gcc version 3.2 20020903) with the latest security updates as of 8/1/03 applied. The updates were downloaded from the Red Hat mirror site ftp://ftp.dulug.duke.edu/pub/redhat/linux/updates/8.0/en/os/i386/, written to CDROM, and copied into the directory /var/tmp/updates on the virtual Linux machine. The updates were installed by executing the following commands from the Linux shell (command line) within the VMware virtual Linux machine.

```
# cd /var/tmp/updates
# rpm -Fv *
```

Note that it is important to keep the host operating system (Windows XP in this case) updated with the latest security fixes and good security practices to minimize the risk of the host operating system being affected by the testing of suspicious programs within the VMware virtual machines. This is a possibility because VMware creates a virtual network interface on the host operating system that interfaces with the VMware virtual network and the virtual machines.

More information on the VMware product, including full documentation, can be found at the VMware Web site http://www.vmware.com.

The filesystem for testing will be an ext2 filesystem existing within a file on hard disk and mounted using the loopback mount option. The loopback option allows an image of a filesystem stored within a single file on hard disk to be mounted and manipulated just like any other filesystem. Performing testing from a relatively small loopback filesystem will allow the filesystem to by easily filled, guaranteeing that blocks of deleted files will be overwritten for the portion of the test that requires this. Also, it is easy to initialize a fresh filesystem between tests to ensure that each portion of the test is being run on a clean filesystem so deleted files from one test do not show up for the next. Each loopback filesystem can also be easily preserved for future reference.

### 2.4 Environmental Conditions

Testing was conducted solely within the Linux VMware virtual machine described in section 2.3. Since the test involved only filesystems local to the test machine, there was no need for network communications or interaction with any other machine. Since the virtual machine was standalone and only one user (the root account being used for testing) was logged in, there were no outside forces present to influence the results of the test.

### 2.5 Description of the Procedures

This section describes the methods and procedures used for testing debugfs's ability to recover deleted files.

## 2.5.1 Equipment Used and Preparation Procedures

The only equipment required for testing was the Dell Inspiron 5100 laptop computer with the operating system and software described in section 2.3. To prepare for testing, the Network cable was disconnected from the laptop computer and the Windows XP operating system was booted. The VMware application was then started. VMware presents a list of the available virtual machines. The fresh copy of the Linux virtual machine created for this test was selected in the VMware window and the virtual machine was booted by selecting the machine from the list of available machines and selecting "Power On" from the menu. The virtual machine desktop appears inside of its own window. All testing was conducted within the Linux virtual machine window.

## 2.5.2 Setup Procedures

The following procedures were performed before each test. These procedures are performed after the test equipment is configured as per section 2.4.1.

**Create a fresh filesystem.** Before each test, a fresh loopback filesystem is created. Creating a fresh filesystem before each test ensures that no residual information from previously deleted files or filesystem activity can affect the results of each test run. As discussed in section 2.3, a loopback filesystem lives completely with a file, which is itself inside of the filesystem of the virtual Linux machine. Being contained inside of a file allows a loopback filesystem to be easily created, stored, and copied. A good example of creating and mounting a loopback filesystem can be found at http://twerner.debian.net/. Following are the procedures used to create each loopback filesystem for use in deleting and recovering files using debugfs.

1. Create the file to hold the filesystem. The file is initially filled with zeros to make it the desired size of the filesystem to be created. All loopback filesystems will be stored in the directory /root/GCFA/recover_eval/filesystems on the Linux virtual machine. The following "dd" command creates the file.

   ```
   # cd /root/GCFA/recover_eval/filesystems
   # dd if=/dev/zero of=filesystem_name.ext2 bs=100k count=500
   ```

   "filesystem_name" is the name of the file to be created. For the purposes of this test, the loopback files will be named based on the scenario number from section 2.5.1 followed by the extension of ".ext2". For example, for testing the recovery of a single small file, the loopback filesystem file name would be "scenario1.ext2". The "dd" command above creates a 50MB filesystem, which will be of sufficient size for the testing to be accomplished.

2. Create a filesystem within the file. Use the mke2fs command to create a filesystem within the file just created. Answer “y” when prompted to proceed despite the fact that the destination is not a device file for a “real” disk.

```
# mke2fs filesystem_name.ext2
mke2fs 1.33 (21-Apr-2003)
filesystem_name is not a block special device.
Proceed anyway? (y,n) y
```

3. Create a mount point and mount the image. Create a mount point for the image. To reduce potential confusion, a separate mount point was created for each image, with the mount point having the same name as the scenario being tested. All mount points were located under /mnt. The image was mounted under the mount point using the “mount” command. The “#” in scenario# is the number of the scenario being tested. For example, “mkdir /mnt/scenario1”.

```
# mkdir /mnt/scenario#
# mount -o loop filesystem_name.ext2 /mnt/scenario#
```

4. Verify the mount. At this point, the filesystem can be accessed through its mount point. Execute “mount” to view the list of mounted filesystems and verify that the loopback filesystem just mounted is present in the list. The file will appear in the list with its full path (/root/GCGA/recover_eval/filesystems/filesystem_name.ext2).

5. Verify that no files are present. Verify that the new filesystem is indeed empty. Execute the command “ls -la /mnt/scenario#”, substituting the correct scenario number for “scenario#”. The only things listed should be “.”, “..”, and “lost+found”.

### 2.5.3 Documentation Standards

All documentation for the test were kept under the directory /root/GCFA/recover_eval. The following documentation and test files were kept as evidence of the testing.

- Script Files. The “script” command captures all input and output to a shell. The script command was executed before each test in order to capture all input and output done in the shell. All testing was conducted within the shell so that all elements of each portion of the test were captured in full detail. The script command is executed with one argument, which is the name of the file to which the script record is stored. The naming convention used during testing was to use “scenario#.script”, with “#” being replaced by the scenario number as defined in section 2.5.1. All script files were stored in the directory /root/GCFA/recover_eval. At the end of a test, the scripting was stopped by pressing the “Ctrl” and “D” keys together.

Note that the script command records everything in its output file, including control information like backspaces and linefeeds. This control information makes the output file rather untidy, but all of the raw information is there from which to draw conclusions. It is also an advantage to a forensics investigation to have such control information in the script file because it shows exactly what keys were pressed what typos were made during the investigation.

Viewing the script output file with an editor like vi or a pager like "less" shows the control information. Using the "cat" command to dump the contents of the script output file to the screen seems to cleanly show the output by processing the control information. For example, if backspace was used to fix a typo while typing on the command line, looking at the output file with vi would show the original typo character, followed by a control character representing the backspace, followed by the corrected character. Using cat to display the output file to the terminal window shows only the corrected text after the backspace was used.

- Loopback Filesystem Files. The loopback filesystem files used in each portion of the test were preserved in the directory /root/GCFA/recover_eval/filesystems as described in section 2.5.2. The filesystem files were preserved as they existed at the end of their use for their portion of the testing.

- Notes. Notes were kept containing comments on test methods or results. The notes were written using the vi text editor and were written in a shell separate from the one being "scripted". One note file was kept, named /root/GCFA/recover_eval/notes.txt. Before each new section of notes, a header was added to the file containing the date, time, and a description of testing being performed.

- Recovered Files. The files recovered by the debugfs dump command were be stored in the /root/GCFA/recover_eval directory and be named "scenario#.recovered#", where the first "#" is the number of the scenario being tested as defined in section 2.51 and the second "#" is the number of the file restored, which is really needed when multiple files are restored as in scenarios five and six.

## 2.5.4 Integrity of Test Results

To protect the integrity of the test results, an MD5 digest value was generated for each of the files mentioned in section 2.4.3 after testing was completed. The "md5sum" command was used to generate the digest value. The digest values were stored in the

/root/GCFA/recover_eval directory with a name beginning with the name of the file for which the digest was being generated, followed by an extension of ".md5".

To further ensure the integrity of the test results, the testing was accomplished on a standalone virtual machine with no users logged in other than the single test account. Once the digest value was generated for a file, that file's permission was then set to mode 444 (read-only for all users).

To ensure that the filesystem from which data is being recovered is not altered, debugfs was used in its default mode of read-only. This ensures that the evidence filesystem is not contaminated by the forensics investigation. An MD5 digest value was used to verify that the filesystem file was not altered by debugfs.

### 2.5.5 Repeatability of Test Results

To be valid, test results must be repeatable and reproducible. The procedures used ensured that the same steps for each portion of the test could be reproduced any number of times in the exact same way as the original test. To provide some proof that the tests were repeatable, at the end of testing each scenario, each test scenario was repeated one time again to ensure that the results were consistent. Before rerunning the tests, the results files from the first run were moved into the directory /root/GCFA/recover_eval/run1. Results files from the second run, when complete, were moved into the directory /root/GCFA/recover_eval/run2.

## 2.6 Criteria for Approval

For each scenario to be tested, the success of debugfs in recovering a file was judged in two areas. First, was debugfs able to identify the deleted inode(s), and second was debugfs able to recover each deleted file's contents. Ideally, debugfs should reliably be able to identify and recover deleted files when no filesystem activity has taken place after the deletion. If filesystem activity has taken place after the deletion, then debugfs should fail to identify any deleted inodes that have been reused for new files.

### 2.6.1 Detailed Test Scenarios

The following scenarios will be tested to demonstrate debugfs's ability to recover deleted files under different conditions.

1. Small file deleted (less than 10KB in size), no other filesystem activity after deletion. This scenario is the simplest case. Since there is no filesystem activity after the deletion, the deleted inode and data blocks should be fully intact. Also, since the file's size is less than 12KB, the deleted file's entire contents should be

stored in the direct data blocks of the inode, so no indirection needs to be dealt with (see section 2.2.2).

2. Large file deleted (approx. 3MB in size), no other filesystem activity after deletion. Since there is no filesystem activity after the deletion, the deleted inode and data blocks should be fully intact. But since the file's size is larger than 12KB, the file's contents will need to use the indirect data blocks, testing dump's ability to follow the indirection.
3. Small file deleted (less than 10KB in size), filesystem activity after deletion. This scenario is similar to scenario one except one new file larger than the deleted file will be created after the original file is deleted in order to force the reuse of the deleted inode. This will test debugfs's behavior if the deleted file's inode or data blocks are overwritten. During testing, it was verified that the system allocated inodes in a sequential order, starting at inode 12 in every case. When inode 12 was deleted, the next file created in the test filesystem reused inode 12. Thus creating one file was sufficient in this case to ensure that the inode was overwritten.
4. Large file deleted (approx. 3MB is size), filesystem activity after deletion. This scenario is similar to scenario two except one new file will smaller than the delete file will be created after the original file is deleted in order to force the reuse of the deleted inode. This will test debugfs's behavior if the deleted file's inode or data blocks are overwritten.
5. Multiple files present (six files of varying sizes), the first, third, and sixth files to be created are then deleted, no other filesystem activity after deletion. This scenario tests debugfs's ability to identify and recover files if multiple deleted inodes exist on the filesystem. For this test, six text files will be created and deleted. Debugfs will then be used to list and dump each of the six files. Since the filesystem will not be changing after the deletion, all of the file's deleted inodes should remain intact.
6. Multiple files present (six files of varying sizes), the first, third, and sixth files to be created are then deleted, three new files created after the deletions. This scenario is similar to the last except that after the three files are deleted, three new files will be created. This will give some insight into how likely inodes are to be reused, at least on a fresh filesystem.
7. Multiple files present and then all deleted. Then one very large file is created to fill all space in the filesystem. This scenario tests what to expect if a deleted file's inode is not needed for another file, but all available data blocks in a filesystem are used by another file.

### 2.6.2 Test Files

The following files were used in the test as the files to be deleted. All are text files so that any recovered files can be easily viewed to see if the files are in tact and to judge what portion of a file was recovered in the case of a partial recovery.

Small file. The small file to be used for scenarios one and three above will be a copy of the ls man page stored to the file "lsfile" using the command "man ls > lsfile". The master copy of this file was stored in /root/GCFA/recover_eval.

Large file. The large file to be used for scenarios two and four above will be a copy of the bash man page repeatedly copied 10 times to a file named "bashfile", making the file's size 3,064,910 bytes. To generate the file, the following command was executed ten times: "man bash >> bashfile". The file was stored in the /root GCFA/recover_eval directory.

Multiple files. The six files used in scenarios five, six, and seven were generated by storing a copy of the man page for six Linux commands to files. Two of the files were the ones created above, lsfile and bashfile. The four others were created from the man pages for "cat", "ps", "grep", and "rm".

The MD5sum value, size, and name for each of the files is listed below.

```
                                  SIZE
          MD5 DIGEST              BYTES    NAME
--------------------------------  ------- --------
fddd0c9d3e53a7b2d6a782f545bdc4a6  3064910 bashfile
efb8742b03fdb05ac28ddd21417e070e     2346 catfile
27f2e4dab9cc019dc5385ed9558ff670    26558 grepfile
51795d784cd0b532c1ff1ed58268595c     8713 lsfile
463479ccbc910b5f1e436487e54dd060    22395 psfile
80e763c57d5f1c977a5d29881f98f584     2807 rmfile
```

Very large file. For scenario seven, a very large file is needed – a file big enough to fill the entire 50MB test filesystem. To generate a file of this size, the tar command was used to create an archive of the /usr/lib directory, about 75MB in size. The archive was stored in the single file "bigfile.tar" in the test filesystem. Since the bigfile.tar file is bigger in size than the test filesystem, tar will write to the file until there is no more space left in the filesystem. Tar will then give an error and stop, leaving behind the bigfile.tar file exactly filling all data blocks in the filesystem.

**For each of the above scenarios, the following test procedures will be followed.**

1. Begin a script of the test session with the command "script scenario#.script", where "#" is the number of the test scenario being performed as defined in section 2.5.1.
2. Generate and mount a fresh filesystem as specified in section 2.5.2, naming the filesystem file in the form "scenario#.ext2", replacing the "#" with the scenario number above.
3. Copy the appropriate test file or files into the mounted loopback filesystem as specified in the test scenario.

    a. For <u>scenarios one and three</u>, copy the file "lsfile" to the loopback filesystem. For example, in scenario one, from the "filesystems" directory, execute "cp ../lsfile /mnt/scenario1".
    b. For <u>scenarios two and four</u>, copy the file "bashfile" to the loopback filesystem. For example, in scenario two, from the "filesystems" directory, execute "cp ../bashfile /mnt/scenario1".
    c. For <u>scenarios five, six, and seven</u>, copy all six files listed in section 2.5.2 to the loopback filesystem one at a time in alphabetical order. For example, in scenario five, from the "filesystems" directory, execute "cp ../filename /mnt/scenario5", where "filename" is one of the six files from section 2.5.2, copied in alphabetical order; bashfile, catfile, grepfile, lsfile, psfile, rmfile.
4. Execute an "ls -la" of the loopback filesystem to verify that the copy from step three was successful.
5. Execute "sync" to ensure that the changes to the inodes are properly written to the filesystem file.
6. Delete the appropriate files, depending on the scenario, from the loopback filesystem that were copied in step three. To delete the files, use the "rm" command. For example, in scenario one use "rm /mnt/scenario1/lsfile".
    a. For <u>scenarios one through four</u>, delete the one file that was copied into the loopback filesystem.
    b. For <u>scenarios five and six</u>, delete the first, third, and sixth files copied into the loopback filesystem. These files would be "bashfile", "grepfile", and "rmfile".
    c. For <u>scenario seven</u>, delete all six files that were copied into the loopback filesystem.
7. Execute "sync" to ensure that the changes to the inodes are properly written to the filesystem file.
8. Execute an "ls -la" of the loopback filesystem to verify that the delete from step six was successful.
9. For <u>scenarios 3 and 4 ONLY</u>:
    a. Create one new file by copying /root/GCFA/recover_eval/psfile into the mounted loopback filesystem. For example, for scenario three, execute "cp psfile /mnt/scenario3".
    b. Verify that the psfile file is present by executing "ls -la" on the mounted filesystem.
    c. Execute "sync" to ensure that the changes to the inodes are properly written to the filesystem file.
10. For <u>scenario 6 ONLY</u>: Create three new files in the loopback filesystem by dumping the output from the manual page for "man" to three different filenames. The contents of the files created here are not important; of interest is how the inodes are allocated when the files are created. Create the files as follows.
    a. man man > /mnt/scenario6/newfile1
    b. man man > /mnt/scenario6/newfile2
    c. man man > /mnt/scenario6/newfile3
    d. Verify that the new files are present by executing "ls -la /mnt/scenario6".

    e. Execute "sync" to ensure that the changes to the inodes are properly written to the filesystem file.

11. For scenario 7 ONLY: Create a single large file to fill the filesystem by executing the command "tar cf /mnt/scenario7/bigfile.tar /usr/lib". The tar command will give an error and stop when the filesystem is full. This is to be expected. Execute "ls -la /mnt/scenario7" to verify that bigfile.tar is there. Then execute "sync" to ensure that the changes to the inodes are properly written to the filesystem file.
12. Unmount the loopback filesystem to help preserve its integrity using the "umount" command. For example, to unmount the loopback filesystem from scenario 1, execute the command "umount /mnt/scenario1".
13. Perform an md5sum on the filesystem file. For example, for scenario 1, execute "md5sum senario1.ext2". The script will serve to record the results.
14. Start debugfs, pointed to the filesystem file. For example, for scenario 1, execute "debugfs scenario1.ext2" while in the /root/GCFA/recover_eval/filesystems directory.
15. In debugfs, execute the command "lsdel" to list the deleted inodes. The script captures all of this information in the script file. Record any anomalies in the notes file. Note the inode number of any deleted files present. This number will be used in recovering the files.
16. If deleted inodes are seen in step 15, dump each of the inode contents to file using the debugfs dump command. The syntax of the dump command is "dump <inode> outfile", where "inode" is the number of the inode to be dumped (the <>s are part of the syntax) and "outfile" is the name of the restored file, to the directory path as defined in section 2.5.3. For example, if inode 12 was shown as deleted, execute "dump <12> ../scenario1.recovered1" from the "filesystems" directory. Quit debugfs with the "quit" command. If no deleted files were present, execute the "ls" command in debugfs to show what is there, and then quit debugfs with the "quit" command and go to step 18.
17. Verify that each dumped file is the same as the original file. For each file recovered in step 16, generate an MD5 digest value using the md5sum command and compare the value to the original file. If they match, the files are identical. The script will serve to record the results.
18. Generate an MD5 digest value for the filesystem file and compare it to the value generated in step 13. If they match, the filesystem was not altered by debugfs.
19. Set permissions on the filesystem file to mode 444 (read-only for all users) to help protect its integrity. This is performed with the "chmod" command. For example, in the command to use for the first scenario would be "chmod 444 scenario1.ext2".
20. Stop the scripting of the shell by entering Ctrl-D (press the Ctrl and D keys at the same time). All input and output done by the shell during the script run is preserved in the script file.
21. Generate an MD5 digest value for the script file just completed. Store the results in a file named the same as the script file with ".md5" at the end. For example, for scenario one, the command would be "md5sum scenario1.script > scenario1.script.md5".

After all seven scenarios were tested as per the above procedures, the test results were moved into the directory /root/GCFA/recover_eval/run1 as follows.

```
   # cd /root/GCFA/recover_eval
   # mkdir run1; mv scenario* run1
   # mkdir run1/filesystems; mv filesystems/* run1/filesystems
```

The test was then run again exactly as before to demonstrate that the test procedures and results are repeatable.  After completion of the second run, the results were moved into the directory /root/GCFA/recover_eval/run2.

## 2.7 Data and Results

The seven test scenarios were executed as per the procedures in section 2.5.  The results will be presented in this section and analyzed in section 2.7.  The test results presented here were taken from the script output file and were cleaned up to remove the special characters that were present in the script file.  The original script file including the special characters, typos, etc. has been preserved for future reference as described in section 2.5.

### 2.7.1 Scenario 1

Scenario one involved creating, deleting, and recovering a file less than 10KB in size. Following is the full output from the script, which captured the entire scenario 1 test.

```
Script started on Sun Aug 17 13:04:52 2003
[root@localhost recover_eval]# cd filesystems
[root@localhost filesystems]# pwd
/root/GCFA/recover_eval/filesystems
[root@localhost filesystems]# ls
[root@localhost filesystems]# dd if=/dev/zero of=scenario1.ext2 bs=100k count=500
500+0 records in
500+0 records out
[root@localhost filesystems]# ls -l
total 50056
-rw-r--r--    1 root     root     51200000 Aug 17 13:05 scenario1.ext2
[root@localhost filesystems]# mke2fs scenario1.ext2
mke2fs 1.33 (21-Apr-2003)
scenario1.ext2 is not a block special device.
Proceed anyway? (y,n) y
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
12544 inodes, 50000 blocks
2500 blocks (5.00%) reserved for the super user
First data block=1
7 block groups
8192 blocks per group, 8192 fragments per group
1792 inodes per group
Superblock backups stored on blocks:
        8193, 24577, 40961
```

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

This filesystem will be automatically checked every 37 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
[root@localhost filesystems]# mkdir /mnt/scenario1
[root@localhost filesystems]# mount -o loop scenario1.ext2 /mnt/scenario.ext2 /mnt/scenario1
[root@localhost filesystems]# mount
/dev/sda2 on / type ext3 (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/sda1 on /boot type ext3 (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
/root/GCFA/recover_eval/filesystems/scenario1.ext2 on /mnt/scenario1 type ext2
(rw,loop=/dev/loop0)
[root@localhost filesystems]# ls -la /mnt/scenario1
total 17
drwxr-xr-x    3 root     root         1024 Aug 17 13:05 .
drwxr-xr-x    5 root     root         4096 Aug 17 13:06 ..
drwx------    2 root     root        12288 Aug 17 13:05 lost+found
[root@localhost filesystems]# cp ../lsfile /mnt/scenario1
[root@localhost filesystems]# ls -la /mnt/scenario1
total 26
drwxr-xr-x    3 root     root         1024 Aug 17 13:10 .
drwxr-xr-x    5 root     root         4096 Aug 17 13:06 ..
drwx------    2 root     root        12288 Aug 17 13:05 lost+found
-rw-r--r--    1 root     root         8713 Aug 17 13:10 lsfile
[root@localhost filesystems]# sync
[root@localhost filesystems]# rm /mnt/scenario1/lsfile
rm: remove regular file `/mnt/scenario1/lsfile'? y
[root@localhost filesystems]# sync
[root@localhost filesystems]# ls -la /mnt/scenario1
total 17
drwxr-xr-x    3 root     root         1024 Aug 17 13:11 .
drwxr-xr-x    5 root     root         4096 Aug 17 13:06 ..
drwx------    2 root     root        12288 Aug 17 13:05 lost+found
[root@localhost filesystems]# umount /mnt/scenario1
[root@localhost filesystems]# pwd
/root/GCFA/recover_eval/filesystems
[root@localhost filesystems]# ls
scenario1.ext2
[root@localhost filesystems]# md5sum scenario1.ext2
29cbe4924a79211226836d6f9d83d9c8  scenario1.ext2
[root@localhost filesystems]# debugfs scenario1.ext2
debugfs 1.33 (21-Apr-2003)
debugfs:  lsdel
 Inode  Owner  Mode    Size    Blocks   Time deleted
    12      0 100644   8713    9/   9 Sun Aug 17 13:11:12 2003
1 deleted inodes found.
debugfs:  dump <12> ../scenario1.recovered1
debugfs:  q
[root@localhost filesystems]# md5sum ../scenario1.recovered1
51795d784cd0b532c1ff1ed58268595c  ../scenario1.recovered1
[root@localhost filesystems]# md5sum scenario1.ext2
29cbe4924a79211226836d6f9d83d9c8  scenario1.ext2
[root@localhost filesystems]# chmod 444 scenario1.ext2
[root@localhost filesystems]#
Script done on Sun Aug 17 13:19:30 2003
```

The results of test met the expectation that debugfs was able to recover the deleted file in tact, and that debugfs did not modify the test filesystem in any way during the file recovery.  The script output above shows that the MD5 digest values for the recovered file (scenario1.recovered1) and the original file (lsfile from section 2.6.2) match perfectly, proving that they are identical files.  Also, the MD5 digest value for the test filesystem, scenario1.ext2, did not change after running debugfs on the filesystem, proving that debugfs does not modify the filesystem when used as described for this testing to recover deleted files.

### 2.7.2 Scenario 2

Scenario two involved creating, deleting, and recovering a file of approximately 3MB in size.  Following is the full output from the script, which captured the entire scenario 2 test.

```
Script started on Sun Aug 17 13:33:46 2003
[root@localhost recover_eval]# cd filesystems
[root@localhost filesystems]# dd if=/dev/zero of=scenario2.ext2 bs=100k count=500
500+0 records in
500+0 records out
[root@localhost filesystems]# ls -l
-r--r--r--    1 root     root     51200000 Aug 17 13:05  [00mscenario1.ext2 [00m
-rw-r--r--    1 root     root     51200000 Aug 17 13:34  [00mscenario2.ext2 [00m
[root@localhost filesystems]# mke2fs scenario2.ext2
mke2fs 1.33 (21-Apr-2003)
scenario2.ext2 is not a block special device.
Proceed anyway? (y,n) y
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
12544 inodes, 50000 blocks
2500 blocks (5.00%) reserved for the super user
First data block=1
7 block groups
8192 blocks per group, 8192 fragments per group
1792 inodes per group
Superblock backups stored on blocks:
                                    8193, 24577, 40961

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

This filesystem will be automatically checked every 21 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
[root@localhost filesystems]# mkdir /mnt/scenario2
[root@localhost filesystems]# mount -o loop scenario2.ext2 /mnt/scenario2
[root@localhost filesystems]# mount
/dev/sda2 on / type ext3 (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/sda1 on /boot type ext3 (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
/root/GCFA/recover_eval/filesystems/scenario2.ext2 on /mnt/scenario2 type ext2
(rw,loop=/dev/loop0)
[root@localhost filesystems]# ls -la /mnt/scenario2
total 17
drwxr-xr-x    3 root     root         1024 Aug 17 13:34 .
drwxr-xr-x    6 root     root         4096 Aug 17 13:34 ..
drwx------    2 root     root        12288 Aug 17 13:34 lost+found
[root@localhost filesystems]# cp ../bashfile /mnt/scenario2
[root@localhost filesystems]# ls -la /mnt/scenario2
total 3024
drwxr-xr-x    3 root     root         1024 Aug 17 13:36 .
drwxr-xr-x    6 root     root         4096 Aug 17 13:34 ..
-rw-r--r--    1 root     root      3064910 Aug 17 13:36 bashfile
drwx------    2 root     root        12288 Aug 17 13:34 lost+found
[root@localhost filesystems]# sync
[root@localhost filesystems]# rm /mnt/scenario2/bashfile
rm: remove regular file `/mnt/scenario2/bashfile'? y
```

```
[root@localhost filesystems]# sync
[root@localhost filesystems]# ls -la /mnt/scenario2
total 17
drwxr-xr-x    3 root     root         1024 Aug 17 13:36 .
drwxr-xr-x    6 root     root         4096 Aug 17 13:34 ..
drwx------    2 root     root        12288 Aug 17 13:34 lost+found
[root@localhost filesystems]# sync
[root@localhost filesystems]# umount /mnt/scenario2
[root@localhost filesystems]# mount
/dev/sda2 on / type ext3 (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/sda1 on /boot type ext3 (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
[root@localhost filesystems]# md5sum scenario2.ext2
44e9e36048e9d6f3208a7007b3066596  scenario2.ext2
[root@localhost filesystems]# debugfs scenario2.ext2
debugfs 1.33 (21-Apr-2003)
debugfs:  lsdel
 Inode  Owner  Mode    Size    Blocks   Time deleted
    12      0 100644 3064910 3007/3007 Sun Aug 17 13:36:36 2003
1 deleted inodes found.
debugfs:  dump <12> ../scenario2.recovered1
debugfs:  quit
[root@localhost filesystems]# md5sum ../scenario2.recovered1
fddd0c9d3e53a7b2d6a782f545bdc4a6  ../scenario2.recovered1
[root@localhost filesystems]# chmod 444 scenario2.ext2
[root@localhost filesystems]#
Script done on Sun Aug 17 13:39:39 2003
```

As with scenario 1, the results of test met the expectation that debugfs was able to recover the deleted file in tact, and that debugfs did not modify the test filesystem in any way during the file recovery.  The script output above shows that the MD5 digest values for the recovered file (scenario2.recovered1) and the original file (bashfile from section 2.6.2) match perfectly, proving that they are identical files.  Also, the MD5 digest value for the test filesystem, scenario2.ext2, did not change after running debugfs on the filesystem, proving that debugfs does not modify the filesystem when used as described for this testing to recover deleted files.

### 2.7.3 Scenario 3

Scenario 3 involved creating, deleting, and attempting to recover a file of less than 10KB in size, with a new file of larger size being created in the test filesystem between the deletion and the attempted recovery.  Following is the full output from the script, which captured the entire scenario 3 test.

```
Script started on Sun Aug 17 14:23:02 2003
[root@localhost recover_eval]# cd filesystems
[root@localhost filesystems]# dd if=/dev/zero of=scenario3.ext2 bs=100k count=500
500+0 records in
500+0 records out
[root@localhost filesystems]# mke2fs scenario3.ext2
mke2fs 1.33 (21-Apr-2003)
scenario3.ext2 is not a block special device.
Proceed anyway? (y,n) y
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
```

```
12544 inodes, 50000 blocks
2500 blocks (5.00%) reserved for the super user
First data block=1
7 block groups
8192 blocks per group, 8192 fragments per group
1792 inodes per group
Superblock backups stored on blocks:
                                    8193, 24577, 40961

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

This filesystem will be automatically checked every 35 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
[root@localhost filesystems]# mkdir /mnt/scenario3
[root@localhost filesystems]# mount -o loop scenario3.ext2 /mnt/scenario3
[root@localhost filesystems]# mount
/dev/sda2 on / type ext3 (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/sda1 on /boot type ext3 (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
/root/GCFA/recover_eval/filesystems/scenario3.ext2 on /mnt/scenario3 type ext2
(rw,loop=/dev/loop0)
[root@localhost filesystems]# ls -la /mnt/scenario3
total 17
drwxr-xr-x    3 root     root         1024 Aug 17 14:23 .
drwxr-xr-x    7 root     root         4096 Aug 17 14:24 ..
drwx------    2 root     root        12288 Aug 17 14:23 lost+found
[root@localhost filesystems]# cp ../lsfile /mnt/scenario3
[root@localhost filesystems]# ls -la /mnt/scenario3
total 26
drwxr-xr-x    3 root     root         1024 Aug 17 14:25 .
drwxr-xr-x    7 root     root         4096 Aug 17 14:24 ..
drwx------    2 root     root        12288 Aug 17 14:23 lost+found
-rw-r--r--    1 root     root         8713 Aug 17 14:25 lsfile
[root@localhost filesystems]# sync
[root@localhost filesystems]# rm /mnt/scenario3/lsfile
rm: remove regular file `/mnt/scenario3/lsfile'? yes
[root@localhost filesystems]# ls -la /mnt/scenario3
total 17
drwxr-xr-x    3 root     root         1024 Aug 17 14:26 .
drwxr-xr-x    7 root     root         4096 Aug 17 14:24 ..
drwx------    2 root     root        12288 Aug 17 14:23 lost+found
[root@localhost filesystems]# cp ../psfile /mnt/scenario3
[root@localhost filesystems]# ls -la /mnt/scenario3
total 40
drwxr-xr-x    3 root     root         1024 Aug 17 14:26 .
drwxr-xr-x    7 root     root         4096 Aug 17 14:24 ..
drwx------    2 root     root        12288 Aug 17 14:23 lost+found
-rw-r--r--    1 root     root        22395 Aug 17 14:26 psfile
[root@localhost filesystems]# sync
[root@localhost filesystems]# umount /mnt/scenario3
[root@localhost filesystems]# md5sum secenario3.ext2
d03da533ad344f508482cecf6102e442  scenario3.ext2
[root@localhost filesystems]# debugfs scenario3.ext2
debugfs 1.33 (21-Apr-2003)
debugfs:  lsdel
 Inode  Owner  Mode    Size    Blocks   Time deleted
0 deleted inodes found.
debugfs:  ls
 2  (12) .    2  (12) ..    11  (20) lost+found    12  (980) psfile
debugfs: ls -l
      2   40755 (2)      0      0    1024 17-Aug-2003 14:26 .
      2   40755 (2)      0      0    1024 17-Aug-2003 14:26 ..
     11   40700 (2)      0      0   12288 17-Aug-2003 14:23 lost+found
     12  100644 (1)      0      0   22395 17-Aug-2003 14:26 psfile

debugfs:  quit
[root@localhost filesystems]# chmod 444 scenario3.ext2
```

```
[root@localhost filesystems]# md5sum secenario3.ext2
d03da533ad344f508482cecf6102e442  scenario3.ext2
[root@localhost filesystems]#

Script done on Sun Aug 17 14:29:57 2003
```

The results met the expectation that debugfs was not able to recover the delete file because the deleted inode was reused by the file psfile that was copied into the filesystem after the deletion.  Debugfs therefore found no deleted inodes to be dumped.  Also, debugfs did not make any changes to the test filesystem.

### 2.7.4 Scenario 4

Scenario 4 involved creating, deleting, and attempting to recover a file of approximately 3MB in size, with a new file of smaller size being created in the test filesystem between the deletion and the attempted recovery.  Following is the full output from the script, which captured the entire scenario 4 test.

```
Script started on Sun Aug 17 14:30:52 2003
[root@localhost recover_eval]# cd filesystems
[root@localhost filesystems]# dd if=/dev/zero of=scenario4.ext2 bs=100k count=500
500+0 records in
500+0 records out
[root@localhost filesystems]# mke2fs scenario4.ext2
mke2fs 1.33 (21-Apr-2003)
scenario4.ext2 is not a block special device.
Proceed anyway? (y,n) yes
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
12544 inodes, 50000 blocks
2500 blocks (5.00%) reserved for the super user
First data block=1
7 block groups
8192 blocks per group, 8192 fragments per group
1792 inodes per group
Superblock backups stored on blocks:
                                    8193, 24577, 40961

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

This filesystem will be automatically checked every 33 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
[root@localhost filesystems]# mkdir /mnt/scenario4
[root@localhost filesystems]# mount -o loop scenario4.ext2 /mnt/scenario4
[root@localhost filesystems]# mount
/dev/sda2 on / type ext3 (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/sda1 on /boot type ext3 (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
/root/GCFA/recover_eval/filesystems/scenario4.ext2 on /mnt/scenario4 type ext2
(rw,loop=/dev/loop0)
[root@localhost filesystems]# ls -la /mnt/scenario4
total 17
drwxr-xr-x    3 root     root         1024 Aug 17 14:31 .
drwxr-xr-x    8 root     root         4096 Aug 17 14:31 ..
drwx------    2 root     root        12288 Aug 17 14:31 lost+found
[root@localhost filesystems]# cp ../bashfile /mnt/scenario4
```

```
[root@localhost filesystems]# ls -la /mnt/scenario4
total 3024
drwxr-xr-x    3 root     root         1024 Aug 17 14:33 .
drwxr-xr-x    8 root     root         4096 Aug 17 14:31 ..
-rw-r--r--    1 root     root      3064910 Aug 17 14:33 bashfile
drwx------    2 root     root        12288 Aug 17 14:31 lost+found
[root@localhost filesystems]# sync
[root@localhost filesystems]# rm /mnt/scenario4/bashfile
rm: remove regular file `/mnt/scenario4/bashfile'? yes
[root@localhost filesystems]# ls -la /mnt/scenario4
total 17
drwxr-xr-x    3 root     root         1024 Aug 17 14:33 .
drwxr-xr-x    8 root     root         4096 Aug 17 14:31 ..
drwx------    2 root     root        12288 Aug 17 14:31 lost+found
[root@localhost filesystems]# sync
[root@localhost filesystems]# ls -la /mnt/scenario4
total 17
drwxr-xr-x    3 root     root         1024 Aug 17 14:33  [00;34m. [00m
drwxr-xr-x    8 root     root         4096 Aug 17 14:31  [00;34m.. [00m
drwx------    2 root     root        12288 Aug 17 14:31  [00;34mlost+found [00m
[root@localhost filesystems]# cp ../psfile /mnt/scenario4
[root@localhost filesystems]# ls -la /mnt/scenario4
total 40
drwxr-xr-x    3 root     root         1024 Aug 17 14:34  [00;34m. [00m
drwxr-xr-x    8 root     root         4096 Aug 17 14:31  [00;34m.. [00m
drwx------    2 root     root        12288 Aug 17 14:31  [00;34mlost+found [00m
-rw-r--r--    1 root     root        22395 Aug 17 14:34  [00mpsfile [00m
[root@localhost filesystems]# sync
[root@localhost filesystems]# umount /mnt/scenario4
[root@localhost filesystems]# md5sum scenario4.ext2
f75cfe03236e3d06ae61297dc0ea341d  scenario4.ext2
[root@localhost filesystems]# debugfs scenario4.ext2
debugfs 1.33 (21-Apr-2003)
debugfs:  lsdel
 Inode  Owner  Mode    Size    Blocks   Time deleted
0 deleted inodes found.
debugfs:  ls
 2  (12) .    2  (12) ..    11  (20) lost+found    12  (980) psfile
debugfs:  ls -l
      2   40755 (2)      0      0    1024 17-Aug-2003 14:34 .
      2   40755 (2)      0      0    1024 17-Aug-2003 14:34 ..
     11   40700 (2)      0      0   12288 17-Aug-2003 14:31 lost+found
     12  100644 (1)      0      0   22395 17-Aug-2003 14:34 psfile

debugfs:  quit
[root@localhost filesystems]# md5sum scenario4.ext2
f75cfe03236e3d06ae61297dc0ea341d  scenario4.ext2
[root@localhost filesystems]# chmod 444 scenario4.ext2
[root@localhost filesystems]#

Script done on Sun Aug 17 14:36:50 2003
```

The results met the expectation that debugfs was not able to recover the delete file because the deleted inode was reused by the file psfile that was copied into the filesystem after the deletion.  Debugfs therefore found no deleted inodes to be dumped. Also, debugfs did not make any changes to the test filesystem.

### 2.7.5 Scenario 5

Scenario 5 involved creating six files, deleting three of the files, and attempting to recover the three deleted files.  Following is the full output from the script, which captured the entire scenario 5 test.

```
Script started on Sun Aug 17 15:13:22 2003
[root@localhost recover_eval]# cd filesystems
[root@localhost filesystems]# pwd
/root/GCFA/recover_eval/filesystems
[root@localhost filesystems]# dd if=/dev/zero of=scenario5.ext2 bs=100k count=500
500+0 records in
500+0 records out
[root@localhost filesystems]# mke2fs scenario5.ext2
mke2fs 1.33 (21-Apr-2003)
scenario5.ext2 is not a block special device.
Proceed anyway? (y,n) y
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
12544 inodes, 50000 blocks
2500 blocks (5.00%) reserved for the super user
First data block=1
7 block groups
8192 blocks per group, 8192 fragments per group
1792 inodes per group
Superblock backups stored on blocks:
                                    8193, 24577, 40961

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

This filesystem will be automatically checked every 38 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
[root@localhost filesystems]# mkdir /mnt/scenario5
[root@localhost filesystems]# mount -o loop secnario5.ext2 /mnt/scenario5
[root@localhost filesystems]# mount
/dev/sda2 on / type ext3 (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/sda1 on /boot type ext3 (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
/root/GCFA/recover_eval/filesystems/scenario5.ext2 on /mnt/scenario5 type ext2
(rw,loop=/dev/loop0)
[root@localhost filesystems]# ls -la /mnt/scenario5
total 17
drwxr-xr-x    3 root     root         1024 Aug 17 15:15 .
drwxr-xr-x    9 root     root         4096 Aug 17 15:15 ..
drwx------    2 root     root        12288 Aug 17 15:15 lost+found
[root@localhost filesystems]# cp ../bashfile /mnt/scenario5
[root@localhost filesystems]# cp ../catfile /mnt/scenario5
[root@localhost filesystems]# cp ../grepfile /mnt/scenario5
[root@localhost filesystems]# cp ../lsfile /mnt/scenario5
[root@localhost filesystems]# cp ../psfile /mnt/scenario5
[root@localhost filesystems]# cp ../rmfile /mnt/scenario5
[root@localhost filesystems]# ls -la /mnt/scenario5
total 3089
drwxr-xr-x    3 root     root         1024 Aug 17 15:16 .
drwxr-xr-x    9 root     root         4096 Aug 17 15:15 ..
-rw-r--r--    1 root     root      3064910 Aug 17 15:16 bashfile
-rw-r--r--    1 root     root         2346 Aug 17 15:16 catfile
-rw-r--r--    1 root     root        26558 Aug 17 15:16 grepfile
drwx------    2 root     root        12288 Aug 17 15:15 lost+found
-rw-r--r--    1 root     root         8713 Aug 17 15:16 lsfile
-rw-r--r--    1 root     root        22395 Aug 17 15:16 psfile
-rw-r--r--    1 root     root         2807 Aug 17 15:16 rmfile
[root@localhost filesystems]# sync
[root@localhost filesystems]# rm /mnt/scenario5/bashfile
rm: remove regular file `/mnt/scenario5/bashfile'? y
[root@localhost filesystems]# rm /mnt/scenario5/grepfile
rm: remove regular file `/mnt/scenario5/grepfile'? y
[root@localhost filesystems]# rm /mnt/scenario5/rmfile
rm: remove regular file `/mnt/scenario5/rmfile'? y
[root@localhost filesystems]# sync
[root@localhost filesystems]# ls -la /mnt/scenario5
```

```
total 52
drwxr-xr-x    3 root     root         1024 Aug 17 15:17 .
drwxr-xr-x    9 root     root         4096 Aug 17 15:15 ..
-rw-r--r--    1 root     root         2346 Aug 17 15:16 catfile
drwx------    2 root     root        12288 Aug 17 15:15 lost+found
-rw-r--r--    1 root     root         8713 Aug 17 15:16 lsfile
-rw-r--r--    1 root     root        22395 Aug 17 15:16 psfile
[root@localhost filesystems]# umount /mnt/scenario5
[root@localhost filesystems]# md5sum scenario5.ext2
f868895152e12f38c14b0e220647d8b4  scenario5.ext2
[root@localhost filesystems]# debugfs scenario5.ext2
debugfs 1.33 (21-Apr-2003)
debugfs:  lsdel
 Inode  Owner  Mode    Size    Blocks   Time deleted
    12      0 100644 3064910 3007/3007 Sun Aug 17 15:17:25 2003
    14      0 100644  26558   27/  27 Sun Aug 17 15:17:31 2003
    17      0 100644   2807    3/   3 Sun Aug 17 15:17:41 2003
3 deleted inodes found.
debugfs:  dump <12> ../scenario5_recovered1
debugfs:  dump <14> ../scenario5_recovered2
debugfs:  dump <17> ../scenario5_recovered3
debugfs:  q
[root@localhost filesystems]# md5sum ../scenario5_recovered*
fddd0c9d3e53a7b2d6a782f545bdc4a6  ../scenario5_recovered1
27f2e4dab9cc019dc5385ed9558ff670  ../scenario5_recovered2
80e763c57d5f1c977a5d29881f98f584  ../scenario5_recovered3
[root@localhost filesystems]# md5sum scenario5.ext2
f868895152e12f38c14b0e220647d8b4  scenario5.ext2
[root@localhost filesystems]# chmod 444 scenario5.ext2
[root@localhost filesystems]# exit

Script done on Sun Aug 17 15:23:08 2003
```

The results met the expectation that debugfs was able to recover all three deleted files in tact, and that debugfs did not modify the test filesystem in any way during the file recovery.  The script output above shows that the MD5 digest values for the recovered files (scenario5.recovered1 thru 3) and the original files (bashfile, grepfile, and rmfile from section 2.6.2) match perfectly, proving that they are identical files.  Also, the MD5 digest value for the test filesystem, scenario5.ext2, did not change after running debugfs on the filesystem, proving that debugfs does not modify the filesystem when used as described for this testing to recover deleted files.

### 2.7.6 Scenario 6

Scenario 6 involved creating six files, deleting three of the files, creating three new files, and then attempting to recover the three deleted files.  Following is the full output from the script, which captured the entire scenario 6 test.

```
Script started on Sun Aug 17 15:24:03 2003
[root@localhost recover_eval]# cd filesystems
[root@localhost filesystems]# pwd
/root/GCFA/recover_eval/filesystems
[root@localhost filesystems]# dd if=/dev/zero of=scenario6.ext2 bs=100k count=500
500+0 records in
500+0 records out
[root@localhost filesystems]# mke2fs scenario6.ext2
mke2fs 1.33 (21-Apr-2003)
scenario6.ext2 is not a block special device.
Proceed anyway? (y,n) y
Filesystem label=
OS type: Linux
```

```
Block size=1024 (log=0)
Fragment size=1024 (log=0)
12544 inodes, 50000 blocks
2500 blocks (5.00%) reserved for the super user
First data block=1
7 block groups
8192 blocks per group, 8192 fragments per group
1792 inodes per group
Superblock backups stored on blocks:
                                    8193, 24577, 40961

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

This filesystem will be automatically checked every 35 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
[root@localhost filesystems]# mkdir /mnt/scenario6
[root@localhost filesystems]# mount -o loop scenario6.ext2 /mnt/scenario6
[root@localhost filesystems]# mount
/dev/sda2 on / type ext3 (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/sda1 on /boot type ext3 (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
/root/GCFA/recover_eval/filesystems/scenario6.ext2 on /mnt/scenario6 type ext2
(rw,loop=/dev/loop0)
[root@localhost filesystems]# ls -la /mnt/scenario6
total 17
drwxr-xr-x    3 root     root         1024 Aug 17 15:24 .
drwxr-xr-x   10 root     root         4096 Aug 17 15:25 ..
drwx------    2 root     root        12288 Aug 17 15:24 lost+found
[root@localhost filesystems]# cp ../bashfile /mnt/scenario6
[root@localhost filesystems]# cp ../catfile /mnt/scenario6
[root@localhost filesystems]# cp ../grepfile /mnt/scenario6
[root@localhost filesystems]# cp ../lsfile /mnt/scenario6
[root@localhost filesystems]# cp ../psfile /mnt/scenario6
[root@localhost filesystems]# cp ../rmfile /mnt/scenario6
[root@localhost filesystems]# ls -la /mnt/scenario6
total 3089
drwxr-xr-x    3 root     root         1024 Aug 17 15:26 .
drwxr-xr-x   10 root     root         4096 Aug 17 15:25 ..
-rw-r--r--    1 root     root      3064910 Aug 17 15:25 bashfile
-rw-r--r--    1 root     root         2346 Aug 17 15:26 catfile
-rw-r--r--    1 root     root        26558 Aug 17 15:26 grepfile
drwx------    2 root     root        12288 Aug 17 15:24 lost+found
-rw-r--r--    1 root     root         8713 Aug 17 15:26 lsfile
-rw-r--r--    1 root     root        22395 Aug 17 15:26 psfile
-rw-r--r--    1 root     root         2807 Aug 17 15:26 rmfile
[root@localhost filesystems]# sync
[root@localhost filesystems]# rm /mnt/scenario6/bashfile
rm: remove regular file `/mnt/scenario6/bashfile'? y
[root@localhost filesystems]# rm /mnt/scenario6/grepfile
rm: remove regular file `/mnt/scenario6/grepfile'? y
[root@localhost filesystems]# rm /mnt/scenario6/rmfile
rm: remove regular file `/mnt/scenario6/rmfile'? y
[root@localhost filesystems]# sync
[root@localhost filesystems]# ls -la /mnt/scenario6
total 52
drwxr-xr-x    3 root     root         1024 Aug 17 15:27 .
drwxr-xr-x   10 root     root         4096 Aug 17 15:25 ..
-rw-r--r--    1 root     root         2346 Aug 17 15:26 catfile
drwx------    2 root     root        12288 Aug 17 15:24 lost+found
-rw-r--r--    1 root     root         8713 Aug 17 15:26 lsfile
-rw-r--r--    1 root     root        22395 Aug 17 15:26 psfile
[root@localhost filesystems]# man man > /mnt/scenario6/newfile1
[root@localhost filesystems]# man man > /mnt/scenario6/newfile2
[root@localhost filesystems]# man man > /mnt/scenario6/newfile3
[root@localhost filesystems]# ls -la /mnt/scenario6
total 97
drwxr-xr-x    3 root     root         1024 Aug 17 15:28 .
```

```
drwxr-xr-x   10 root     root         4096 Aug 17 15:25 ..
-rw-r--r--    1 root     root         2346 Aug 17 15:26 catfile
drwx------    2 root     root        12288 Aug 17 15:24 lost+found
-rw-r--r--    1 root     root         8713 Aug 17 15:26 lsfile
-rw-r--r--    1 root     root        14250 Aug 17 15:28 newfile1
-rw-r--r--    1 root     root        14250 Aug 17 15:28 newfile2
-rw-r--r--    1 root     root        14250 Aug 17 15:28 newfile3
-rw-r--r--    1 root     root        22395 Aug 17 15:26 psfile
[root@localhost filesystems]# sync
[root@localhost filesystems]# umount /mnt/scenario6
[root@localhost filesystems]# md5sum scenario6.ext2
b5263689085e3afd348487a26cc6f741  scenario6.ext2
[root@localhost filesystems]# debugfs scenario6.ext2
debugfs 1.33 (21-Apr-2003)
debugfs:  lsdel
 Inode  Owner  Mode    Size    Blocks   Time deleted
0 deleted inodes found.
debugfs:  ls
 2  (12) .    2  (12) ..    11  (20) lost+found    12  (16) newfile1
 13  (16) catfile    14  (16) newfile2    15  (16) lsfile    16  (16) psfile
 17  (900) newfile3
debugfs:  ls -l
      2   40755 (2)      0      0    1024 17-Aug-2003 15:28 .
      2   40755 (2)      0      0    1024 17-Aug-2003 15:28 ..
     11   40700 (2)      0      0   12288 17-Aug-2003 15:24 lost+found
     12  100644 (1)      0      0   14250 17-Aug-2003 15:28 newfile1
     13  100644 (1)      0      0    2346 17-Aug-2003 15:26 catfile
     14  100644 (1)      0      0   14250 17-Aug-2003 15:28 newfile2
     15  100644 (1)      0      0    8713 17-Aug-2003 15:26 lsfile
     16  100644 (1)      0      0   22395 17-Aug-2003 15:26 psfile
     17  100644 (1)      0      0   14250 17-Aug-2003 15:28 newfile3

debugfs:  q
[root@localhost filesystems]# md5sum scenario6.ext2
b5263689085e3afd348487a26cc6f741  scenario6.ext2
[root@localhost filesystems]# chmod 444 scenario6.ext2
[root@localhost filesystems]# exit

Script done on Sun Aug 17 15:30:24 2003
```

The results met the expectation that debugfs was not able to recover any of the deleted files because the deleted inodes were reused by the three new files created (newfile1, 2, and 3). This test demonstrated that the system will allocate the lowest numbered available inode when new files are created. The MD5 digest value for the test filesystem, scenario6.ext2, did not change after running debugfs on the filesystem, proving that debugfs does not modify the filesystem when used as described for this testing to recover deleted files.

### 2.7.7 Scenario 7

Scenario 7 involved creating six files, deleting all six of the files, creating one large file to fill the filesystem, and then attempting to recover any of the six deleted files. Following is the full output from the script, which captured the entire scenario 6 test.

```
Script started on Sun Aug 17 22:37:57 2003
[root@localhost recover_eval]# cd filesystems
[root@localhost filesystems]# dd if=/dev/zero of=scenario7.ext2 bs=100k count=500
500+0 records in
500+0 records out
[root@localhost filesystems]# mke2fs scenario7.ext2
mke2fs 1.33 (21-Apr-2003)
```

```
scenario7.ext2 is not a block special device.
Proceed anyway? (y,n) y
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
12544 inodes, 50000 blocks
2500 blocks (5.00%) reserved for the super user
First data block=1
7 block groups
8192 blocks per group, 8192 fragments per group
1792 inodes per group
Superblock backups stored on blocks:
                                    8193, 24577, 40961

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

This filesystem will be automatically checked every 36 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
[root@localhost filesystems]# mkdir /mnt/scenario7
[root@localhost filesystems]# mount -o loop scenario7.ext2 /mnt/scenario7
[root@localhost filesystems]# mount
/dev/sda2 on / type ext3 (rw)
none on /proc type proc (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/sda1 on /boot type ext3 (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
/root/GCFA/recover_eval/filesystems/scenario7.ext2 on /mnt/scenario7 type ext2
(rw,loop=/dev/loop0)
[root@localhost filesystems]# ls -la /mnt/scenario7
total 17
drwxr-xr-x    3 root     root         1024 Aug 17 22:38 .
drwxr-xr-x   12 root     root         4096 Aug 17 22:39 ..
drwx------    2 root     root        12288 Aug 17 22:38 lost+found
[root@localhost filesystems]# cp ../bashfile /mnt/scenario7
[root@localhost filesystems]# cp ../catfile /mnt/scenario7
[root@localhost filesystems]# cp ../grepfile /mnt/scenario7
[root@localhost filesystems]# cp ../lsfile /mnt/scenario7
[root@localhost filesystems]# cp ../psfile /mnt/scenario7
[root@localhost filesystems]# cp ../rmfile /mnt/scenario7
[root@localhost filesystems]# ls -la /mnt/scenario7
total 3089
drwxr-xr-x    3 root     root         1024 Aug 17 22:40 .
drwxr-xr-x   12 root     root         4096 Aug 17 22:39 ..
-rw-r--r--    1 root     root      3064910 Aug 17 22:40 bashfile
-rw-r--r--    1 root     root         2346 Aug 17 22:40 catfile
-rw-r--r--    1 root     root        26558 Aug 17 22:40 grepfile
drwx------    2 root     root        12288 Aug 17 22:38 lost+found
-rw-r--r--    1 root     root         8713 Aug 17 22:40 lsfile
-rw-r--r--    1 root     root        22395 Aug 17 22:40 psfile
-rw-r--r--    1 root     root         2807 Aug 17 22:40 rmfile
[root@localhost filesystems]# sync
[root@localhost filesystems]# rm /mnt/scenario7/*file
rm: remove regular file `/mnt/scenario7/bashfile'? y
rm: remove regular file `/mnt/scenario7/catfile'? y
rm: remove regular file `/mnt/scenario7/grepfile'? y
rm: remove regular file `/mnt/scenario7/lsfile'? y
rm: remove regular file `/mnt/scenario7/psfile'? y
rm: remove regular file `/mnt/scenario7/rmfile'? y
[root@localhost filesystems]# sync
[root@localhost filesystems]# ls -la /mnt/scenario7
total 17
drwxr-xr-x    3 root     root         1024 Aug 17 22:41 .
drwxr-xr-x   12 root     root         4096 Aug 17 22:39 ..
drwx------    2 root     root        12288 Aug 17 22:38 lost+found
[root@localhost filesystems]# tar cf /mnt/scenario7/bigfile.tar /usr/lib
tar: Removing leading `/' from member names
tar: /mnt/scenario7/bigfile.tar: Wrote only 4096 of 10240 bytes
tar: Error is not recoverable: exiting now
```

```
[root@localhost filesystems]# ls -la /mnt/scenario7
total 48411
drwxr-xr-x    3 root     root         1024 Aug 17 22:42 .
drwxr-xr-x   12 root     root         4096 Aug 17 22:39 ..
-rw-r--r--    1 root     root     49360896 Aug 17 22:42 bigfile.tar
drwx------    2 root     root        12288 Aug 17 22:38 lost+found
[root@localhost filesystems]# sync
[root@localhost filesystems]# umount /mnt/scenario7
[root@localhost filesystems]# md5sum scenario7.ext2
e6911bde783a84ef20d042d68240b269  scenario7.ext2
[root@localhost filesystems]# debugfs scenario7.ext2
debugfs 1.33 (21-Apr-2003)
debugfs:  lsdel
 Inode  Owner  Mode    Size    Blocks   Time deleted
0 deleted inodes found.
debugfs:  ls
 2  (12) .    2  (12) ..    11  (20) lost+found    12  (980) bigfile.tar
debugfs:  ls -l
      2   40755 (2)      0      0    1024 17-Aug-2003 22:42 .
      2   40755 (2)      0      0    1024 17-Aug-2003 22:42 ..
     11   40700 (2)      0      0   12288 17-Aug-2003 22:38 lost+found
     12  100644 (1)      0      0   49360896 17-Aug-2003 22:42 bigfile.tar

debugfs:  q
[root@localhost filesystems]# md5sum scenario7.ext2
e6911bde783a84ef20d042d68240b269  scenario7.ext2
[root@localhost filesystems]# chmod 444 scenario7.ext2
[root@localhost filesystems]#
Script done on Sun Aug 17 22:46:29 2003
```

The results that debugfs was not able to recover any of the deleted files because debugfs reported that there were no deleted inodes present. This test demonstrated that even though six deleted inodes were created and only one reused, the process of filling up the filesystem with the one large file somehow erased or corrupted information debugfs needs to identify deleted inodes. Perhaps the deleted inode information was still present but since all of the data blocks the deleted inode used were now allocated to another file, debugfs simply did not report any deleted inodes. The test does not indicate exactly what happened to the five deleted inodes that were not reused. This may imply, but was not verified, that debugfs will only report a deleted inode if the inode's associated data blocks have not been reused. The MD5 digest value for the test filesystem, scenario7.ext2, did not change after running debugfs on the filesystem, proving that debugfs does not modify the filesystem when used as described for this testing to recover deleted files.

### 2.7.8 Re-run of All Tests

To demonstrate the repeatability of the test procedure and results, the test was re-run as described in section 2.5.5. All results were exactly the same as with the first run. The files from the test were preserved in the directory /root/GCFA/recover_eval/run2.

## 2.8 Analysis

Based on the results of the testing above, the use of debugfs to restore deleted files in a Linux ext2 filesystem will work reliably with files of varying size as long as the inode or data blocks from the delete file have not been reused.  As shown by debugfs's ability to dump the full contents of the deleted inode from the 3MB file, debugfs seems to be able to track down the indirect data blocks for files over 12KB in size.  Debugfs, however, does not seem capable of recovering any data if the inode from the deleted file has been reused.  In this case, the critical information in the inode has been overwritten, even though some of the deleted file's data blocks may still be contain data from the file. Without the file's original inode information, there is no easy way to track down the blocks from the deleted file.

Testing also showed that if the data blocks associated with a deleted inode are used by another file, the deleted inode will not be reported by debugfs.

Testing showed that the first inode available for use for new files on a fresh filesystem was inode number 12.  Testing also showed that when files were deleted and new files created, the lowest numbered free or deleted inode was used for the new file and the system did not give preference to fresh empty inodes over deleted ones.

Testing also confirmed that when used to located and dump the contents of deleted inodes, debugfs is in read-only mode and does not change the file system in any way. This was verified by comparing the MD5 digest values of the filesystem file before and after debugfs was run on it.

With the limitations discussed above in mind, debugfs can be a valuable tool to a forensic investigator as a quick and convenient way to identify and recover lost files from a filesystem without contaminating the filesystem.  Debugfs would be a good tool to use in the field since most Linux systems already have debugfs installed, or debugfs and its libraries can be placed on CDROM and made portable.  Debugfs complements the more advanced filesystem analysis tools available as part of the TCT (The Coroner's Toolkit) and TASK (The @stake Sleuth Kit).  All the investigator has to do is to attach debugfs to the filesystem, issues the "lsdel" command to identify all deleted inodes, and then use the "dump" command to copy out the deleted file's contents to another file outside of the filesystem being investigated.  Everything about the deleted file can be recovered except the name of the file, which is stored separately in a directory entry.  The investigator can then examine the recovered file to identify what it was and perhaps discover more clues in the investigation.

## 2.9 Presentation

There are two important outputs generated by debugfs.  The first output is that of the lsdel command.  The lsdel command lists all deleted inodes in the filesystem.  Lsdel displays important information about the deleted file, such as the date and time of its deletion, the user id of the owner, the inode number, and the deleted file's size.  Here is an example from test scenario 5 above.

```
debugfs:  lsdel
 Inode  Owner  Mode    Size    Blocks   Time deleted
    12      0 100644 3064910 3007/3007 Sun Aug 17 15:17:25 2003
    14      0 100644  26558    27/  27 Sun Aug 17 15:17:31 2003
    17      0 100644   2807     3/   3 Sun Aug 17 15:17:41 2003
```

Perhaps the most important information, other than the inode number that is needed to recover the file, is the time of deletion. If there were many deleted files in a filesystem, the deleted time would help the investigator narrow down the list to those files that were deleted during the time the incident under investigation was happening. In this way the investigator can restore the most critical files to the investigation first. For presentation as formal evidence, as in a court of law, the output of debugfs should be carefully recorded and some sort of signature (digital or pen and ink on a printed copy) applied to it so the authenticity of the information can be established.

Additional hints as to a deleted file's name may be found by using the debugfs "ls -d" command to list the contents of a directory including deleted file names still stored in the directory inode. To do this, the debugfs "ls -d" command is used. For example, performing "ls -d" on the scenario7.ext2 filesystem file at the end of the test returns this information.

```
debugfs: ls -d
2 (12) .  2 (12) ..  11 (20) lost+found  12 (980) bigfile.tar
<0> (16) grepfile  <0> (16) lsfile  <0> (16) psfile  <0> (900) rmfile
```

Note that only four of the six deleted filenames appear in the" ls -d" listing above. This demonstrates that this method is not foolproof to show deleted file names if activity has taken place on the filesystem after deletion. The first number to the left of the file or directory name is the inode number. For the current directory of the filesystem (the "." directory), the inode number is 2.

It is important to keep in mind that the deletion time given by debugfs is interpreted by the system on which debugfs is being run and is dependent on the time zone setting of that system. The actual time stored in an inode is dependent on how accurately the time was set on the system that housed the filesystem at the time the inode was updated. Fortunately this information can usually be taken into account easily and the time adjustment explained as part of the formal evidence.

The other important output of debugfs used in the testing is the dumped file's contents. As shown in the testing, all the investigator has to do is provide a name and location for the deleted file to be dumped to and debugfs does the rest. Once restored, the file can be analyzed for evidence to determine what the file is and what role it may have played in the incident. The limitation of reused data blocks discussed in section 2.7 must be kept in mind, however. Based on the testing above, a completely reliable file restoration can only be had if there has been no activity on the deleted file's filesystem that touched the deleted file's inode or data blocks since the file has been deleted.

It is a good idea to make an MD5 digest value of all restored files so that their integrity can always be assured. Any further analysis on restored files should be done on a copy of the file, not the original restored file. The copy can always be verified as being identical to the original based on MD5 digest values.

### 2.10 Conclusion

This test provided proof that the debugfs tool can reliably identify and recover deleted files from an ext2 filesystem to an external location as long as filesystem activity has not overwritten the deleted file's inode or data blocks. Debugfs is present on many Linux installations by default and can also be placed on CROM or other removable media, along with it's necessary libraries to minimize dependency on files from an untrusted system under analysis. Deleted files of interest can be identified when their reported deletion times correspond to a known period of suspicious activity. It was also proven that debugfs does not make any changes to the filesystem from which the deleted files are being recovered.

In the testing, the "lsdel" and "dump" functions of debugfs were successfully used to identify and recover deleted files. The use of these commands is relatively simple and the data they present is immediately usable to a forensic investigator. Comparison of MD5 digest values of the test filesystem before and after being analyzed by debugfs proved that the use of debugfs as described in the test procedures does not change the filesystem in any way. This is of great importance in preserving evidence during an investigation.

Debugfs is designed to be a low-level filesystem debugging tool and not a pure system forensics tool. Debugfs overall is thus quite complex, with many functions that can report on and actively manipulate a filesystem. If a pure forensics approach is desired from a file recovery tool, the filesystem analysis tools of the TASK and Autopsy forensics tool kits are perhaps better suited. However, if nothing else is available in a pinch, the debugfs program can be used to identify and recover deleted files in a reliable and repeatable manner.

## Part 3 – Legal Issues of Incident Handling

For this section, it is assumed that the findings from Part 1 of this practical show that John Price was distributing copyrighted material on publicly available systems. Based on this scenario, the following questions will be answered based on United States and Maryland law.

### 3.1 Based on the type of material John Price was distributing, what, if any, laws have been broken based on the distribution?

### 3.1.1 What is a Copyright?

Copyrights are designed to protect the expression of ideas, but not the ideas themselves. Expressions of ideas include creative works such as poems, stories, and photographs. They might also include computer programs. A copyright protects an individual's right to make a living from their creations while maintaining that ideas are free to exchange for the better good of society. Copyright law allows for "fair use" of a copyrighted work for purposes such as criticism, comment, news reporting, and teaching. The principle is that the copyright upholds the creator's right to compensation for use of the work, while allowing others to use the ideas that are expressed.

Only the creator of the material has the right to make and sell copies to the public. It's important to note that copyright law intends that the copyrighted material be made available for sale or distribution to the public; a copyright is not intended to protect material that the creator desires to keep secret. By definition, for material to be copyrighted, the material must have a determinable creator and must be intended for distribution to the public. Copyrighted work must also be original. General works that are public knowledge cannot be copyrighted. For example, no one could copyright the song "Yankee Doodle Dandee" itself, but a particular performance or arrangement of the song could be copyrighted. Works such as Yankee Doodle Dandee are considered to be in the *public domain*. An infringement of copyrighted material can be prosecuted in court. The owner of the copyright must prove that the work was copied and that the copying caused substantial loss to the owner. If some other person came up with the same work independently, it would not be a violation of the copyright.

The reference http://www.umuc.edu/library/copy.html#whatc provides a good discussion of general copyright definitions and issues.

### 3.1.2 Copyright Law Relating to Computer Crime

This section discusses the law as it relates to John Price's situation. The only information presented on Mr. Price's alleged crime is that he was distributing copyrighted material on publicly available systems. The exact crime and penalties applicable to this situation depend on many factors including the dollar amount of the loss incurred by the copyright holders and the number of illegal copies that were created. The following are some laws that could apply to Mr. Price's situation.

**Title 17 U.S.C. Section 506**

Title 17 of the U.S. Code (U.S.C.) is also known as the Copyright Act. Title 17 defines copyright law and the penalties for violating that law. Section 506(a) of Title 17 was added as part of the 1976 general Copyright Revision Act and requires that to be criminal, a copyright violation, including illegally making copies and distributing copyrighted material, must be done "willfully and for purposes of commercial advantage or private financial gain." (http://www.cybercrime.gov/CFAleghist.htm). Title 17 also defines as criminal, "the reproduction or distribution, including by electronic means, during any 180-day period, of 1 or more copies or phonorecords of 1 or more copyrighted works, which have a total retail value of more than $1,000". Notes that Title 17 U.S.C Section 506(a) defines the crime, but Title 18 U.S.C defines the penalties.

The actual text of Section 506 is not very long, and is included here as obtained from the Web site http://www.cybercrime.gov/17usc506.htm.

**§ 506. Criminal offenses**

(a) Criminal Infringement.--Any person who infringes a copyright willfully either–

(1) for purposes of commercial advantage or private financial gain, or

(2) by the reproduction or distribution, including by electronic means, during any 180-day period, of 1 or more copies or phonorecords of 1 or more copyrighted works, which have a total retail value of more than $1,000,

shall be punished as provided under section 2319 of title 18, United States Code. For purposes of this subsection, evidence of reproduction or distribution of a copyrighted work, by itself, shall not be sufficient to establish willful infringement.

(b) Forfeiture and Destruction.--When any person is convicted of any violation of subsection (a), the court in its judgment of conviction shall, in addition to the penalty therein prescribed, order the forfeiture and destruction or other disposition of all infringing copies or phonorecords and all implements, devices, or equipment used in the manufacture of such infringing copies or phonorecords.

(c) Fraudulent Copyright Notice.--Any person who, with fraudulent intent, places on any article a notice of copyright or words of the same purport that such person knows to be false, or who, with fraudulent intent, publicly distributes or imports for public distribution any article bearing such notice or words that such person knows to be false, shall be fined not more than $2,500.

(d) Fraudulent Removal of Copyright Notice.--Any person who, with fraudulent intent, removes or alters any notice of copyright appearing on a copy of a copyrighted work shall be fined not more than $2,500.

(e) False Representation.--Any person who knowingly makes a false representation of a material fact in the application for copyright registration provided for by section 409, or in any written statement filed in connection with the application, shall be fined not more than $2,500.

(f) Rights of Attribution and Integrity.--Nothing in this section applies to infringement of the rights conferred by section 106A(a).

Based on the text of Title 17 U.S.C. Section 506 above, more details on Mr. Price's activities would need to be known to understand exactly how this law would apply. For example, it would need to be shown that Mr. Price was profiting from his activities or that the retail value of what he was distributing was greater than $1,000. If neither of these situations were true, then Mr. Price may not have broken this law.

The letter to someone named Mike found on the floppy disk image mentioning "batch of files" and "advance orders" could possibly be used to support the claim that Mr. Price was collaborating with someone else in a somewhat organized process for taking orders and distributing material to satisfy those orders. This may help to satisfy condition (1) of Section 506(a).

## Title 18 U.S.C. Section 2319

Title 18 U.S.C. Section 2319 defines the felony penalties for violating Title 17 U.S.C. Section 506(a). There are two penalties defined, one for violating 17 U.S.C. Section 506(a)(1), and one for violating Section 506(a)(2). Within each section the penalties vary depending on the number of illegal copies made and the retail value of the copies.

Following is the actual text of 18 U.S.C. Section 2319 from the Web site http://www.cybercrime.gov/18usc2319.htm.

**§ 2319. Criminal Infringement of a Copyright**

(a) Whoever violates section 506(a) (relating to criminal offenses) of title 17 shall be punished as provided in subsections (b) and (c) of this section and such penalties shall be in addition to any other provisions of title 17 or any other law.

(b) Any person who commits an offense under section 506(a)(1) of title 17--

(1) shall be imprisoned not more than 5 years, or fined in the amount set forth in this title, or both, if the offense consists of the reproduction or distribution, including by electronic means, during any 180-day period, of at least 10 copies or phonorecords, of 1 or more copyrighted works, which have a total retail value of more than $2,500;

(2) shall be imprisoned not more than 10 years, or fined in the amount set forth in this title, or both, if the offense is a second or subsequent offense under paragraph (1); and

(3) shall be imprisoned not more than 1 year, or fined in the amount set forth in this title, or both, in any other case.

(c) Any person who commits an offense under section 506(a)(2) of title 17, United States Code--

(1) shall be imprisoned not more than 3 years, or fined in the amount set forth in this title, or both, if the offense consists of the reproduction or distribution of 10 or more copies or phonorecords of 1 or more copyrighted works, which have a total retail value of $2,500 or more;

(2) shall be imprisoned not more than 6 years, or fined in the amount set forth in this title, or both, if the offense is a second or subsequent offense under paragraph (1); and

(3) shall be imprisoned not more than 1 year, or fined in the amount set forth in this title, or both, if the offense consists of the reproduction or distribution of 1 or more copies or phonorecords of 1 or more copyrighted works, which have a total retail value of more than $1,000.

(d)

(1) During preparation of the presentence report pursuant to Rule 32(c) of the Federal Rules of Criminal Procedure, victims of the offense shall be permitted to submit, and the probation officer shall receive, a victim impact statement that identifies the victim of the offense and the extent and scope of the injury and loss suffered by the victim, including the estimated economic impact of the offense on that victim.

(2) Persons permitted to submit victim impact statements shall include--

(A) producers and sellers of legitimate works affected by conduct involved in the offense;
(B) holders of intellectual property rights in such works; and
(C) the legal representatives of such producers, sellers, and holders.

(e) As used in this section--

(1) the terms "phonorecord" and "copies" have, respectively, the meanings set forth in section 101 (relating to definitions) of title 17; and

(2) the terms "reproduction" and "distribution" refer to the exclusive rights of a copyright owner under clauses (1) and (3) respectively of section 106 (relating to exclusive rights in copyrighted works), as limited by sections 107 through 120, of title 17.

Again, based on Title 18 Section 2319, the penalties that Mr. Price might face depend on the retail value of any illegal copies he made, and on the number of illegal copies he made. Penalties are more severe if Mr. Price has had any previous convictions under this law.

**Title 17 U.S.C. Sections 1201 - 1205 (Digital Millennium Copyright Act)**

The Digital Millennium Copyright Act (DMCA) deals with the circumvention of copyright protection systems. Specifically, DMCA makes it generally illegal to "circumvent a technological measure that effectively controls access to a work", with several exceptions defined in the text of the law. Section 1201 is of the most interest to this case and can be found at the Web site http://www.cybercrime.gov/17usc1201.htm.

The DMCA specifically discusses issues such as breaking encryption or copy protection that is designed to prevent the illegal copying of copyrighted material. For Mr. Price to be guilty of violating the DMCA there would have to be evidence that he was breaking copy protection schemes in order to provide access to the copies of the material he was distributing. No evidence of any such activity was found based on the investigation of the floppy image, but this is an area worthy of further investigation.

**Title 18 U.S.C. Section 2318**

Title18 U.S.C Section 2318 deals with the "Trafficking in Counterfeit Labels for Phonorecords, Copies of Computer Programs or Computer Program Documentation or Packaging, and Copies of Motion Pictures or Other Audio Visual Works, and Trafficking in Counterfeit Computer Program Documentation or Packaging." Of interest here is the trafficking in "Copies of Motion Pictures or Other Audio Visual Works". Several HOWTO files were found on the floppy disk image that relate to multimedia on Linux. This could be used to show an interest on Mr. Price's part in video and audio files. This could include things such as movies and songs. If Mr. Price were distributing illegal copies of movies or songs, then this law would apply.

Section 2318 specifically addresses the issue of "trafficking", which, as defined in the law, means "to transport, transfer or otherwise dispose of, to another, as consideration for anything of value or to make or obtain control of with intent to so transport, transfer or dispose of". This law would cover the "distribution" aspect of Mr. Price's activities. As possible evidence of distribution, the netcat program was found on the floppy image that was analyzed in Part 1 of this practical. Netcat can be used as a simple and flexible way to transfer files between systems. Netcat makes it very easy to manipulate network port numbers used and works has very little overhead in transferring files. This makes netcat a good choice when covert file transfers or system access is desired and more standard means of file transfer (such as FTP) are blocked or controlled on the network.

The penalties for someone violating Section 2318 include being “fined under this title or imprisoned for not more than five years, or both”.

The text of Title 18 U.S.C. Section 2318 is available at http://www.cybercrime.gov/18usc2318.htm.

**The No Electronic Theft (“NET”) Act**

The NET Act was signed into Law on 16 December 1997 as H.R. 2265. This law amended the criminal copyright and trademark provisions of Titles 17 and 18 discussed above. The amendment includes making it a crime to perform large-scale illegal reproduction of copyrighted material even if there is no discernible profit motive. The definition “financial gain” in Title 17 was expanded to include “, or expectation of receipt, of anything of value, including the receipt of other copyrighted works” so that those who conduct illegal business by using barter rather than cash are explicitly covered under the law. The NET Act also clarifies that “reproduction or distribution” explicitly includes performing such acts by electronic means. The NET Act could obviously apply to Mr. Price’s situation, and would provide coverage if there were no evidence of a profit motive found. More information on the NET Act can be found at http://www.cybercrime.gov/netsum.htm.

**Case Law**

Following are some examples of actual convictions based on the laws discussed above.

On 15 December, 2000 Jason Everett Spatafore of Phoenix, AZ, pled guilty to one count of violating 17 U.S.C. § 506(a)(2) and 18 U.S.C. § 2319(c)(3). The defendant admitted that he “willfully infringed a copyright by reproducing and distributing by electronic means copies of parts of the film Star Wars Episode I: The Phantom Menace.” Mr. Spatafore posted copies of parts of the film on various Web sites and encouraged others to download the material. More information on this case can be found at http://www.cybercrime.gov/spataforeplea.htm.

On 20 August 1999, Jeffrey Levy was convicted of violating the NET Act. Mr. Levy pled guilty to “illegally posting computer software programs, musical recordings, entertainment software programs, and digitally-recorded movies on his Internet web site, allowing the general public to download these copyrighted products.” More information on this case can be found at http://www.cybercrime.gov/iplaws.htm#IIc.

More information on court cases relating to computer crime and intellectual property can be found at http://www.cybercrime.gov/ipcases.htm.

## 3.2 What would the appropriate steps be to take if you discovered this information on your systems? Cite specific statutes.

Hopefully an organization facing such a situation would have an incident policy covering the appropriate actions to take. This policy would describe what to do and who to contact when a computer security incident takes place. Finding illegally copied copyrighted files on a computer is indeed a computer incident and the evidence must be treated carefully if any prosecution is to take place. Standard incident handling techniques should be followed.

One of the first things to do would be to preserve the evidence. This would involve protecting the evidence from access by others, especially those potentially guilty of creating it. Some options to preserve the evidence might include taking the system or storage containing the evidence offline and/or creating an image of the hard disk using a tool such as “dd” or Norton Ghost. An MD5 digest value should be generated for the disk image as a digital signature and the original copy of the image should be protected along with the original storage media if possible. The chain of custody should be fully documented for all evidence.

Also of interest at this time is to analyze and preserve the system and network logs for signs of suspicious activity, such as file transfers of unusual size or to unusual destinations. This information could help to narrow down the accounts and computers used to transfer and store the illegal files.

In most cases, the company management and legal staff would be contacted for instructions on how to proceed. In cases where such a staff does not exist, law enforcement might be contacted directly.

Such laws as the Wiretap Act, the Pen/Trap Statute, and the Electronic Communications Privacy Act (ECPA) define the legal aspects of finding and gathering evidence on a system. The Wiretap Act and the Pen/Trap Statute involve the real-time monitoring of information. The ECPA covers accessing and disclosing stored communications such as files on a computer. In the case presented here of finding files on a computer, the most applicable law would be the ECPA. The ECPA defines privacy rights for customers and subscribers of a computer network service (a publicly accessible service). If a file was found on a computer owned and used by a private company, that company would have full rights to disclose the information to law enforcement. If the company were in the business of providing network service to the public at large, then it would be restricted as to what it can release. In the case of John Price, it was defined that the illegal files were being distributed on publicly accessible systems. It can be implied in this scenario that Mr. Price was using a private company computer to copy the illegal files out to publicly accessible computers. In this case, his employer could make the evidence found on the company computer available to law enforcement.

The ability of the private company to gather such information on Mr. Price would be legal under the ECPA because the ECPA privacy protections do not apply to privately owned computers.

More information on these laws and on the legal issues of gathering evidence can be found at http://www.cybercrime.gov/s&smanual2002.htm.

## 3.3 In the event your corporate counsel decides to not pursue the matter any further at this point, what steps should you take to ensure any evidence you collect can be admissible in proceedings in the future should the situation change?

The most important thing to do to ensure that any evidence will be admissible in future proceedings would be to carefully record all information, create MD5 digest signatures for all files, and to carefully record the chain of custody for each piece of evidence. Also, store all copies of the evidence, and especially the original evidence, in a secure location along with all written testimony and records created during the investigation. There are no officially sanctioned standards for non-law enforcement collection of evidence. It is mostly a matter of following good system incident handling and forensics practices such as those defined in the SANS "Computer Security Incident Handling" step-by-step guide. More information on this guide can be found at https://store.sans.org/.

Success in the courtroom depends on having strong evidence of the illegal activity and on proving that the evidence is reliable and has not been tampered with.

## 3.4 How would your actions change if your investigation disclosed that John Price was distributing child pornography?

If Mr. Price was found to be distributing child pornography and such files were found on the company computer, Mr. Price would be facing much more severe charges than those relating to copyright violation. Title 18 Section 2252 of the U.S.C. defines the definitions and penalties for conducting "certain activities relating to material involving the sexual exploitation of minors". This law defines it as a criminal violation if any person knowingly transports or ships in interstate or foreign commerce by any means including by computer or mails any depiction of a minor engaging in sexually explicit conduct. This would certainly include Mr. Price's use of the netcat and bmap tools to transport and store such files using company computers and the Internet. The penalties include fine and up to 15 years in prison or both.

One potential difference in the evidence itself is that with illegal copies of copyrighted material, it is the copy that is illegal, not the original material itself. With child pornography, it is the original material (the image) itself that is illegal. But it is unclear how this distinction might impact the response of the incident handler and forensics investigator.

Since child pornography potentially involves the safety and exploitation of the victim children, law enforcement and company management are generally very interested and will to put forth the necessary resources to catch and prosecute individuals who violate this law. Other than this motivational element, the handling of the evidence should not be any different than for handling other types of incident such as finding illegal copies of copyrighted material. Both types of files are illegal and both are potential evidence for a court of law. The preservation of the evidence and the chain of custody are of prime importance if any case is to be prosecuted.

An important difference in the incident response, though, may involve law or policy that requires mandatory reporting of the discovery of child pornography, where as in the case of the illegal copies of copyrighted material, no information on mandatory reporting requirements could be found. Title 42 U.S.C., Chapter 132, Subchapter IV, Section 13032 available at http://www4.law.cornell.edu/uscode/42/13032.html requires mandatory report of child pornography by electronic communication service providers. In the case presented in this practical, this law would not seem to apply since the information found seems to have been present on the employers computer, not as part of an account with a communication service provider.

Section 5-704 and Section 5-705 of the Maryland Code, Title 5, Subtitle 7 require that any person who has reason to believe that a child has been subjected to neglect or abuse must notify the appropriate law enforcement agency or local department. Child pornography could be considered abuse and thus if the incident happened in Maryland, it would have to be reported to law enforcement authorities "by telephone or direct communication, as soon as possible" as defined in Section 5-704.

Maryland Code can be found at the Web site http://198.187.128.12/maryland/lpext.dll?f=templates&fn=fs-main.htm&2.0.

An interesting article about the pros and cons regarding mandatory reporting can be found at http://www.avn.com/print/articles/12212.html.

In summary, the identification of child pornography would invoke mandatory reporting of the incident to law enforcement as soon as practical, while the finding of illegally copied copyrighted material would not require mandatory reporting to law enforcement.

## REFERENCES

- **Bmap**
    - Chuvakin, Anton, Ph. D. "Linux Data Hiding and Recovery." 10 March 2002. URL: http://www.linuxsecurity.com/feature_stories/data-hiding-forensics.html (Accessed 8/2/2003)
    - Source code download site. URL: ftp://ftp.scyld.com/pub/forensic_computing/bmap
- **MD5**
    - URL: ftp://coast.cs.purdue.edu/pub/tools/unix/crypto/md5/ (Accessed 8/2/03)
- **Steps in Incident Handling**
    - SANS Institute. Incident Handling Step-by-Step and Computer Crime Investigation. SANS Institute, SANS Pacific Northwest 2002 Conference, 2002. 2-5.
- **VMware**
    - VMware Web site. URL: http://www.vmware.com (Accessed 8/2/03)
- **Linux Multimedia**
    - Xmms Red Hat package download site. URL: http://havardk.xmms.org/dist/xmms-1.2.7-rh8-rh9-rpm/ (Accessed 8/2/03)
    - Linux Video Project Web site. URL: http://www.linuxvideo.org/ (Accessed 8/3/03)
- **Computer Use Policy Examples**
    - Devzone Web. "Security Policy Template." URL: http://www.devzonedevelopment.com/securitypolicytemplate.html (Accessed 8/8/03)
    - "BSA Corporate Software Policy regarding the use of personal computer software." URL: http://www.mercynet.edu/IT/ethics/modules/CorporateSoftwarePolicy.htm (Accessed 8/8/03)
- **Ext2 Filesystem**
    - Crane, Aaron. "Linux Ext2fs Undeletion mini-HOWTO." V1.3 2 February 1999. URL: http://www.praeclarus.demon.co.uk/tech/e2-undel/html/howto.html (Accessed 8/10/03)
    - "Recover homepage (v1.3c)." URL: http://recover.sourceforge.net/linux/recover/ (Accessed 8/10/03)
    - Card, Remy; Ts'o, Theodore; Tweedie, Stephen. "Design and Implementation of the Second Extended Filesystem." URL: http://e2fsprogs.sourceforge.net/ext2intro.html (Accessed 8/10/03)
    - "Ext2fs Home Page". URL: http://e2fsprogs.sourceforge.net/ext2.html (Accessed 8/10/03)
    - Werner, Torsten. "Undelete for the ext2 filesystem (version 0.3)." 20 April 2003. URL: http://twerner.debian.net/ (Accessed 8/14/03)
    - Source Forge. "Welcome to the E2fsprogs Sourceforge Page." URL: http://e2fsprogs.sourceforge.net/ (Accessed 8/10/03)

- **Computer Crime Law**
  - “Computer Crime and Intellectual Property Section (CCIPS).” URL: http://www.cybercrime.gov/iplaws.htm#IIc (Accessed 8/18/03)
  - “Intellectual Property Cases.” URL: http://www.cybercrime.gov/ipcases.htm (Accessed 8/18/03)
  - UMUC. “Copyright and Fair Use in the Classroom, on the Internet, and the World Wide Web.” 2003. URL: http://www.umuc.edu/library/copy.html#whatc (Accessed 8/18/03)
  - “Criminal Offenses”, Title 17 U.S.C. Section 506. 22 June 2000. URL: http://www.cybercrime.gov/17usc506.htm (Accessed 8/18/03)
  - “18 U.S.C. 2319 Criminal Infringement of a Copyright.” 22 June 2000. URL: http://www.cybercrime.gov/18usc2319.htm (Accessed 8/18/03)
  - “17 U.S.C. 1201 Circumvention of Copyright Protection Systems.” 22 June 2000. URL: http://www.cybercrime.gov/17usc1201.htm (Accessed 8/18/03)
  - “18 U.S.C. 2318.” 22 June 2000. URL: http://www.cybercrime.gov/18usc2318.htm (Accessed 8/18/03)
  - “The No Electronic Theft (“NET”) Act.” 18 February 1998. URL: http://www.cybercrime.gov/netsum.htm (Accessed 8/18/03)
  - “US CODE COLLECTION Sec. 2252. – Certain activities relating to material involving the sexual exploitation of minors.” URL: http://www4.law.cornell.edu/uscode/18/2252.html (Accessed 8/18/03)
  - “Title 42 Section 13032 – Reporting of child pornography by electronic communication service providers” URL: http://www4.law.cornell.edu/uscode/42/13032.html (Accessed 8/18/03)
  - Lexis Nexis (Maryland Code) URL: http://198.187.128.12/maryland/lpext.dll?f=templates&fn=fs-main.htm&2.0. (Accessed 8/18/03)