Global Information Assurance Certification Paper

An Introduction to Hiding and Finding Data on Linux

GIAC Security Essentials Certification (GSEC)
Practical Assignment Version 1.4b - Option 1
Gary Robertson

Table of Contents

Abstract

This paper provides an introduction to several of the common techniques for hiding data on Linux systems (specifically those using the ext2 file system), as well as some methods for finding hidden data on these systems. The techniques of hiding directories, camouflaging files, deleting files, unlinking open files and using file slack space are explained with a focus on simple, step-by-step examples. For each of these data hiding techniques there is a corresponding section that provides an example-led approach to finding the data. Aside from using Linux commands to find hidden data, some basic computer forensics techniques are presented using the open source tools of The Sleuth Kit and its companion browser-based interface, Autopsy.

Introduction

The techniques outlined in this paper are aimed at educating beginners in the fields of system administration or computer forensics as to how data may be hidden on systems, and how one may go about finding this data. The paper is not meant to be a guide for real-world investigations. In particular, the data finding sections do not represent best practice in the area of computer forensics. For the sake of simplicity the forensic images are copied onto the same system that houses the "evidence", thereby breaking one of the basic rules of computer forensics (Shinder, p.552). Nevertheless, the example-led approach to explaining the techniques should provide a valuable starting point for people interested in data hiding on Linux systems.

This paper does not cover two important methods of data hiding, encryption and steganography. Both of these methods rely on specific algorithms, which often vary from application to application. They are not methods that are peculiar to Linux or other UNIX-like operating systems, and for this reason they have been excluded from the discussion.

All of the examples in this paper have been carried out on a system running Red Hat Linux version 8.0, using the ext2 file system rather than the default ext3 file system. Except where noted, all of the data hiding techniques have been performed by user "sans", whose home directory is /home/sans. All of the techniques for finding hidden data have been performed by the superuser, "root". The Linux examples will often show a command prompt that includes the user name, the machine name (always "tortilla") and the current directory name. For example, a prompt such as "[root@tortilla /]#" indicates the superuser working in the base directory of the system. The final character in the prompt will always be a "#" for the superuser and a "$" for the user "sans".

Forensic Software Installation

Some of the demonstrations for finding hidden data require the use of forensic software tools. For this reason a quick discussion of these tools is provided before starting on the demonstrations.

Installation of The Sleuth Kit

The Sleuth Kit is an open source forensic toolkit that has its origins in a group of file system analysis tools by Wietse Venema and Dan Farmer called The Coroner's Toolkit

(TCT). Up until quite recently the toolkit went under the name of TASK, an acronym for The @stake Sleuth Kit. The person now in charge of development is Brian Carrier, with the current version being 1.61. The Web site for The Sleuth Kit is:

http://www.sleuthkit.org/sleuthkit/index.php

To install The Sleuth Kit, download the compressed file from the Web site, unpack it in a directory on your Linux machine (I have created a directory of /usr/local/security for this purpose), and follow the directions in the INSTALL file. Make sure to create a soft link to the directory created during the unpacking by entering a command such as:

```
ln -s sleuthkit-1.61 sleuthkit
```

Installation of Autopsy

Autopsy is a companion program to the Sleuth Kit. Autopsy makes it easier to work with the many tools provided with the Sleuth Kit by providing a graphical interface to those tools in the form of a browser. At the time of writing, the current version of Autopsy is 1.71, and its Web site is:

http://www.sleuthkit.org/autopsy/index.php

The instructions for installing Autopsy are very similar to those for The Sleuth Kit: download the compressed file from the Web site, unpack it in a directory on your Linux machine, and follow the directions in the INSTALL file. As for The Sleuth Kit, I unpacked Autopsy into my /usr/local/security directory and created a soft link to the resultant directory using the `ln` command:

```
ln -s autopsy-1.71 autopsy
```

The Autopsy installation prompts for an "Evidence Locker Directory". This is the base directory for storage of files for forensic investigation. I created a directory of /usr/local/forensics for this purpose.

Hidden Directories

This is a basic method of hiding data that relies on the non-discovery of the directory containing the data. The actual data files are often not disguised in any way; the effort instead going into hiding the directory itself. There are two main approaches to accomplishing this. The first approach involves giving the directory a strange name that may go unnoticed on file listings, whilst the second approach involves creating the directory in a part of the system where it is least likely to be found by a system administrator.

Hiding Data in Hidden Directories

There are a couple of strange directory names that are used over and over by hackers and others who want to conceal data. They are "..." (three dots) and ".. " (two dots and a space). I used these classic directory names to hide data in subdirectories under my /home/sans/dir directory. The commands used to create the directories were:

```
[sans@tortilla dir]$ mkdir ...
[sans@tortilla dir]$ mkdir ".. "
```

I also created a normal directory called "accounts" and placed some Perl script files in all three directories. A simple listing of my /home/sans/dir directory fails to show the hidden directories as they both begin with a dot:

```
[sans@tortilla dir]$ ls -l
total 5
drwxrwxr-x    2 sans     sans         1024 May 21 12:28 accounts
-rw-rw-r--    1 sans     sans           15 May 21 12:27 file1.txt
-rw-rw-r--    1 sans     sans         1568 May 21 12:27 file2.txt
-rw-rw-r--    1 sans     sans          167 May 21 12:27 file3.txt
```

A full listing of the same directory reveals the hidden directories, but they may go unnoticed due to their similarity in the listing with the current and parent directories (represented by a single dot and two dots respectively):

```
[sans@tortilla dir]$ ls -al
total 9
drwxrwxr-x    5 sans     sans         1024 May 21 12:27 .
drwx------    4 sans     users        1024 May 21 12:27 ..
drwxrwxr-x    2 sans     sans         1024 May 21 12:29 ..
drwxrwxr-x    2 sans     sans         1024 May 21 12:28 ...
drwxrwxr-x    2 sans     sans         1024 May 21 12:28 accounts
-rw-rw-r--    1 sans     sans           15 May 21 12:27 file1.txt
-rw-rw-r--    1 sans     sans         1568 May 21 12:27 file2.txt
-rw-rw-r--    1 sans     sans          167 May 21 12:27 file3.txt
```

The other approach to hiding directories, that of creating the directory in a seldomly traversed part of the system, is fairly simple to comprehend, so I will not demonstrate it here. Note, however, that the /dev directory is one of the most frequently used locations to hide other directories (Green, p. 35). This is because the hidden directory can go unnoticed amongst the hundreds of other files and directories in this location.

Finding Data in Hidden Directories

Using Linux Utilities

The find command with various options can be used to clearly show directories which have been hidden by giving them a prefix of ".":

```
[root@tortilla dir]# find /home/sans -type d -name ".*" -print
/home/sans/.kde
/home/sans/dir/...
/home/sans/dir/..
```

The "-type d" option in the above command limits output to directories only. In addition to the two directories that were created earlier, the output shows a directory named ".kde". This directory has been created by the KDE application, and can be ignored for the purposes of this demonstration.

The following command (split over two lines for presentation purposes) not only outputs the directories, but also the data files that reside in these directories. Note that in this command the "-print0" and "-0" are important - without them the shell interprets the directory of ".. " (two dots and a space) as the parent directory ".." (two dots):

```
[root@tortilla dir]# find /home/sans -type d -name ".*" \
-print0 | xargs -0 ls -l
/home/sans/dir/.. :
total 5
-rw-r--r--    1 sans     sans         1070 May 21 12:29 common.pl
-rw-r--r--    1 sans     sans          554 May 21 12:29 edit.pl
-rw-r--r--    1 sans     sans         1039 May 21 12:29 show.pl

/home/sans/dir/...:
total 3
-rw-r--r--    1 sans     sans          342 May 21 12:28 index.pl
-rw-r--r--    1 sans     sans         1147 May 21 12:28 search.pl

/home/sans/.kde:
total 1
drwxr-xr-x    2 sans     users        1024 Jun 30  2003 Autostart
```

A system administrator wishing to periodically check for hidden directories could create a cron job containing a similar `find` command. However, the command would fail to find directories with names that do not begin with a dot, but have instead been hidden in a seldomly traversed part of the system. Therefore perhaps the best method for system administrators to counter the subterfuge of hidden directories is by using a file integrity checker such as Tripwire to report on directory creation.

Camouflaged Files

This is a basic method of hiding data that relies only on the file name. Files containing forbidden data are simply given names implying legitimacy. In particular, the file extension is changed. For example, an employee trying to hide downloaded MP3 audio files or pornography image files under their account may fool a novice system administrator by giving the files innocuous names and changing the file extensions to “.doc”.

Hiding Data with Camouflage

As mentioned above, this technique simply requires changing filenames. I found several image, audio and document files and placed them in my /home/sans/cam directory as shown:

```
[sans@tortilla cam]$ ls -l
total 422
-rw-rw-r--    1 sans     sans       136617 May 21 13:14 beach.jpg
-rw-rw-r--    1 sans     sans        21077 May 21 13:14 book.gif
-rw-rw-r--    1 sans     sans        83260 May 21 13:14 forest.jpg
-rw-rw-r--    1 sans     sans        95668 May 21 13:14 gong.wav
-rw-rw-r--    1 sans     sans        37376 May 21 13:14 report1.doc
-rw-rw-r--    1 sans     sans        19335 May 21 13:14 table.gif
-rw-rw-r--    1 sans     sans        28380 May 21 13:14 train.wav
```

I then changed the filenames of three of the files (beach.jpg, table.gif and train.wav) to

camouflage the data that they contained. All of these camouflaged files were given the prefix of "cam" for demonstration purposes. These image and audio files now appear at first glance to contain word processing documents:

```
[sans@tortilla cam]$ ls -l
total 422
-rw-rw-r--    1 sans     sans        21077 May 21 13:14 book.gif
-rw-rw-r--    1 sans     sans        19335 May 21 13:14 cam_report1.doc
-rw-rw-r--    1 sans     sans       136617 May 21 13:14 cam_report2.doc
-rw-rw-r--    1 sans     sans        28380 May 21 13:14 cam_report3.doc
-rw-rw-r--    1 sans     sans        83260 May 21 13:14 forest.jpg
-rw-rw-r--    1 sans     sans        95668 May 21 13:14 gong.wav
-rw-rw-r--    1 sans     sans        37376 May 21 13:14 report1.doc
```

Finding Data Hidden with Camouflage

Using Linux Utilities

Linux provides a command called `file` that can be used to determine the file type. The man page for this command states that it "uses a combination of file system tests, magic number tests, and language tests" to classify the file. The tests are carried out in that order, with the command terminating on the first successful test. The magic number tests rely on particular file formats containing a consistent binary identifier at the same offset within the file. The output from running the `file` command in my directory containing camouflaged files is:

```
[root@tortilla cam]# file *
book.gif:        GIF image data, version 87a, 640 x 480,
cam_report1.doc: GIF image data, version 87a, 640 x 480,
cam_report2.doc: JPEG image data, JFIF standard 1.02, resolution (DPI), 72 x 72
cam_report3.doc: RIFF (little-endian) data, WAVE audio, Microsoft PCM, 8 bit,
mono 11025 Hz
forest.jpg:      JPEG image data, JFIF standard 1.01, resolution (DPI), 72 x 72
gong.wav:        RIFF (little-endian) data, WAVE audio, Microsoft PCM, 8 bit,
mono 11025 Hz
report1.doc:     Microsoft Office Document
```

The `file` command has identified the three camouflaged files as containing image or audio data rather than document data. It should be noted, however, that the command is not foolproof. For example, it is possible to use a hex editor to alter specific bytes within a file, causing the `file` command to classify it incorrectly. This somewhat advanced technique is outside the scope of this paper.

Using The Sleuth Kit / Autopsy

The Sleuth Kit includes a tool called `sorter` that can be used to find camouflaged files. Although it can be run from the command line, it is easier to run via Autopsy. The brief instructions for accomplishing this are:

1. Create an image of the partition as set out in the appendix "Creating a Forensic Image".
2. Run Autopsy in a browser and create a case and a host name for investigation of the image – my case is called "sans" and its originating host is "tortilla". A symbolic link should be created in a subdirectory of the Autopsy Evidence Locker Directory to link to the forensic image (a HTML form is provided for this). In my example it resulted in

a link from /usr/local/forensics/sans/tortilla/images/dev_hda7.img to the image contained at /usr/local/forensics/dev_hda7.img.

3 From the main Autopsy menu, choose "File Type", then choose the link "Sort Files by Type". Ensure that the only checked box is "Extension and File Type Validation", then choose "OK". The `sorter` command will then be run, with Autopsy providing the full path to an index.html file which will help in analysing the results.

4 Paste the index.html file into another browser window and click on the link to "Extension Mismatches". My browser's output included two of the three camouflaged files:

```
/mnt/forensics/home/sans/cam/cam_report1.doc
  GIF image data, version 87a, 640 x 480 (Ext: doc)
  Image: /usr/local/forensics//sans/tortilla/images/dev_hda7.img Inode: 32130

/mnt/forensics/home/sans/cam/cam_report2.doc
  JPEG image data, JFIF standard 1.02, resolution (DPI), 72 x 72 (Ext: doc)
  Image: /usr/local/forensics//sans/tortilla/images/dev_hda7.img Inode: 32131
```

Note that the `sorter` tool has not reported cam_report3.doc as having an extension mismatch, despite the fact that it is really an audio file with a .wav extension. This is just a configuration issue. By default the `sorter` tool when run under Linux uses the configuration files of default.sort and linux.sort, neither of which lists the .wav extension as a known category. After copying the appropriate rule set from the provided windows.sort file to linux.sort, the output of `sorter` was able to identify all of the camouflaged files:

```
/mnt/forensics/home/sans/cam/cam_report1.doc
  GIF image data, version 87a, 640 x 480 (Ext: doc)
  Image: /usr/local/forensics//sans/tortilla/images/dev_hda7.img Inode: 32130

/mnt/forensics/home/sans/cam/cam_report3.doc
  RIFF (little-endian) data, WAVE audio, Microsoft PCM, 8 bit, mono 11025 Hz
(Ext: doc)
  Image: /usr/local/forensics//sans/tortilla/images/dev_hda7.img Inode: 32132

/mnt/forensics/home/sans/cam/cam_report2.doc
  JPEG image data, JFIF standard 1.02, resolution (DPI), 72 x 72 (Ext: doc)
  Image: /usr/local/forensics//sans/tortilla/images/dev_hda7.img Inode: 32131
```

The `sorter` tool could also have been run from the command line. The `man` page for `sorter` explains the various arguments required to run it. Of particular note, however, is the optional "-e" argument, which restricts the checks to extension mismatches only. I identified my camouflaged files with the following command:

```
[root@tortilla sleuthkit]# ./bin/sorter -h -m '/mnt/forensics/home/' -d  \
'/usr/local/forensics/sans/tortilla/output/sorter-dev_hda7.img/' -f  \
linux-ext2 -e '/usr/local/forensics/sans/tortilla/images/dev_hda7.img'
```

Deleting Files

Of course one of the most basic methods of hiding data is simply to delete the file containing the data. In the ext2 file system the data concerned does not immediately disappear from the hard disk. Instead, the file system merely marks the relevant area on the hard disk as being available for use. Depending on several factors such as the amount

of free space on the disk and the level of disk activity, it may take a significant amount of time before that part of the disk is reclaimed by another file, thus destroying the original data. Taking these factors into account, there exists the possibility of recovering the contents of a deleted file and therefore finding the hidden data. Note that file deletion is handled differently under the newer ext3 file system; the block pointers in the file's inode are cleared, making recovery significantly more difficult ("Linux ext3 FAQ"). The recovery techniques outlined below are specifically for the ext2 file system.

Hiding Data by Deleting Files

People wishing to *temporarily* hide large amounts of data on a system would rarely choose the method of file deletion due to the potential difficulties in recovering all of the data intact. However, it is a method often used by people who wish to *permanently* hide data. Oftentimes users of UNIX-like systems in the workplace are told by Computer Support staff that files erroneously deleted cannot be recovered; they can only be retrieved from backups. These users may therefore form the impression that on Linux and UNIX systems where backups are not routinely performed (such as systems at home), they can permanently remove information from their hard disk by simple file deletion. The fact is that computer data are surprisingly resilient, and there is a good chance that at least some of the deleted data can be recovered (Crane).

To demonstrate hiding and finding data by file deletion I created a directory called /home/sans/del. In this directory I placed two files, one of 1250 bytes and one of 20000 bytes. The files were given names to indicate how many blocks (of 1024 bytes) they should take up on the disk:

```
[sans@tortilla del]$ ls -l two_blocks.txt
-rw-r--r--    1 sans     sans         1250 May 21 16:25 two_blocks.txt

[sans@tortilla del]$ ls -l twenty_blocks.txt
-rw-r--r--    1 sans     sans        20000 May 21 16:25 twenty_blocks.txt
```

As the idea was to delete these files and then attempt to recover them, I ran the `md5sum` tool on the files to generate a 128-bit message digest for later comparison. I then deleted the files with the `rm` command:

```
[sans@tortilla del]$ md5sum two_blocks.txt
e03ebe17c915edd1ee7bbcda09b2baeb  two_blocks.txt

[sans@tortilla del]$ md5sum twenty_blocks.txt
69405efd08f20c77b6842c0cb6999e8f  twenty_blocks.txt

[sans@tortilla del]$ rm two_blocks.txt
[sans@tortilla del]$ rm twenty_blocks.txt
```

Finding Data in Deleted Files

Using Linux Utilities

The Red Hat Linux distribution includes a file system debugging tool for the ext2 file system called `debugfs` that can be used to recover deleted files consisting of twelve or

fewer disk blocks. I decided to use this tool to see if I could recover the deleted file named two_blocks.txt. Firstly I unmounted the /dev/hda7 partition (/home) and created a forensic image to work with, as outlined in the appendix "Creating a Forensic Image". Although the `debugfs` tool does not necessarily require an image (it can work on the partition whilst it remains mounted), creating an image as soon as possible after deleting a file will increase the chances of being able to recover the file intact. Upon running `debugfs`, I requested to see a list of the deleted inodes by entering `lsdel` at the prompt:

```
[root@tortilla root]# debugfs /usr/local/forensics/dev_hda7.img
debugfs 1.27 (8-Mar-2002)
debugfs:  lsdel
 Inode  Owner  Mode    Size    Blocks   Time deleted
 36150    520 100644   1250    2/   2 Wed May 21 17:00:13 2003
 36149    520 100644  20000   21/  21 Wed May 21 17:00:17 2003
 36146    520 100644   7500    8/   8 Wed May 21 18:57:31 2003
 36147    520 100644   3750    4/   4 Wed May 21 18:57:37 2003
 36148    520 100644  10000   10/  10 Wed May 21 18:57:39 2003
 16074    520 100600    959    1/   1 Wed May 21 19:11:53 2003
 38156    520 100600  12288   12/  12 Wed May 21 19:11:53 2003
 38155    520 100664    113    1/   1 Wed May 21 19:25:41 2003
 16075    520 100600     53    1/   1 Wed May 21 19:28:17 2003
9 deleted inodes found.
```

Fortunately I knew from the file sizes that the first inode listed (36150) referred to the file two_blocks.txt that I deleted earlier. I then ran the `stat` command to display the contents of this inode:

```
debugfs:  stat <36150>
Inode: 36150   Type: regular    Mode:  0644   Flags: 0x0   Generation: 254443
User:   520   Group:   520   Size: 1250
File ACL: 0    Directory ACL: 0
Links: 0   Blockcount: 4
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x3ecc3d3d -- Wed May 21 17:00:13 2003
atime: 0x3ecc3a2d -- Wed May 21 16:47:09 2003
mtime: 0x3ecc352c -- Wed May 21 16:25:48 2003
dtime: 0x3ecc3d3d -- Wed May 21 17:00:13 2003
BLOCKS:
(0-1):147745-147746
TOTAL: 2
```

I then ran the `dump` command within `debugfs` to dump the contents of the inode to a file in my /tmp directory. This file should be identical to the deleted file two_blocks.txt. I then exited from `debugfs` and confirmed the successful file recovery with the `md5sum` command:

```
debugfs:  dump <36150> /tmp/two_blocks.rec
debugfs:  quit

[root@tortilla root]# ls -l /tmp/two_blocks.rec
-rw-r--r--    1 root     root         1250 May 22 11:42 /tmp/two_blocks.rec

[root@tortilla root]# md5sum /tmp/two_blocks.rec
e03ebe17c915edd1ee7bbcda09b2baeb  /tmp/two_blocks.rec
```

Using The Sleuth Kit / Autopsy

As the `debugfs` tool is inappropriate to recover files of more than twelve blocks in size, I

used Autopsy to attempt to recover the deleted file twenty_blocks.txt. The brief instructions for accomplishing this are:

1. Create an image of the partition as set out in the appendix "Creating a Forensic Image".
2. Assuming that a case and a host name have already been set up during the earlier section on camouflaged files, choose "File Analysis" from the main menu and click on the "All Deleted Files" button to show a listing of deleted files with details such as file type, file name, MAC times (modification, access and inode change times), file size, and inode number. The inode number (appearing under the column "Meta") for the file twenty_blocks.txt is 36149.
3. Choose "Meta Data" from the Autopsy main menu and enter in the inode number obtained from the previous step. The output is interesting because it shows that the file actually takes up 21 blocks on the disk, one of them being an indirect block. In the ext2 file system, the inode requires the use of indirect blocks to store the block numbers of all but the first twelve blocks of a file (Crane). In my example the block numbered 147736 contains no data for the file, but instead contains the block numbers for the final eight data blocks of the file (147737 - 147744):

```
Pointed to by file:
inode not currently used
File Type:
ASCII C program text
MD5:
69405efd08f20c77b6842c0cb6999e8f
Details:
inode: 36149
Not Allocated
Group: 18
uid / gid: 520 / 520
mode: -rw-r--r--
size: 20000
num of links: 0

Inode Times:
Accessed: Thu May 22 02:47:17 2003
File Modified: Thu May 22 02:25:48 2003
Inode Modified: Thu May 22 03:00:17 2003
Deleted: Thu May 22 03:00:17 2003

Direct Blocks:
147724 147725 147726 147727 147728 147729 147730 147731
147732 147733 147734 147735 147737 147738 147739 147740
147741 147742 147743 147744

Indirect Blocks:
147736
```

4. In order to recover the contents of inode 36149, I then clicked on the "Export Contents" button and saved the file to /tmp/twenty_blocks.rec. Finally I confirmed that the recovered file was identical to the original by running the md5sum command and comparing it to the message digest obtained earlier:

```
[root@tortilla root]# ls -l /tmp/twenty_blocks.rec
-rw-------    1 root     root        20000 May 22 13:57 /tmp/twenty_blocks.rec

[root@tortilla root]# md5sum /tmp/twenty_blocks.rec
69405efd08f20c77b6842c0cb6999e8f  /tmp/twenty_blocks.rec
```

Although I was successful in recovering both of the deleted files, this is in no way guaranteed by the ext2 file system. Some of the factors that lower the chances of file recovery are large file sizes, high levels of disk fragmentation and high use of the system by multiple users (Crane).

Unlinking Open Files

This is a strategy often used by sniffing programs to log information such as passwords to a file whilst minimising the chances of that file being discovered by a system administrator. It involves a process first opening a file for logging purposes, then calling the `unlink` function on this file, and then writing information to the file. The ext2 file system (along with all major UNIX-like file systems) will keep a lock on the resources used by the file descriptor until either the process exits or the file descriptor is closed (Green p.37). This means that although the file name will not appear on any listings done on the system, its data is protected from being overwritten. The person running the process can therefore return at a later time to collect the logged information, content that their efforts have caused minimal changes to the file system.

Hiding Data via Unlinking an Open File

I created a directory of /home/sans/unlink with which to experiment with this method. In this directory I placed a small text file that simulates the user and password information that may be collected by a sniffing program:

```
[sans@tortilla unlink]$ cat junk1.txt
user: tim       passwd: tim123
user: dawn      passwd: dawn123
user: david     passwd: david123
user: gareth    passwd: gareth123
```

I also placed into this directory a small C program that demonstrates the unlinking of an open file. This program was copied from Bach (p.137) with only one alteration of note (mentioned within the source code):

```
[sans@tortilla unlink]$ cat testdel.c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

main(int argc, char **argv)
{
  int fd;
  char buffer[1024];
  struct stat statbuf;

  if (argc != 2) {
    /* Sorry, I need a filename to delete */
    printf("Error: No filename provided.\n");
    exit();
  }

  fd = open(argv[1], O_RDONLY);

  if (fd == -1) {
```

```
    /* Can't open file */
    printf("Error: Cannot open file %s\n", argv[1]);
    exit();
  }

  if (unlink(argv[1]) == -1) {
    /* Can't unlink the file */
    printf("Error: Cannot unlink file %s\n", argv[1]);
    exit();
  }
  else {
    printf("File has been unlinked.\n");
  }

  /* Check the file by its name */
  if (stat(argv[1], &statbuf) == -1) {
    printf("stat %s fails as it should.\n", argv[1]);
  }
  else {
    printf("stat %s succeeded!\n", argv[1]);
  }

  /* Check the file by its file descriptor */
  if (fstat(fd, &statbuf) == -1) {
    printf("fstat %s fails!\n", argv[1]);
  }
  else {
    printf("fstat %s succeeds as it should.\n", argv[1]);
  }

  /* Added by G Robertson for demo purposes. */
  printf("Pause for 300 seconds, then output file...\n\n");
  sleep(300);

  /* Read open but deleted file */
  while (read(fd, buffer, sizeof(buffer)) > 0) {
    printf("%1024s", buffer);
  }

}
```

Although this program does not write data to a file in the manner explained earlier, it is sufficient to demonstrate the concept of unlinking an open file. The program expects the file name of an existing file to be provided as its only argument. The file is opened with the code "fd = open(argv[1], O_RDONLY)", and is later unlinked with the code "unlink(argv[1]". I added a couple of lines of code to the original program so that it pauses for a few minutes during execution. This provides me with some time to try and detect the process (before it completes) using operating system tools. I compiled the program with the following command:

```
[sans@tortilla unlink]$ gcc -o testdel testdel.c
```

A file listing of the directory after compilation shows the source and executable files for the program, along with the test file:

```
[sans@tortilla unlink]$ ls -l
total 17
-rw-rw-r--    1 sans     sans          113 May 21 19:01 junk1.txt
-rwxrwxr-x    1 sans     sans        12880 May 21 19:03 testdel
-rw-r--r--    1 sans     sans         1250 May 21 18:59 testdel.c
```

I ran the program with the command that follows, and quickly moved on to the next section to try and detect the file that had been unlinked:

```
[sans@tortilla unlink]$ ./testdel junk1.txt
File has been unlinked.
stat junk1.txt fails as it should.
fstat junk1.txt succeeds as it should.
Pause for 300 seconds, then output file...
```

```
                              user: tim        passwd: tim123
user: dawn      passwd: dawn123
user: david     passwd: david123
user: gareth    passwd: gareth123
```

Finding Data Hidden via Unlinking an Open File

Using The Sleuth Kit / Autopsy

I performed this search during the five minutes that the `testdel` program was paused. The object of the search was the contents of the file junk1.txt. This file no longer appeared on the directory listing, as it had been unlinked early on in the program:

```
[root@tortilla sleuthkit]# cd /usr/local/security/sleuthkit

[root@tortilla sleuthkit]# ls -l /home/sans/unlink/
total 16
-rwxrwxr-x    1 sans     sans        12880 May 21 19:03 testdel
-rw-r--r--    1 sans     sans         1250 May 21 18:59 testdel.c
```

I used the Sleuth Kit's `ils` tool to assist me in the search for the hidden data. The `ils` tool by default lists the inodes of removed files, and by using the "-o" option the output is limited to the files that are still open or executing. I entered the following command to search on the /dev/hda7 partition (/home), specifying linux-ext2 as the relevant file system:

```
[root@tortilla sleuthkit]# ./bin/ils -of linux-ext2 /dev/hda7
class|host|device|start_time
ils|tortilla|/dev/hda7|1053580871
st_ino|st_alloc|st_uid|st_gid|st_mtime|st_atime|st_ctime|st_dtime|st_mode|st_nli
nk|st_size|st_block0|st_block1
1|a|0|0|1056989311|1056989311|1056989311|0|0|0|0|0|0
3|a|0|0|0|0|0|0|0|0|0|0
4|a|0|0|0|0|0|0|0|0|0|0
5|a|0|0|0|0|0|0|0|0|0|0
6|a|0|0|0|0|0|0|0|0|0|0
7|a|0|0|0|0|0|0|0|0|0|0
8|a|0|0|0|0|0|0|0|0|0|0
9|a|0|0|0|0|0|0|0|0|0|0
10|a|0|0|0|0|0|0|0|0|0|0
38155|a|520|520|1053579712|1053579724|1053580841|0|100664|0|113|155922|0
```

The output from `ils` is given in many columns, the meaning of which is explained in the `man` page. Of main interest in the search for deleted files, however, are the first, third and fourth columns. These show the file's inode number, user ID and group ID respectively. As for the rows of output, only the final row refers to a non-root user, so I ignored the

remainder. Therefore the inode number of 38155 should contain the data for the deleted file.

The Sleuth Kit's `icat` tool is ideal at this moment, as it provides for copying files by inode number. I used the following command to copy the contents of inode 38155 to a file in my /tmp directory:

```
[root@tortilla sleuthkit]# ./bin/icat -f linux-ext2 /dev/hda7 38155 >
/tmp/junk1.rec
```

As the output from the temporary file shows, the `ils` and `icat` tools have successfully found the data being hidden via the method of unlinking an open file:

```
[root@tortilla sleuthkit]# cat /tmp/junk1.rec
user: tim       passwd: tim123
user: dawn      passwd: dawn123
user: david     passwd: david123
user: gareth    passwd: gareth123
```

Slack Space

In order to understand the use of slack space to hide data, one first has to understand a little about how computer hard disks are divided up and how operating systems read and write data to them. During a low-level format, hard disks are divided up into tracks and sectors so that operating systems can later use these divisions to store and find data. Tracks are concentric circles on a disk surface, whilst sectors are angular portions of the disk, like pieces of a pie. Most hard disks use a sector size of 512 bytes (Kuepper). After a high-level format has been performed, the filesystem will perform read and write operations on the disk in groupings of sectors called blocks (or clusters for the Windows operating system). On the ext2 file system a block will invariably be a grouping of either two, four or eight sectors - in other words 1024, 2048 or 4096 bytes (Chuvakin). The /dev/hda7 (/home) partition on my disk has a block size of 1024 bytes. Of course file sizes are only dependent on the amount of data being stored, and therefore are rarely exact multiples of block sizes. However, the effective space taken up by the file when written to the hard disk will always be a multiple of the block size. For example, a file of only ten bytes stored on a Linux partition that uses 1024 byte blocks will still take up 1024 bytes on the hard disk (1014 bytes will go unused). If exactly 1500 bytes of data is later appended to that file, giving it a file size of 1510 bytes, it will then take up 2048 bytes (ie two blocks) on the hard disk. The area on the hard disk between the end-of-file indicator and the final block boundary is referred to as the slack space for the file. As the slack space is not addressable by the file system, it can be used to hide data, although the amount of hidden data is limited to the file system's block size.

Hiding Data in Slack Space

There is a tool called `bmap` that can be used to access file slack space on Linux systems (superuser privileges are required). It was written by Daniel Ridge for Scyld Computing Corporation. I downloaded the RPM file bmap-1.0.20-1.i386.rpm from the following site:

ftp://ftp.scyld.com/pub/forensic_computing/bmap/RPMS/i386/

Before installing bmap yourself, please note that it may damage your hard disk. The README file states:

WARNING: This may spank your hard drive

I installed bmap with the following command:

```
[root@tortilla /]# rpm -iv bmap-1.0.20-1.i386.rpm
```

I created a directory of /home/sans/slack in which to experiment with bmap. I created two text files for the purpose of using them to hide data. The exact sizes of the files are shown by the following command:

```
[sans@tortilla slack]$ ls -l file?.txt
-rw-r--r--    1 sans     sans           10 May 21 15:54 file1.txt
-rw-r--r--    1 sans     sans         1503 May 21 15:54 file2.txt
```

As the /dev/hda7 (/home) partition on my disk has a block size of 1024 bytes, this is also the maximum size of file slack space I can use to hide data. For the purposes of this demonstration I used far less than that. The following commands show how I (after logging in as the superuser) hid the same text string, “cybercriminal”, in both files using `bmap` with the `putslack` option:

```
[root@tortilla slack]# echo "cybercriminal" | bmap --putslack file1.txt
stuffing block 139522
file size was: 10
slack size: 1014
block size: 1024

[root@tortilla slack]# echo "cybercriminal" | bmap --putslack file2.txt
stuffing block 139524
file size was: 1503
slack size: 545
block size: 1024
```

The following commands demonstrate that the data has been hidden in the files' slack space. Firstly, I use `bmap` with the `slack` option to retrieve the contents of slack space for the two files. Secondly, I show that neither of the file sizes has changed; Linux is not aware of the hidden data. Thirdly, I confirm using `grep` and the `cat` command on the smaller file that the hidden data is not accessible via normal operating system utilities:

```
[root@tortilla slack]# bmap --slack file1.txt
getting from block 139522
file size was: 10
slack size: 1014
block size: 1024
cybercriminal

[root@tortilla slack]# bmap --slack file2.txt
getting from block 139524
file size was: 1503
slack size: 545
block size: 1024
cybercriminal

[root@tortilla slack]# ls -l file?.txt
-rw-r--r--    1 sans     sans           10 May 21 15:54 file1.txt
-rw-r--r--    1 sans     sans         1503 May 21 15:54 file2.txt

[root@tortilla slack]# grep cybercriminal *

[root@tortilla slack]# cat file1.txt
123456789
```

It should also be noted at this point that not even message digests produced with the `md5sum` utility will indicate the presence of data in the file slack space, as this utility operates only on the file contents.

The technique of hiding data in file slack space is seldom used because of two main reasons. Firstly, it does not allow for hiding large amounts of data. Even file systems with a block size of 4096 bytes are restricted to only 4KB of hidden data per file. Secondly, the technique is only useful for files that are very stable, as modifications to the file can make the hidden data inaccessible even to the tool responsible for its placement. As a demonstration, I used a text editor to delete the first character in the smaller of the two files containing hidden data. Subsequently I found that attempting to retrieve the hidden data with `bmap` (using the `slack` option) resulted in my screen filling with spurious characters. The tiniest modification to the file had caused my method of accessing its slack space to fail completely, although other slack space tools may be able to handle file modifications.

The limitations of this technique mean that it is very unlikely to be used by insiders as a method of hiding data. After all, legitimate users of the system presumably have the permissions to write and modify files, so they have plenty of opportunity to store small amounts of data. File slack space is more likely to be used by people who don't have the permission to write files on the system, such as hackers. For example, hackers could use the technique to store small Perl scripts or lists of cracked passwords.

Finding Data Hidden in Slack Space

Using Linux Utilities

The search for data hidden in file slack space is best done via a forensic investigation of the partition in question. To achieve this on my system, I first unmounted the /home partition and then used the `dd` utility to create an image of the partition to work with. The instructions for doing this are contained in the appendix "Creating a Forensic Image". I then used the `strings` utility to search the entire image. By default this command will output any strings of four or more printable characters that it finds within the given file (in this case the file is an image of the entire partition):

```
[root@tortilla /]# cd /usr/local/forensics

[root@tortilla forensics]# strings dev_hda7.img | grep "cybercriminal"
cybercriminal
cybercriminal
```

Although the search has been successful, it requires follow-up investigation on any matches because the output provided no information about where on the partition the strings were located. The lack of this information makes it difficult for investigators to see the strings in context, or even to establish that they were being stored in file slack space.

Using The Sleuth Kit / Autopsy

Autopsy provides the ability to search for key words in both allocated disk space and unallocated disk space. As file slack space falls into the category of allocated disk space (the relevant blocks have been allocated), this demonstration focuses on that portion of my /dev/hda7 partition. The brief instructions for accomplishing this are:

1. Create an image of the partition as set out in the appendix "Creating a Forensic Image".
2. Assuming that a case and a host name have already been set up in previous sections, choose "Keyword Search" from the main menu. If this is the first time a search for a particular string is being conducted on the allocated portion of the forensic image, the "Extract Strings" button should be clicked. Autopsy will then go through the image and create an indexed file of all strings discovered. This file can be quite large - my 300 MB image resulted in a strings file of almost 40 MB – however, it enables upcoming searches to scan only the strings file rather than the entire image.
3. Enter the string to search for, in my case "cybercriminal", and click on the search button. The Autopsy output from my search identified two occurrences of this string. It also provided me with the block numbers on the partition where the strings were found, the offset within these blocks and links to view the blocks in either ASCII or hexadecimal. As shown in the following output, the block numbers (also referred to by Autopsy as Fragments or Data Units), were 139522 and 139524:

```
2 potential occurrences of cybercriminal were found
139522 (Hex - Ascii)
- offset 10 bytes
139524 (Hex - Ascii)
- offset 479 bytes
```

4. The Autopsy output is a significant improvement on the `strings` command I ran previously (admittedly the `strings` command was run without options). By providing the opportunity to view the block where the string was located, the string can be seen in context, which may provide vital clues in an investigation. I clicked on the "Ascii" link to see the contents of block 139522, although I could also have seen the block's contents by choosing "Data Unit" on the Autopsy main menu:

```
ASCII Contents of Fragment 139522 (1024 bytes) in images/dev_hda7.img

123456789
cybercriminal
```

5. Further information was provided when I clicked on the link to “ASCII report” - in particular, the name of the file that points to block 139522. An analysis of the file contents and the block contents may then lead an investigator to deduce that file slack space has been used to hide data:

```
Autopsy ascii Fragment Report (ver 1.71)

------------------------------------------------------------
Fragment: 139522
Length: 1024 bytes
Pointed to by Inode: 34138
Pointed to by files:
  /mnt/forensics/home/sans/slack/file1.txt
MD5 of raw Fragment: 377cef21391dd33d948afcdcc364b572
MD5 of ascii output: e64c7c4c4895b3bdcc672ca99208ff46
Image: /usr/local/forensics//sans/tortilla/images/dev_hda7.img
Image Type: linux-ext2
Date Generated: Fri May 23 13:42:34 2003
Investigator: gjr
------------------------------------------------------------
```

```
123456789
cybercriminal
```

Appendix

Creating a Forensic Image

Forensic examination of a suspect system should never be performed on the system itself. Instead, it should be performed on a trusted system, using images taken from the suspect disk. These images, which are exact bit-by-bit copies of the original partitions, effectively allow the system to be examined without fear of contaminating the original system state. Some of the techniques for finding hidden data that are discussed in this paper deal with a forensic image of the /dev/hda7 partition on my system, mounted on the /home directory. Rather than duplicating the instructions for creating and mounting a forensic image in various places throughout the paper, I have outlined the necessary steps in this section, and the examples in the paper refer to them as required.

The /dev/hda7 (/home) partition on my system is just over 300 MB in size, as shown by the following `df` command:

```
[root@tortilla root]# cd /

[root@tortilla /]# df -H
Filesystem             Size   Used  Avail Use% Mounted on
/dev/hda8              1.0GB   80MB  896MB   9% /
/dev/hda1               88MB  5.4MB   77MB   7% /boot
/dev/hda10              64MB   14kB   60MB   1% /hack
/dev/hda7              303MB  518kB  286MB   1% /home
none                    97MB      0   97MB   0% /dev/shm
/dev/hda6              406MB  8.9MB  376MB   3% /tmp
/dev/hda2              5.0GB  1.8GB  2.9GB  38% /usr
/dev/hda3              2.3GB   58MB  2.1GB   3% /usr/local
/dev/hda5              406MB   35MB  350MB   9% /var
```

The output from the `df` command shows that I have enough room on any one of a number of other partitions to store an image of 300 MB for investigation. Although this violates one of the main rules for forensic investigation, it is okay for demonstration purposes. I now unmount the /dev/hda7 device. This is necessary to stop any further file system interaction with the device, therefore preserving the data:

```
[root@tortilla /]# umount /dev/hda7
```

I now use the `dd` command to create a forensic image of my /dev/hda7 (/home) partition. The image that results, /usr/local/forensics/dev_hda7.img, is an exact bit-by-bit copy of the partition. At various stages throughout this paper I create such an image for analysis with forensic tools:

```
[root@tortilla /]# dd if=/dev/hda7 of=/usr/local/forensics/dev_hda7.img
610406+0 records in
610406+0 records out
```

Note that upon completion of a section requiring the use of a forensic image, the subsequent section will require remounting of the partition (to effect the data hiding) and recreation of the image (to reflect the changes).

References

Bach, Maurice J. "The Design of the UNIX Operating System."
Prentice Hall. Englewood Cliffs, NJ. 1986.

Cheng, Derek. "Freeware Forensics Tools for UNIX." November 1 2001.
URL: http://online.securityfocus.com/infocus/1503 (30 April 2003)

Chuvakin, Anton. "Linux Data Hiding and Recovery." 10 March 2002.
URL: http://www.linuxsecurity.com/feature_stories/data-hiding-forensics.html
(30 April 2003)

Crane, Aaron. "Linux Ext2fs Undeletion mini-HOWTO." Version 1.3. 2 February 1999.
URL: http://www.tldp.org/HOWTO/mini/Ext2fs-Undeletion.html (30 April 2003)

Di Pietro, Roberto and Mancini, Luigi V. "A Methodology for Computer Forensics Analysis."
Proceedings of the 2002 IEEE Workshop on Information Assurance, United States Military Academy, West Point, New York. June 2002.

Dittrich, Dave. "Basic Steps in Forensic Analysis of UNIX Systems."
URL: http://staff.washington.edu/dittrich/misc/forensics (30 April 2003)

Farmer, Dan. "Bring Out Your Dead."
Dr Dobb's Journal. January 2001.
URL: http://www.ddj.com/documents/s=871/ddj0101h/0101h.htm (30 April 2003)

Green, John. "Basic Forensic Principles Illustrated with UNIX - Hands On."
Course Notes from SANS Conference, Sydney 2003, "Track 8 - System Forensics, Investigation and Response". SANS Institute. 2003.

Kruse, Warren G. II. and Heiser, Jay G. "Computer Forensics: Incident Response Essentials." Addison-Wesley. Boston, MA. 2001.

Kuepper, Brian. "What You Don't See On Your Hard Drive." 4 April 2002.
URL: http://www.sans.org/rr/paper.php?id=653 (10 May 2003)

"Linux ext3 FAQ." Authors unknown. 9 April 2003.
URL: http://batleth.sapienti-sat.org/projects/FAQs/ext3-faq.html (15 May 2003)

Shinder, Debra L. "Scene of the Cybercrime."
Syngress Books. Rockland, MA. 2002.

Sorenson, Holt. "Incident Response Tools for Unix, Part One: System Tools."
27 March 2003.
URL: http://www.securityfocus.com/infocus/1679 (30 April 2003)

Venema, Wietse. "File Recovery Techniques."
Dr Dobb's Journal. December 2000.
URL: http://www.ddj.com/documents/s=878/ddj0012h/0012h.htm (30 April 2003)