DRAFT

Computer Forensics Procedures and Methods

J. Philip Craiger, Ph.D., CISSP
Assistant Director for Digital Evidence
National Center for Forensic Science &
Department of Engineering Technology
University of Central Florida[1]

Email: philip@craiger.net

To appear in H. Bigdoli (Ed.), *Handbook of Information Security*. John Wiley & Sons.

Keywords

Digital forensics, computer forensics, network forensics, cyberforensics, digital evidence, computer evidence, computer crime, incident response, Linux forensics, Windows forensics, computer forensic tools, computer forensics procedures, disk forensics, media forensics, intrusion forensics, intrusion detection systems, Knoppix.

Abstract

Computer forensics involves the preservation, identification, extraction and documentation of digital evidence in the form of magnetically, optically, or electronically stored media. It is a relatively new science that is becoming increasingly important as criminals aggressively expand the use of technology in their enterprise of illegal activities. This chapter is a *technical* introduction and overview to some of the fundamental methods and procedures of computer forensics. The topics covered parallel the order in which computer forensic procedures are typically conducted, beginning with process of creating a bit-stream image of the evidence and subsequent verification of the evidence using one-way hash functions. Two forms of forensic analysis are covered, including logical and physical analysis procedures. Analytic procedures we demonstrate include hash and signature analysis; keyword and email searches; recovery and analysis of cookies, print spool and application residual files; slack and unallocated space analysis; manual recovery of deleted files; behavioral timelines creation; and collecting evidence from running systems. We close the chapter by describing several commercial tools.

1. Introduction
    a. Computer Forensic Tools
    b. The Forensic Server
2. Sound Computer Forensic Practice
3. Arriving at the Scene: Initial Response
    a. Creating a Forensic Image
    b. Verifying Image Integrity
    c. Imaging Over a Network
    d. Sterilizing Forensic Media
4. Analysis of a Forensic Image
    a. Drive Geometry
    b. Mounting the Image
    c. Reducing our Search Space
        i. Hash Analysis
        ii. Signature Analysis
    d. Searching A Forensic Image
        i. Keyword Searches
        ii. Finding Files by Type
        iii. Email Searches
        iv. Swap file
        v. Web-based Email
        vi. The Windows Swap File
    e. I know what you did with your computer last summer…
        i. Cookies
        ii. Deleted Files and the INFO2 File
        iii. Application Residual Files
        iv. UNICODE
        v. Print Spool Files
    f. Physical Analysis
        i. What Happens when a File is Deleted
        ii. Unallocated Space Revisited
        iii. Slack Space
        iv. Recovering Deleted Files
        v. Dealing with Formatted Drives
    g. Behavioral Timelines: What Happened and When?
5. Collecting Evidence from Live Systems
    a. Volatile Evidence
    b. Log Files as Digital Evidence
    c. Reducing the Potential for Evidence Contamination
6. Commercial Tools
7. Conclusion
8. Glossary
9. References
10. Further Reading

Introduction

Computer forensics involves the preservation, identification, extraction and documentation of computer evidence stored in the form of magnetically, optically, or electronically stored media. It is a relatively new science that is becoming increasingly important as criminals aggressively expand the use of technology in their enterprise of illegal activities. Computer forensic techniques are not as advanced as those of the more mature and mainstream forensics techniques used by law enforcement, such as blood typing, ballistics, fingerprinting, and DNA testing. Its immaturity is partly attributable to fast-paced changes in computer technology, and the fact that it is a multidisciplinary subject, involving complicated associations between the legal system, law enforcement, business management, and information technology.

This chapter is a *technical introduction and overview* to fundamental methods and procedures of computer forensics. To get the most out of this chapter we have assumed readers will have technical skills with computers running a variety of operating systems.

The Handbook of Information Security, in particular volume II, has several chapters related to numerous aspects of computer forensics, including the legal, law enforcement, and managerial aspects. These chapters include *Computer Forensics in Law Enforcement*, *Forensic Science and Computers*, and *Computer Security Reviews Using Computer Forensics Tools*, *Digital Evidence, Digital Courts, Law, and Evidence, Cybercrime and Cyberfraud, and Hack, Cracker,* and *Computer Criminals*. To fully understand the practice and implications of computer forensics we urge readers to carefully examine *each* of these chapters. And as you read this chapter be aware that computer forensics is a set of technical activities that occurs with a complex setting of interacting stake holders who often have conflicting goals. Before conducting a computer forensics investigation we advise the reader to seek advice from legal counsel to ensure that no local, state, or federal laws are broken. Nothing in this chapter is intended to be legal advice, and should not be construed as such.

In this chapter we illustrate both *offline* and *online* analyses. An offline analysis occurs when an investigator powers down the computer and removes it from the network. This allows the investigator to create an exact copy of the computers hard drive to ensure that the files remained unchanged, and to ensure all evidence, but condemning, as well as exculpatory, is collected. In contrast, there are occasions when it is impossible to power down a computer, requiring an online analysis. For instance, management may not permit the shutdown of a company's only e-commerce server. In this circumstance the investigator must gather as much evidence as possible while the system remains running and connected to a network. From a purely forensic standpoint, the preferred situation is to 'freeze' the computers state by powering down the system. However, in reality this is not always possible, and investigators should be proficient in methods for gathering evidence from a running computer system.

We begin this chapter by describing an offline analysis involving desktop computers running versions of Microsoft Windows™. Windows plays a prominent role because of its large worldwide market share, and the fact that the law enforcement agencies (Dartmouth, 2002), as well as the FBI's Computer Analysis and Response Team (Pollitt, 2002, personal communication) have indicated that the majority of investigations involve computers running some version of Windows. We conclude this chapter by discussing an online

analysis, such as a server running Linux or UNIX that cannot be shutdown, requiring the investigator to work on a running computer system.

Computer Forensics Tools

Investigators have a variety of forensic tools from which to choose. Some tools run exclusively under Windows, others under Linux/UNIX, and some on several operating systems. The focus of this chapter is on illustrating fundamental computer forensics *concepts*, and not a particular tool. An important reason for *not* using point-and-click tools for our demonstrations is that these tools don't illustrate the fundamental, technical details that form the core of computer forensic procedures. For example, when an investigator clicks on a GUI button labeled [Recovered Deleted File] from within a GUI-based tool she should be able to explain *what* the program is doing to recover the files. In real life of course, investigators are likely to use one of the many GUI-based tools that are available (some of these are described at the end of this chapter). Nevertheless, a fundamental understanding of these concepts is especially important for the credibility of the investigator should she be required to testify in a court of law as an expert or technical witness.

Accordingly, we use Linux-based tools for our demonstrations. Linux is an operating system kernel. The command line utilities we use to conduct our forensic procedures are part of the GNU utilities (www.gnu.org) that are included in every Linux distribution. A Linux *distribution* is a vendor compilation of the Linux kernel, GNU utilities, and hundreds of software programs, plus an installer. There are several dozen Linux distributions, although only a few are major commercial distributions, the largest of which are Redhat (www.redhat.com) and SuSE (www.suse.com). Readers are directed to www.distrowatch.com and www.linux.org to learn more about various Linux distributions. All demonstrations herein were tested with SuSE Professional 9.0/9.1, Fedora Core 1, and Redhat 9.

The Forensic Server

We assume that most computer crime investigations will involve at least one 'subject' computer, i.e., the source of the evidence we are seeking, as well as a 'forensic server' that contains our forensics toolkits. For our demonstration purposes we assume that the forensic server has a Linux distribution installed, or is dual-bootable Window/Linux, which is how all of our computers are configured. Although in theory the Linux distribution should be irrelevant, certain commercial versions such as Red Hat (www.redhat.com) and SuSE (www.suse.com) are often preferred because they offer support services and are fairly easy to install and update.

The activities performed in a forensic analysis may easily tax the average computer. Therefore it is important that a good deal of thought is put into the components that compose the forensic server to ensure that it is of sufficient quality and power so that imaging and analysis is not problematic. For instance, it is desirable to have as much physical RAM as one can afford, as well as a fast processor (all of which are relative and changing daily). The forensic server will enough drive space to hold the operating system, several forensic tools, as well as all of the forensic images collected from the subject's computer. Other considerations include the need to read and examine numerous portable disk formats, including old ZIP disks, superdrive disks, old floppy formats such as the 5.25

floppies, and so on. It is a good idea to have on hand many different types of disk readers that can be placed into the forensic computer should the need arise. Should the investigator come upon a digital evidence format that is not common, E-Bay (www.ebay.com) or similar sites may have such equipment available.

Sound Computer Forensic Practice

There are numerous circumstances that may require a computer forensic investigation but not necessarily law enforcement intervention. For instance, a company employee suspected of sending sexually explicit emails or running a personal business may be subject to an investigation because these activities violate corporate acceptable use policy and subject the employee to disciplinary action; however, they are not illegal and do not require law enforcement intervention. Nevertheless, it is good practice to work under the assumption that *any* investigation could end up in court. The reason is that there are countless stories of investigations that started off for one reason but escalated to a point requiring law enforcement intercession. For instance, an investigation instigated by allegations that an employee surfing pornography web sites on his lunch break may reveal evidence of a cache of child pornography on the employee's hard drive, a Federal crime under 18 USC 2251 and 2252. The situation must be reported to law enforcement and would likely end up in court.

The Federal rules of evidence (http://www.law.cornell.edu/rules/fre/overview.html) govern the introduction of evidence in both civil and criminal proceedings in Federal courts. These rules are strict regarding the handling of evidence. Evidence not collected in accordance with the Federal rules of evidence may be disallowed by a judge. Sound forensic practices decreases the potential for a defense attorney to question the integrity of evidence, and for the judge to disallow the introduction of evidence into a court proceeding.

Computer forensics procedures can be distilled into three major components:

1) Make a digital copy of the original evidence. Investigators make a copy of the evidence and work with the copy to reduce the possibility of inadvertently changing the original evidence.
2) Authenticate that the copy of the evidence. Investigators must verify the copy of the evidence is exactly the same as the original.
3) Analyze the digital copy. The specific procedures performed in an investigation are determined by the specific circumstances under which the investigation is occurring.

Our chapter will generally follow this outline. We begin by demonstrating two ways in which to make forensically-sound copies of digital evidence, followed by a demonstration of a simple and effective way of verifying the integrity of a digital copy. The remaining portions of this chapter are devoted to procedures for analyzing digital evidence.

Arriving at the Scene: The Initial Response

There are two important rules regarding the initial response to a computer crime scene. One of the most critical times at any crime scene is when the crime is first discovered. The first activity performed by law enforcement at a physical crime is to restrict access by surrounding the crime scene with yellow tape, something most of us have seen on TV

hundreds of times. It is just as important to *restrict access* to the computer at a computer crime scene, to decrease the likelihood of changing the evidence.

The second rule is to *document* the crime scene and all activities performed. Good documentation is crucial for several reasons. First, it allows the investigator to refresh her memory should she have to testify. Second, it allows the court to verify that correct forensic procedures were performed. Finally, it allows for the recreation of the activities that were performed in the initial response.

Special agent Mark Pollitt (retired), former Unit Chief of the Federal Bureau of Investigations Computer Analysis and Response Team (CART) says: "Computer forensics is all about process" (Pollitt, 2003, personal communication). The process should be repeatable and predictable, and stay within the confines of the law. We underscore the importance of following sound forensic practice during investigations, as there is a significant potential for any investigation to have legal implications for the investigator, her employer, or the subject of the investigation. Below we outline of the activities that should be performed as an initial response to a potential computer crime scene. The following is partly adapted from International Association of Computer Investigation Specialists http://www.iacis.com/forensic_examination_procedures.htm as well as *U.S. Secret Service's Best Practices Guide for Seizing Electronic Evidence* (http://www.secretservice.gov/electronic_evidence.shtml).

1. Immediately determine if a destructive program is running on the computer. If one is running, the investigator should pull the power plug from the *back* of the computer (not at the outlet). This will ensure no further evidence is lost. Place tape across all open disk drives so that no media is inadvertently placed in the disk drives. The system date and time should be collected from the BIOS setup. This time should be compared with a reliable time source (e.g., one synchronized with an atomic clock), and any discrepancies noted. This may be important if it is necessary to correlate events between two computers, or between the activities of a user and the times associated with particular files on the computer.
2. Document the computer and its surroundings. Video tape and photographs are good supplements to handwritten notes. Things to document include: The computer's make, model, serial number, attachments to the computer (e.g., external hard drives, speakers, cable modem, USB or network hubs, wireless network routers, and so on), the state of the machine, i.e., whether it was on or not, as well as the surrounding environment.
3. If the computer is running, take a photograph of the screen. Photographs demonstrate that the computer was running as well as visually documenting what was running at the time of the initial response.
4. Take photographs of the front, side, and back of the computer. A photograph of the back of the computer will allow an investigator to recreate the computer setup should the computer need to be seized and taken back to a lab for further investigation. If the computer is to be seized, label connectors (network, USB, firewire, etc.).
5. Physically open the computer and take photographs of the inside of the computer. These photographs will show the number of hard disks connected, as well as any peripherals, such as network and sound cards.
6. *Bag-and-tag* of all potential evidence. Bag-and-tag is a law enforcement term that refers to the process of placing crime scene evidence (e.g., hairs, fibers, guns, knives, and so on) into bags, and tagging them with relevant information including date and time collected, name of investigator, where collected, etc. All potential evidence such as floppy disks, CDs, DVDs, papers surrounding the computer, etc., should be subjected to a bag-and-tag.

7. Some situations require the confiscation of the source computer by law enforcement (Heverly & Wright, 2002). If the computer is to be transported to an offsite forensics ab, label each computer part and place in an appropriate container for transport.
8. Search for 'sticky notes' or any other written documentation near the computer (including under the keyboard, under the desk, in desk drawers, etc.). Users often write down passwords and leave them in convenient places near the computer. Passwords may be necessary if the user has used encryption to obfuscate file contents. Make sure to look at the waste basket as it may hold valuable information.
9. Take any computer manuals in case they are needed for reference back at the forensics lab.
10. If the original evidence is to be confiscated it should be stored in a secure place.

Creating a Forensic Image

The first step in acquiring digital evidence is to create an *exact physical copy* of the evidence. This copy is often called a *bit-stream image* (Kruse & Heiser, 2001), *forensic duplicate* (Mandia, Prosise, & Pepe, 2003), or *forensic image*. Creating a forensic image is important for several reasons. From a legal standpoint, courts look favorably upon forensic images because it demonstrates that *all* of the evidence was captured, condemning as well as exculpatory, following the spirit of the Federal Rules of Evidence. From an investigatory perspective, forensic images contain the contents of previously deleted files and other ambient data, information not available if only a logical copy of files is made.

Historically a running computer was of little concern to law enforcement as the standard operating procedure was to remove the power source from the computer, i.e., pull-the-plug, whether the computer was running or not. Pulling-the-plug follows general police investigative procedures to 'freeze' the crime scene. Pulling-the-plug is analogous to the yellow "Do Not Cross Police Line" tape as it freezes the computer crime scene. This practice is no longer a hard-and-fast rule as pulling-the-plug may lose valuable evidence, e.g., running network connections and the contents of RAM.

Options for freezing the computer include pulling the plug or gracefully shutting the computer down (i.e., that is through the mouse sequence: Start: Turn Off Computer: Turn Off). In personal experiments involving computers running versions of Windows 98 and Windows XP we observed that a graceful shutdown results in changes to several hundred files on the disk. This could have a significant effect upon a law enforcements ability to prosecute a case, especially trying to explain how the investigating officer managed to change several hundred files on the evidence media. Our advice is that the *circumstances must determine whether it is appropriate to pull-the-plug or perform a graceful shutdown*.

We must access the hard drive on the subject's computer to create our image. How do we access the source hard drive? One way is to physically remove it from the source computer and connect it to the investigator's forensics machine. For example, if the source drive is an ATA (IDE), it is relatively simple to remove it and reconnect it to an open IDE ribbon cable connection in the forensics computer, as demonstrated in Figure 1.
.

Figure 1. Direct connection to internal IDE controller

There are two caveats to this method. First, if we are imaging using a Windows-based application to image then we must use a write blocker (see Figure 2 below) to ensure that no data is written back to the subject's hard drive, as Window's will automatically mount the hard drive as read+write, which may change files on the hard drive. If we are using Linux then a write blocker is not required because we can manually mount the hard drive as 'read only.' Forensic imaging under Windows therefore requires the use of a special 'write-blocker,' a hardware mechanism that allows reading from, but not writing to, the hard drive (see Figure 2). Write blockers are available from several sources, including FireFly from www.digitalintel.com, and FastBloc from www.guidancesoftware.com.

Second, directly connecting the subject's hard drive to our forensic server's IDE chain requires that the jumpers on the source drive be appropriately set to slave or cable select. Changing jumper settings will not change any files on the hard drive.

Figure 2. Write-block with firewire connection

Figure 2 demonstrates the use of a write-block to connect a subject's hard drive. The hard drive is connected to the write-block, which is connected via a USB or firewire cable to the forensic server.

Once we have connected our hard drive we can begin the imaging process. In the following demonstration we will create a forensic image of a floppy; however, the process if very similar for all media, any differences are indicated below. Later we will demonstrate how to image a hard drive over a network connection.

From a terminal window we use the GNU utilities from a command prompt to make our forensic image. First we write protect the floppy, then place the disk in our floppy drive on our forensic server. The commands we executed have been marked in brackets for ease of reference.

```
[1] jpc@simba:~> script case.1034
    Script started on Tue 23 Mar 2004 03:25:36 PM CST
[2] jpc@simba:~> date
    Tue Mar 23 15:25:39 CST 2004
[3] jpc@simba:~> dd if=/dev/fd0 of=1034.dd bs=1024 conv = noerror,
notrunc,sync
    2880+0 records in
    2880+0 records out
[4] jpc@simba:~> md5sum /dev/fd0 1034.dd > evidence.md5
[5] jpc@simba:~> cat evidence.md5
    04c09fa404ac7611b20a1acc28e7546c  /dev/fd0
    04c09fa404ac7611b20a1acc28e7546c  1034.dd
[6] jpc@simba:~> date
    Tue Mar 23 15:34:13 CST 2004
[7] jpc@simba:~> exit
    Script done on Tue 23 Mar 2004 03:34:14 PM CST
```

A good investigator documents the crime scene as well as the procedures performed. We can supplement our handwritten notes with the `script` command. `script` [1] makes a

copy of everything printed on the screen (both input and output) and places it in a file, here descriptively called *case.1034*.

We then print the current date and time with `date` [2]. It is good practice to 'sandwich' your forensic activities between two `date` commands [2]& [6] to demonstrate when and how long the activities required.

Next we use the `dd` [3] command to create a forensic image of the floppy disk:

```
# dd if=/dev/fd0 of=1034.dd
```

`dd` takes as input a stream of bits, and outputs a stream of bits, making an exact physical duplicate of a file, drive, etc. In this example `dd` is reading from the device */dev/fd0*, a logical device associated with the floppy disk drive, and writing it to a file we have named *1034.dd*. In Linux physical devices are logically associated with a file residing in the */dev* directory. These associations are shown in Table 1. Note that the names of the logical devices may differ between Linux distributions, or between versions of Unix. The nomelcature below is fairly standard for most POSIX compliant Linux distributions, including Redhat, SuSE, Mandrake, Slackware, Debian, and Gentoo.

Logical Device	**Physical Device**
/dev/hda	1st IDE hard drive on primary controller
/dev/hda1	1st partition 1st hard drive on primary controller
/dev/hdb	2nd IDE hard drive on primary controller
/dev/hdc	1st IDE hard drive on secondary controller
/dev/hdd5	5th partition on 2nd hard drive on secondary controller
/dev/sda	1st SCSI device
/dev/sda1	1st partition on 1st SCSI device
/dev/cdrom	1st CDROM drive
/dev/fd0	1st floppy disk

Table 1. Mapping from logical to physical device

The argument `if=` specifies the source image, here the logical device associated with the floppy drive. The argument `of=` specifies the output file's name. The `bs=` argument specifies the block size to read and write, and is optional. The default block size is 512 bytes. The *conv* argument specifies other command line arguments to include. For imaging we include: a) `noerror`, continue after a read error; b) `notrunc`, do not truncate the output in case of an error, and c) `sync`, in case of a read error, pad input blocks with zeros.

After the `dd` operation is completed it prints to the screen the number of records (i.e., blocks) it read and wrote. Here it read 1440 records and wrote the same number. This is correct because we have specified a block size of 1024 bytes (1 kilobyte), and 1 kilobyte multiplied by the number of blocks (records = 1440) is 1.44MB, which is the size of a high-density floppy disk.

Verifying Image Integrity

Next we must verify that we made an exact bit-for-bit copy of the evidence, i.e., a forensic image of the floppy. We use *md5sum* [4] to verify the integrity of image:

```
#md5sum /dev/fd0 1034.dd > 1034.md5
```

md5sum calculates an MD5 cryptographic hash, also known as a *message digest*, or simply *hash*. The MD5 message digest (Rivest, 1992) is a one-way hash algorithm that takes as an input a file of arbitrary length and outputs a 128-bit hexadecimal formatted number that is unique to a file's contents. Two files with the same contents will always result in the same 128-bit hash value. The file's contents alone determine a hash value, not associated metadata (e.g., file name, date and times, size, etc.). This fact will be important when we later consider ways to identify illegal or inappropriate files.

We verify the integrity of our evidence by calculating the MD5 hash for the original floppy disk (*/dev/fd0*) and the forensic image (*1034.dd*). If the hashes are the same we can rest assured the copy we made is a bit-for-bit duplicate of the evidence. We save the contents of the command by redirecting it to a file that can be printed for safekeeping.

Any differences between the contents of the floppy disk and our forensic image are so indicated in the hash. To illustrate this phenomenon, we used a hex editor to add a single space into the boot sector of the *1034.dd* image, and then reran the MD5 hash on the image. Comparing the old and new hashes:

```
Old Hash: 04c09fa404ac7611b20a1acc28e7546c
New Hash: dbbbd457d0283103e7148075abb5b91e
```

Imaging over a Network

An alternative to removing the hard drive from the subject computer is to use a network connection. A 'network acquisition' requires both computers have network interface cards (NICS; i.e., Ethernet cards), a network crossover cable, and a bootable Linux CD. A network crossover cable allows the investigator to directly connect two computers without a hub or switch.

Several bootable Linux-based CD-ROMs are available. Most of these are based on the popular Knoppix CD (www.knoppix.com). Knoppix contains over a 1.7 gigabytes of software on a 700MB CD, possible through compression. The CD contains utilities that are useful for forensics imaging and previewing. Knoppix loads itself into a ram disk during boot, and will not access the hard drive of the subject's computer (as of Knoppix 3.3). Knoppix boots into a graphical user interface that allows the investigator read-only access to the hard drives on the subject computer, which is very useful for previewing the contents of the source drive.

As of 2004 there are several forensics bootable Linux CDs. These include Local Area Security (www.localareasecurity.com), Knoppix Secure Tools Distribution (http://www.knoppix-std.org/) and Penguin Sleuth Kit (http://www.linux-forensics.com/), and FIRE (http://fire.dmzs.com/), the former three of which are based upon Knoppix.

Although strictly speaking Knoppix is not 'forensics distribution,' it contains enough tools to make itself extremely useful in these circumstances. It is so popular that we would surmise that it will be around for a long, long time.

Sterilizing the Forensic Media

Before imaging the subject's hard drive it is good practice to 'sterilize' or 'wipe' the destination media on the forensic computer. A forensic wipe removes any vestiges of previous contents on the drive, ensuring that a defense attorney cannot claim that any evidence recovered from the subject's hard drive was from the previous contents on the disk, caused by the co-mingling of evidence.

A forensic wipe is different than formatting a hard drive. For instance, a quick formatting under Windows or DOS only deletes the bookkeeping portion of the file system (described later), including the root directory and the file allocation table. A quick format does not remove the actual files: these files will remain until overwritten by the operating system. A full format is the same as a quick format except that it also writes F6h over each of the sectors in the data area of the disk. It does not, however, overwrite areas of the hard drive that are unaddressable by the operating system (usually several clusters at the end of a drive).

A forensic wipe can be accomplished with the `dd` command:

```
# dd if=/dev/zero of=/dev/hdb1 bs=2048
```

The logical device `/dev/zero` is an infinite source of zeros. Because we have specified our output to be `/dev/hdb1` it will write a series of zeros to every single sector of the first partition on the second IDE hard drive. The `bs` argument specifies that `dd` should read in blocks of 2048 bytes, overriding the default of 512 bytes.

We can verify that the procedure was successful using the `grep` command:

```
# grep -v '0' /dev/hdb1
```

`grep` is a utility that searches for keywords within files. We are searching for the string '0' on the logical device */dev/hdb1*. The `-v` argument specifies somewhat of a reverse search, that is, display everything on the media that is *not* '0.' If `grep` finds anything that is not '0' it will print the results to the screen. (We have personally never seen `dd` fail in this task.).

Wiping the drive removes the file system, so we must perform a high-level format before we can copy the image to our destination hard drive. Linux allows us to formats a drive in several different file system formats, including Linux/UNIX-based such as EXT2 or EXT3, or Windows FAT16/32. Although Linux can read from and write to FAT32 formatted drives, in this instance it is not our best choice as FAT32 has 4GB (2^{32} bits) size limit for files, and most hard drives as 2004 are much larger. Unfortunately, NTFS write support under the current Linux kernel (either 2.4 or 2.6) is experimental, and is therefore not the best choice for critical tasks. For purposes of this chapter we will create an EXT2 file system on the destination drive. Filesystems based on the Large File System specifications can hold files up to 2^{63} bytes in size (www.suse.de/~aj/linux_lfs.html).

```
# mkfs.ext2 /dev/hdb1
```

The `mkfs` command is a wrapper for several programs that create files systems. Here we are creating a standard Linux EXT2 file system.

Once we have prepared our destination forensic drive we can begin our imaging process. Here are the steps to use a bootable Linux CD to acquire and preview a subject's hard drive over a network:

1. Check the boot order of your subject computer. It is *crucial that the boot order of the subject's computer is set to boot from the CD before the hard drive*. If the reverse is true then the hard drive will boot instead of the CD. This will result in changing the access and/or modification times on several hundred files on the hard drive. Boot order is managed from the computer's BIOS. To check the boot order we first remove the power cable from the back of the source computer, not at the wall outlet. We then open the subject's computer and *remove the power supply connector* from all of the hard drives. This is critical because it guarantees that the source hard drive cannot boot inadvertently. We next replace the computer's power cable and power the subjects computer on. During the boot process the computer should display an onscreen message indicating a key to press to access the BIOS setup (e.g., F1, F2, [Delete], etc.). Press the appropriate key to access the BIOS. If you miss the chance to access the BIOS the source hard drive cannot boot as its power source has been removed. Reboot and try again. Once in the BIOS setup change the boot order to CD first, followed by the hard drive. Place the Knoppix CD into the source computer. Replace the power supply connectors to all of the hard drives. DO NOT TURN ON THE SOURCE COMPUTER YET. This is also a good time to note the systems date and time setting.
2. We then connect the network crossover cable between the source and forensics computer's NICs, and then power on the subject's computer. It will boot into the Knoppix graphical user interface. Icons that represent the hard drive partitions found by Knoppix will appear on the desktop (see Figure 3). *These drives are not mounted.* You may preview the contents of the drives by clicking on the icons, which will open the drives read-only, displaying their contents in a file manager.

Figure 3. Knoppix Desktop

3. You must manually configure each computer with a network address because neither computer is connected to a DHCP server. Open up terminals on both computers. Figure 4 demonstrates how to set a network address under Linux. First we switch to root on both computers using the substitute user [`su`] command. Under Knoppix version 3.3 and later the `su` command does not require a root password. Set the network address using the `ifconfig` command: `# ifconfig <interface> <network address> <network mask>`. Do this for both computers (using different network addresses for each of course. Make sure that the network address was correctly set by issuing the command [`ifconfig`]. Ping the source computer to make sure you have a connection.

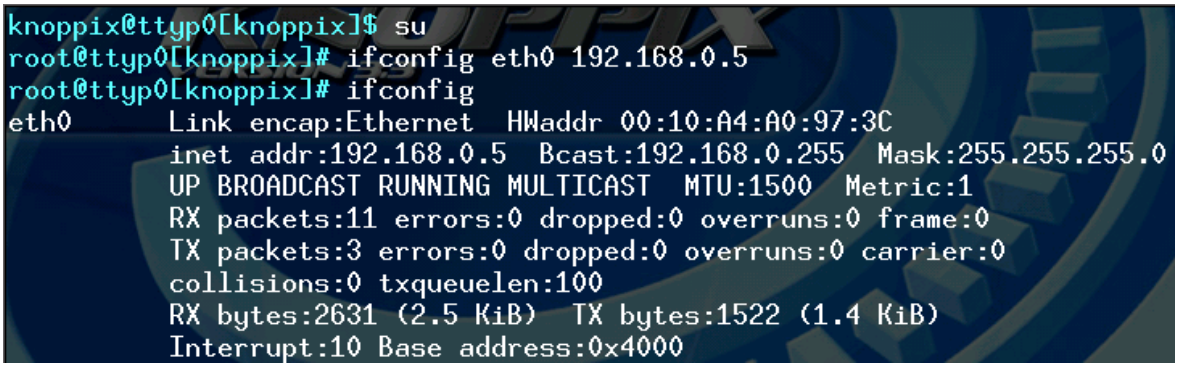

Figure 4. Viewing the network address of the subject's computer

4. Before we can continue we need more specific information on the hard drives from the subject's computer. From a terminal command line run the command `fdisk -l` (That's a lower case 'L'). As illustrated in Figure 5, this command displays the number of hard drives connected; the number of partitions on each drive, and the formatting of each partition. Figure 5 shows that our subject's computer has a single 14GB hard drive. (Note: Is our destination forensic drive large enough to hold this image?) We note that the drive has five partitions */dev/hda1* through */dev/hda5*. The fourth partition */dev/hda4* is an extended partition (as noted from the *W95 Ext'd* tag), and was created because drives can only have four primary partitions. This system appears to be a dual-boot Windows/Linux machine based on how the system is partitioned and our own experience. */dev/hda1* if formatted NTFS and likely contains a Window's operating system, although we can't guarantee this until we preview the image.

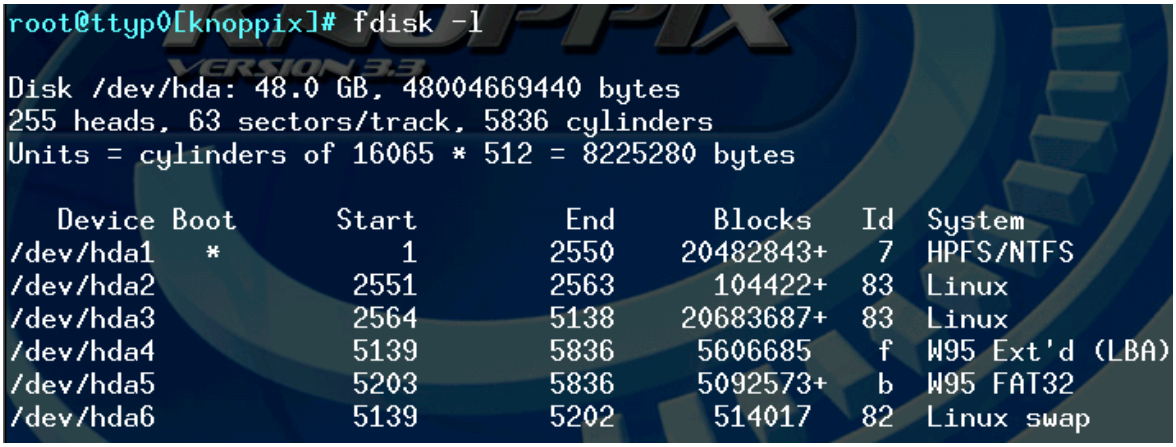

Figure 5. Determining a drive's geometry

5. We have sufficient information to begin an acquisition over the crossover cable. We use the *netcat* utility (www.atstake.com) to create the network connection between the two computers. Netcat is a small, free utility available for several operating systems. *Netcat* reads and writes bits over a network connection. The command to run on the forensics server is:

   ```
   # nc -l -p 8888 > evidence.dd
   ```

   This sets up the listen process on the forensics serve prior to sending the data from the subject's computer. From the command line arguments (below) `nc` is the netcat

executable (it may be called *netcat* under SuSE); The `-l` argument (a lowercase 'L') indicates *listen* for a connection; The argument `-p` specifies the port on which to listen, and we redirect the output to a file name of our choosing, here *evidence.dd*. If we don't redirect to a file then the output is directed to the standard output, the screen.

6. On the subject's computer we use the `dd` command to read the first partition:

```
#  dd if=/dev/hda1 | nc 192.168.0.2  8888 -w 3
```

We pipe the output of the `dd` command to *netcat*, which sends the bits over the network to the specified network address and port on our listening forensic computer. The argument `-w 3` indicates that *netcat* should wait 3 seconds before closing the connection upon finding no more data.

The time required to create a forensic image depends upon several factors, including the size of the source media, the speed of the connection (a directly connected IDE versus network acquisition), and the speed of the computers' hardware. Creating a forensic image of a floppy takes only a few minutes, whereas a 60GB hard drive will take several hours.

After we create the image we must verify its integrity. We can calculate the hash of the source hard drive by issuing the following command from the subject's computer:

```
# md5sum /dev/hda1 | nc 192.168.0.2 8888 -w 3
```

This command calculates the MD5 hash of the source hard drive and pipes the results over the network to our forensic server. We capture this information by setting up a listening process on the forensic computer as demonstrated in the first command below:

```
# nc -l -p 8888  >> evidence.md5
```

The command

```
# md5sum evidence.dd >> evidence.md5
```

calculates the MD5 hash of our forensic image and appends it to the previously created MD5 file. The ">>" command appends the output of the command to an existing file. WARNING: If we were to use a single ">" the file evidence.md5 would have been overwritten by the output of the command, rather than appended.

If our hashes match then we can assume success in our imaging process. We are now ready to begin an analysis of our image.

Analysis of a Forensic Image

Computer forensic procedures can be somewhat artificially divided between *logical* analysis and *physical analysis.* A logical analysis views the evidence from the perspective of the file system as in Figure 6.

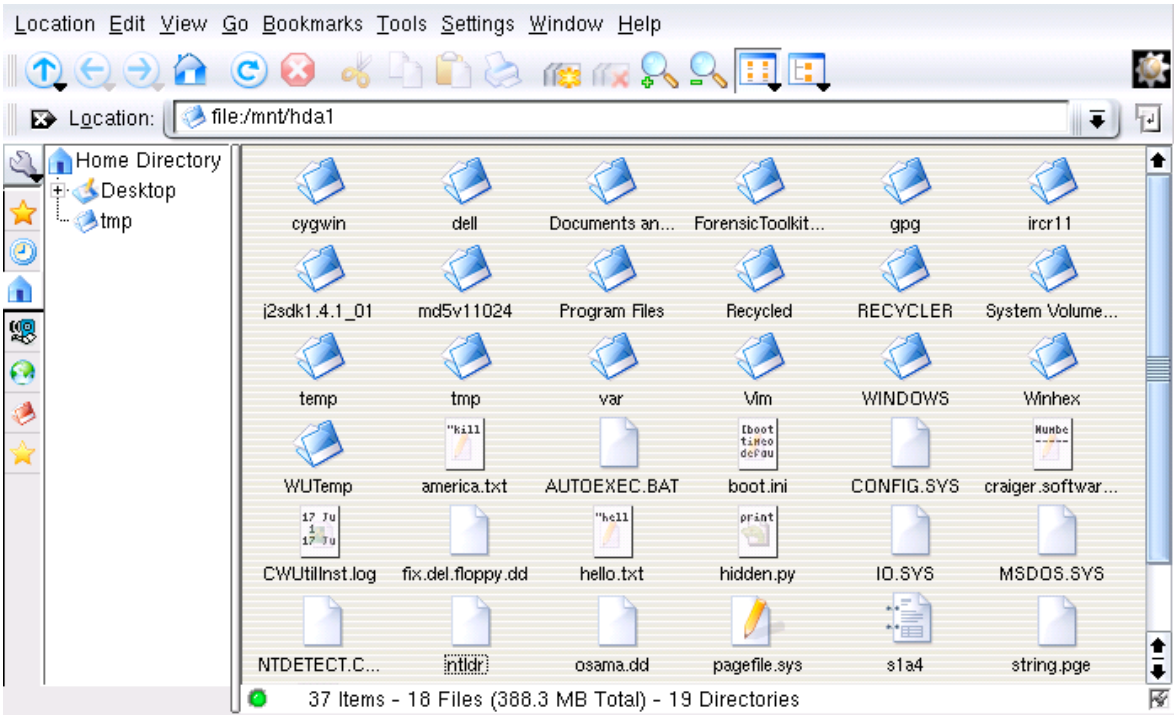

Figure 6. Forensic image preview under Knoppix

The investigator can use graphical tools, e.g., file managers, file viewers, etc., that are normally used on a computer. In contrast, a physical analysis views the forensic image from a purely physical viewpoint -- there is no file system to consider *per se*. Because physical analysis does not view the image from the perspective of a file system it requires the use of a hex editor and similar tools.

Logical and physical analyses are discussed in turn below. First we provide readers with a description of the drive which will be important in understanding aspect of both physical and logical analysis.

Drive Geometry

Figure 7 is a very simple, abstract model of a single magnetic disk. The single disk is called a *platter*. A hard drive will consist of multiple platters stacked on top of one another. Each platter is divided into a number of *tracks*. A *cylinder* is the column of two or more tracks on two or more platters (Nelson, et al., 2003). A *head* is a device that reads and writes data to a platter. Each track is divided into multiple *sectors*. Sectors are created via low-level formatting at the factory and are commonly 512 bytes in size.

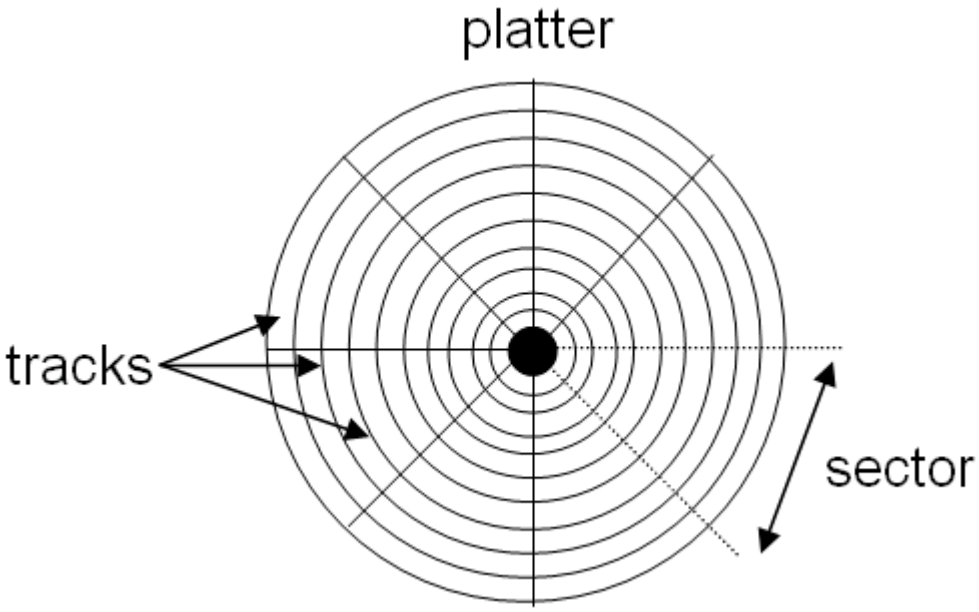

Figure 7. Drive geometry

High-level formatting places a file system on the disk. In the abstract, a file system is composed of a table to track file metadata (name, size, permissions, etc.), and an index of free disk space. File systems operate on specifically size units of disk space. These units are called *clusters* in Windows, *blocks* in UNIX, or *allocation units* in the general. Each cluster consists of one or more hardware sectors. The size of a cluster will differ depending upon the size of the disk. For instance, on a high-density floppy disk a cluster consists of a single sector. For efficiency, clusters on larger disks will consist of multiple sectors, in multiples of two.

Most modern file systems we discuss in this chapter have a default cluster size of four sectors, or 4096 bytes, although it is configurable. The default cluster size on a Windows 98 FAT32 formatted file system is 64 sectors, or 32,768 bytes/cluster. Cluster size is important for investigators as larger cluster leave more residual forensic information on the disk. To understand why cluster size is important, let's consider three conceptual categories of disk space:

1. *Allocated space* is composed of clusters allocated to a file and that are tracked by the file system.
2. *Unallocated space* is composed of clusters not in use by a file. Unallocated space may contain residual information, e.g., from deleted files. We provide greater detail on unallocated space, and how to recover files in unallocated space, in the *Physical Analysis* section.
3. *Slack space* "is the space left over between the end of the data and the end of the last cluster or block" (Kruse & Heiser, p. 75, 2002). Slack space may contain residual information, e.g., from files previously deleted but which have been partially overwritten. We provide greater detail on slack space in the *Physical Analysis* section.

The procedures used to analyze the contents of each of the categories of space will differ. A logical analysis, which views our forensic image from the perspective of a file system, only examines information in allocated space. In contrast, a physical analysis allows us to examine information in unallocated and slack space as well.

Mounting the Image

We must *mount* our forensic image to access the file system contained therein. Recall that mounting disk or image makes a file system available to the operating system's kernel. Once an image is mounted, we can use any tools that we would normally use to work with files (search, view, sort, print, etc.). We want to ensure that we do not (cannot) change our forensic image, so we mount our forensic image in read-only mode. This guarantees that we do not change anything on our image as we are analyzing it.

We use the `mount` command to mount our forensic image to an existing directory.

```
# mount -t vfat -o ro,loop image.dd image/
```

We first create a directory called '*image'* (which we could have named anything). Next we execute the `mount` command with the following arguments: `-t` specifies the type of file system on the image, here `vfat` (virtual FAT, the same as FAT only able to understand long file names of 256 characters). The `-o` specifies options; here we want to mount the image read-only (`ro`), and we specify `loop` to interpret the image as if it contains a file system. The next arguments are the image to mount (`image.dd`), followed by the directory on which to mount the image (`image/`). If we receive no error message the image should be mounted.

Depending upon the situation, it may be desirable to create a list of all the files contained on a disk. (Several law enforcement personnel have relayed stories where a prosecuting attorney demanded a list of all files on a disk, so this does occur in practice. This procedure is also listed as an important forensic procedure on the IACIS site.) We can create a list of files and their associated hashes with the *file* and *md5sum* commands:

```
# find / -type f -print0 | xargs -0 md5sum > /evidence/files.md5
```

`find` starts searching for regular files at the root directory, and pipes a list of the files as a single line (via the `-print0` argument) to the `md5sum` command. The results are directed to a file – not in a directory on the mounted image of course -- for safe keeping. Here is a snippit from the *files.md5* file.

```
82f28b86ed26641a61a0b7a3ee0647a0  ./agenda.doc
093aa48b0b7ae587b9320ada28ae600a  ./suzy.doc
a6fff9e1af9393d7cb1d367f407250a0  ./1034.md5
3fe0b92fd2e93aa125e7d1a2c9508963  ./foo.txt
b9e5e46186f9e92d908feccf2aa2dd82  ./folder/.foo
e65ad7ea32ec3c21a4eb5e7296c1aa0c  ./folder/Yahoo.aba.txt
7349b1a5429cb4a7b36796444eb88528  ./folder/2004_03_01_Minutes.doc
31efc94982e64ec04a30768fe799f3fb  ./folder/grandmaletter.txt
4ef2b14aa970dc14bb260dcd7ba67ba5  ./folder/school.ppt
23958202e2e750090d60f26911842722  ./folder/.hiddenfile
8dad5d4b67ecdd7a0ba5d0d943edabbb  ./bagheera.txt
0c8fb94c2b437896aa2d36ba7d3a2cab  ./list.of.words
```

Once our image is mounted read-only we can conduct searches, view or print the files, and treat the contents of the image as if it were a live mounted disk, without worrying about altering any files on the image because we mounted our image in read-only mode.

We can unmount the image using the *umount* command (note: not unmount,):

```
# umount image/
```

Reducing our Search Space

What forensic procedures do we perform first? The answer is partly determined in part by the circumstances of our investigation. If we are investigating someone for distributing child pornography then the primary evidence will be graphical images. (There may be other important evidence that should not be overlooked, including emails, electronic documents, etc.) In contrast, an E-Bay Internet fraud case we may be more concerned with email messages, electronic documents, and the contents of Internet Browser history and temporary files. Let the circumstances dictate the approach.

Simple cases where we know exactly what evidence we are looking for may take a half day. Complicated cases involving multiple hard drives and where we are not exactly sure what constitutes evidence may take weeks or even months. We want to make the best use of our time. The first activity is to distinguish which files are of probative value and which are not. The reason is that most computers contain anywhere from 10,000 to hundreds of thousands (or more) files. At some point we will be forced to conduct a manual analysis of scores of files. We can make efficient use of our time if we can reduce to a manageable size the number of files we must manually analyze. Computer forensics is a very tedious and time consuming task, and therefore the more we can use our tools to automate the process of identifying potential evidence the better. One automated way of filtering files is through a hash analysis.

Hash Analysis

A hash analysis compares the hashes of the files to a set of hashes of files of a known content. Files in a hash set typically fall into one of two categories *known* or *notable*. Known files are files that can be ignored, such as typical system files (iexplore.exe, winword.exe, explore.exe, and so on). Notable files are ones that have been identified as illegal or inappropriate, such as hacking tools, pictures of child pornography, and so on. A hash analysis automates the process of distinguishing between files that can be ignored while identifying the files known to be of possible evidentiary value. Once the known files have been identified then these files can be filtered. Filtering out the known files may reduce the number of files the investigator must evaluate by half or more.

Hash analysis may be useful in an organizational setting in determining if there is any corporate espionage. Companies may be concerned that insiders are emailing intellectual property to competitors. Company IT staff could make a hash set by hashing all critical intellectual property-related documents. These hashes could then be compared against the hashes of files on the employees' computer to determine if the documents are located on the disk. (Note: before doing this consult legal counsel as there are legal ramifications for this type of activity.)

Recall that only a file's contents, not its metadata, are used in calculating a file's hash value. (It is important to understand this point as subjects will try to hide evidence by changing various file attributes such as files' names, attributes, etc.) We conduct a small experiment to demonstrate this claim for the readers. We take a file and make three copies of it [1], [2], [3]). We change the first file's name [4], the second's owner and permissions [5], [6], and leave the third unchanged. We then calculate the hash value for all three files [7]. Note that the hashes are the same. This fact is helpful when subject's have changed

files' names, e.g., the notable files described above, in order to hide their true content and identity, as we will demonstrate shortly

```
[1] jpc@simba:~> echo 'hello world\!' > file.1
[2] jpc@ simba:~> cp file.1 file.2
[3] jpc@ simba:~> cp file.2 file.3
[4] jpc@ simba:~> mv file.1 file.changed
[5] jpc@ simba:~> chown jpc.users file.3
[6] jpc@ simba:~> chmod 751 file.2
[7] jpc@ simba:~> md5sum file.changed file.2 file.3
    7cf0564cb453a9186431ee9553f7f935  file.changed
    7cf0564cb453a9186431ee9553f7f935  file.2
    7cf0564cb453a9186431ee9553f7f935  file.3
```

To perform a hash analysis we need to specify a list of MD5s. In this instance say we are interested in finding several known bad files, including several files from a rootkit as well as several illegal images. Here are the MD5s that compose our file (KNOWN.BAD) of notable files:

```
f53ce230616c1f6aafedf546a7cc0f0f  Trojan ps
77f7628ee6fa6cd37ee8b06278149d1d  Trojan netstat
64a3877b3105cd73496952c1ef8f48e8  Trojan ls
41791681dff38e3a492c72d3e7335f82  Trojan lsof
bbf3aeb654477c4733bddf9a6360d2c5  Illegal Image
2eff0db0a3cac3fc08add30e21257459  Illegal Image
d297c866310377f10b948d53b798c227  Illegal Image
```

We then run *md5deep* (md5deep.sourceforge.net) against all of the files in the directory, specifying the file KNOWN.BAD file for comparison.

```
pc@simba:~/chapter.stuff> md5deep -r -m KNOWN.BAD *
/image/chapter/.x
/image/chapter/misc/.y
/image/chapter/misc/preview.png
/image/chapter/misc/stuff.java
/image/chapter/misc/subset/bar
/image/chapter/preview.png
/image/chapter/large.jpg
/image/chapter/README.txt
```

The `-r` arguments indicates that md5deep run recursively (recurse through directories). The `-m` argument indicates the file containing the list of known files. Note that this list may contain either known good (e.g., Windows system files), or known bad files (hacker tool kits, illegal images, etc.).

Md5deep reads in the list of known hashes and then proceeds to hash each file in the path indicated at the command line. It compares the file's hashes with the contents of the list of known hashes. If a match occurs, it lists it on standard out (the screen). This procedure is essentially how all commercial forensic applications hash analyses function. As demonstrated above, *md5deep* found all of the files on the list of known hashes. Note that it appears that someone has changed the names of some of the notable files as a means of hiding their identity.

The National Institute of Standards and Technology (www.nist.gov) develops and maintains a very large set of hashes called the *National Software Reference Library*. (www.nist.gov/nsrl) The NSRL contained over 6,000,000 hashes as of early 2004.

Another large hash set is the *Hashkeeper* hash set, which can be downloaded at www.hashkeeper.org. An investigator or IT staff can build a custom hash set easily by using the procedures outlined above and elsewhere in this chapter, based on the companies need to protect intellectual property,

Signature Analysis

A *signature analysis* is an automated procedure for identifying potential evidence. A *file signature* is a header or footer (or both) within a file that indicates the application associated with a typical file, i.e., or the 'type' of file. For instance, we opened three files in a hex editor (Figure 8): a Word document, a JPG graphic, and an Adobe Acrobat File. The signatures (in hexadecimal) are:

- `25h 50h 44h 46h` for Adobe Acrobat Reader files (PDF).
- `FFh D8h FFh`, for JPEG graphical files; and
- `D0h CFh 11h E0h A1h B1h 1Ah E1h` for Microsoft Office files;

(Numbers in hexadecimal format are distinguished from decimal format by or appending with an `h`.)

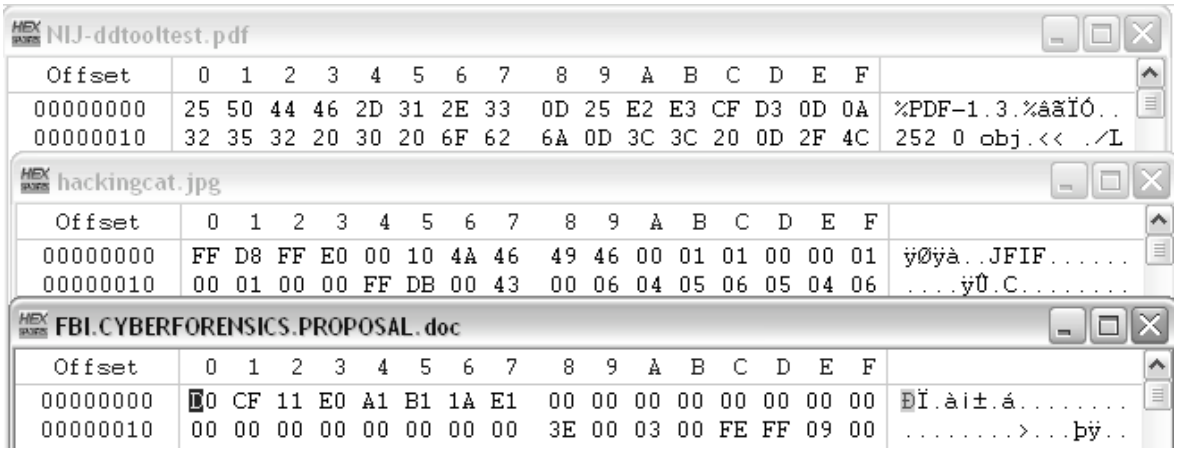

Figure 8. File signatures for three files

File signatures are useful for evaluating whether a subject is attempting to 'hide files in plain sight' by changing file extensions. For instance, renaming a graphic *naked_body.jpg* to *homework.doc* can be effective in hiding a file from prying -- albeit naïve -- eyes. A cursory examination of files in a file manager will not reveal the fact that the *homework.doc* file is *not* a Word document but rather a graphical file. To make matters worse, Windows Explorer will happily display the graphical file that has a *.doc* extension with a Word icon, confirming to the user that the file is what it purports to be. This is true even if we request a thumbnail view in Windows Explorer. Only if the graphical file has a graphical extension (e.g., GIF, BMP, PNG, JPG, etc.) will it display as a graphical thumbnail.

How does an investigator find these 'hidden' files? We simply compare the file's extension with its corresponding file signature. If the two match, then no effort was made to obscure the file type. If there is a mismatch between the extension and signature, then the file should be exposed to closer examination.

To illustrate how simple a hiding technique this is, the Figure 9 is a directory listing in Windows Explorer that shows files with several different extensions. We changed the extensions of several files to disguise their true identity. Can you tell which files were changed?

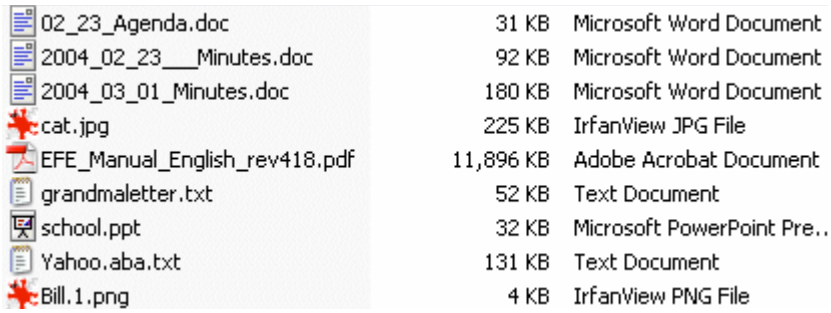

Figure 9. Viewing the contents of a floppy disk under Windows Explorer

The `file` command uses several sources of information, including the files signature, to verify a file's type. The example below illustrates the use the of the `file` command against the files from Figure 9 above.

```
jpc@simba:~/chapter.stuff/> file *
02_23_Agenda.doc:              Microsoft Office Document
2004_02_23___Minutes.doc:      Microsoft Office Document
2004_03_01_Minutes.doc:        Microsoft Office Document
cat.jpg:                       PDF document, version 1.3
EFE_Manual_English_rev418.pdf: PDF document, version 1.3
file.script:                   empty
grandmaletter.txt:             Microsoft Office Document
school.ppt:                    JPEG image data, JFIF standard 1.01
Yahoo.aba.txt:                 raw G3 data, byte-padded
```

We ran `file` is run against every file in the current directory. We then manually compared the extensions against the known file type. (We could create a script to do this for us; however, this is beyond the scope of this chapter.) For example, the file *course.doc* exhibits the Word extension, and the signature confirms it is a Microsoft Office Document. The file *grandma.txt* exhibits a text file extension; however, the signature indicates that it is a JPG graphical file. The file *homework.doc* has a Word extension, but its signature also indicates it to be a JPG graphical file. The remaining files appear to be what they claim.

How much credence should we put in the extension to denote the type of file? The simple answer is: none. As demonstrated, changing the file extensions is a simple, and often successful, means of hiding inappropriate or illegal files.

Searching a Forensic Image

Two common search tasks involve searching for specific keywords within documents, or searching for particular types of files (e.g., graphical files in a child pornography case, or Excel, Quicken, or Money files in a money-laundering case.) We first describe how to conduct keyword searches, followed by searching for particular types of files.

Keyword Searches

Once the investigator reduces the search space by identifying and filtering known files, as well as identified suspect files via signature analyses, she can turn her attention to searching for specific keywords within the forensic image.

Below we use the *grep* utility to search for keywords on the image. *grep* has the capability to search for multiple keywords simultaneously. This is accomplished by creating a text file containing a list of keywords, and then using the `-f` flag to indicate that we are using a file, instead of a single keyword, as input to *grep*. In this example our keyword text file contained the following key words, one per line, with no trailing blank line: marijuana, crack, crank, cocaine, oxycontin.

Again, it is important that there is *no blank line* at the end of our keyword file. Had we not used a file for our keywords we would have had to perform five separate single keyword searches.

Now we are ready to search our forensic image for the keywords. We execute the following command:

```
# grep -i -r -f keywords /image/* > /evidence/grep.results
```

The flags are interpreted as follows:

- `-i` indicates case insensitive search, thus 'cocaine,' 'COCAINE.' and 'CoCainE' are the same.
- `-r` indicates a recursive search, i.e., traverse all of the subdirectories beneath the current directory.
- `-f` indicates the next parameter is the file containing our keywords.

We redirect the results of our `grep` search into a file so that we can more closely analyze the results and print it out if need be. Abbreviated results are displayed below:

/mnt/evidence/bruce: WASHINGTON, D.C. U.S. Representative issued the following statement regarding the GAO Report to Congress titled, OxyContin Abuse and Diversion and Efforts to Address the Problem:

/mnt/evidence/bruce:This report reinforces what I suspected all along: Purdue Pharma has engaged in highly questionable practices regarding the marketing of OxyContin, leaving a plague of abuse and broken lives in its path....

/mnt/evidence/dynamic.dll:Cocaine Anonymous is a fellowship of men and women who share their experience, strength and hope with each other that they may solve their common problem and help others to recover from their addiction. The only requirement for membership is a desire to stop using cocaine ...

/mnt/evidence/homework.doc:OxyContin is a trade name for the drug oxycodone hydrochloride. Manufactured by Purdue Pharma L.P., OxyContin is a controlled-release form of oxycodone prescribed to treat chronic pain. When used properly, OxyContin can provide pain relief for up to 12 hours ....

/mnt/evidence/todo:marijuana: facts for teens…

/mnt/evidence/todo:teen boy qoute that reads I used to be real athletic. When I started using drugs, I just stopped playing all together because I thought I had more important things to do. Q: What are the short-term effects of marijuana use?...

/mnt/evidence/todo:A: The short-term effects of marijuana include:...

`grep` found instances of the terms marijuana, cocaine, and OxyContin in files contained on our forensic image. Note that there are several different cases used in the spelling of the terms (e.g.., 'Cocaine,' 'cocaine,'). A case sensitive search would have failed to find several instances of the keywords, therefore, it is usually best to include the `-i` flag if case does not matter.

Finding Files by Type

Say we have been asked by law enforcement to find all graphics files on a subject's hard drive that contains over 500,000 files. What would be the most efficient means of conducting this search? Clearly we want to automate as much of the search as possible. We could search for files with the appropriate graphical file extension, for instance:

```
# find / -type f \( -name '*.gif' -or -name '*.jpg' -or -name '*.bmp' -or
-name '*.png' \)
```

This command finds all files with the GIF, JPG, or BMP extension. This command would *not* find the files with changed extensions, however. One way find graphic files whose extension has been changed is to combine three GNU utilities: `find`, `file`, and `grep`. The best way to explain the combined use of these commands is through a demonstration.

Our goal is to find all graphical files regardless of extension. We want to do the following:

Step 1: Use the `find` command to find all regular files on the hard drive (as opposed to directories, special devices, and so on). Pipe the results of this command to the:
Step 2: `file` command, which returns the type of file using header information. Pipe the results of this command to the:
Step 3: `grep` command to search for graphical-related keywords.

Here is our command to perform the steps above:

```
# find /image -type f ! \( -name '*.jpg' -or -name '*.png' -or -name
'*.bmp'  -or -name '*.tiff' \) -print0 | xargs -0 file | grep -f
keywords.txt
```

The */image* argument specifies the directory in which to start. The argument `-type f` specifies that are we interested in regular files as opposed to special files such as devices or directories. The `find` command is recursive by default so it is essentially recursively finding all files beginning at the */image* directory. The exclamation mark (!) modifies the contents within the parenthesis, and says that we want to process files whose extension is not *.jpg, or *.png, or *.bmp, or *.tiff. The `-print0` is a special formatting command that is required to format the output of `find` for piping to the next command.

We pipe the results of `find` to `xargs -0`, which hands each file from the previous command to `file`. `file` evaluates each file's signature, returning a description of the type of file. These results are piped to `grep` to search for the specific keywords that are

contained within the *keywords.txt* file. The arguments for `grep` include `-i` for case insensitive search, and the `-f keywords.txt`, the file containing the list of keywords: 'PNG,' 'GIF,' 'bitmap,' 'JPEG,' and 'image.'

The results are as follows:

```
# find /image -type f ! \( -name '*.jpg' -or -name '*.png' -or -name
'*.bmp'   -or -name '*.tiff' \) -print0 | xargs -0 file | grep -f
keywords

./agenda.doc:          PC bitmap data, Windows 3.x, 382 x 61 x 24
./suzy.doc:            PNG image data, 571 x 135, 8-bit/color RGB
./folder/.foo:         PC bitmap data, Windows 3.x, 536 x 177 x 24
./folder/school.ppt:   JPEG image data, JFIF standard 1.01
./folder/.hiddenfile:  PNG image data, 351 x 374, 8-bit/color RGB,
./bagheera.txt:        PC bitmap data, Windows 3.x, 536 x 307 x 24
./9.11.xls:            JPEG image data, EXIF standard, 10752 x 2048
./list.of.words:       PNG image data, 571 x 135, 8-bit/color RGB
```

We found eight graphical files, including instances of JPEG, PNG, and Bitmap files. Note each of the files found were graphical files with misleading file extensions. What can we deduce from this result? Because applications will not arbitrarily change an extension of a graphical file, an investigator might reasonably deduce that a user has manually renamed the files, possibly in an attempt to hide their nature. This result is not incontrovertible, but would warrant further investigation.

Email Searches

Email is ubiquitous, supplanting regular mail as a preferred form of communication for many. Email can be a rich source of evidence for many types of investigations.

There are several email applications available in Windows, the most popular of which are Microsoft Outlook and Outlook Express, the latter of which is purportedly the most popular email client as it is comes with every version of Windows. Microsoft Outlook is a full-fledged personal information manager that includes email, calendar, contact list, task list, and scheduler. There are several, less popular, email applications available on other platforms, including Eudora, Netscape Mail, and Mozilla Mail. Each of these applications includes an address book or contact list that may prove useful in an investigation.

Some email applications store messages in proprietary binary formats, including Outlook, Outlook Express, and Eudora. Outlook uses a proprietary format that is different from its sibling Outlook Express. Netscape and Mozilla mail store mail in a non proprietary *mbox* format that is easily readable in a text editor.

To conduct an email investigation we must locate the mailbox files. The mailbox locations differ depending upon the version of Windows and the application used:

- Outlook Express
  - Windows 2000/XP
    - C:\Documents & Settings\<*username*>\Local Settings\Application Data\Identities\<*unique string*>\Microsoft\Outlook Express\
  - Windows NT
    - C:\winnt\profiles\<*username*>\Local Settings\Application Data\Identities\<*unique string*>\Microsoft\Outlook Express\
  - Windows 95/98/ME

        - C:\Windows\Application Data\Identities\<*unique string*>\Microsoft\Outlook Express

- Netscape/Mozilla Mail
  - Windows 2000/XP
    - C:\Documents & Settings\Application Data\Mozilla\profiles\<*username*>\<*unique string*>.*slt*\Mail
  - Windows NT
    - C:\winnt\Application Data\ Mozilla\ profiles\<*username*>\<*unique string*>.*slt*\Mail
  - Windows 95/98/ME
    - C:\Windows\Application Data\ Mozilla\profiles\<*username*>\<*unique string*>.*slt* \Mail

Outlook Express stores email messages and folders in files with a *dbx* extension. Each folder has a corresponding *dbx* file, whose name coincides with the folder's name. For example, the *outbox.dbx* file corresponds to the Outbox folder (http://mail-repair.com/outlook-express-repair.html). An investigator must use an application that understands the *dbx* proprietary format to extract the folders and messages in a human-readable format. The simplest method is to copy *dbx* or *pst* files to another Windows machine that contains the Outlook application, and import the appropriate application, either Outlook Express (*dbx* files) or Outlook (*pst* files).

A second alternative is to use a non Microsoft application that understands the mailbox formats and has the capability of extracting the messages and folders. For example, LibPST (http://sourceforge.net/projects/ol2mbox/) is an open source utility that converts messages from Outlook *PST* format to a standard *mbox* format. In our experiments we used LibPST to extract the contents of the Outlook mailbox. This demonstrates it is always good to have a wide variety of tools in your forensic toolkit. Below is a sanitized email that we recovered with LibPST.

```
From "Philip Craiger" Wed Jan  09:54:24 2004
X-Apparently-To: philip@craiger.net via web12824.mail.yahoo.com; Wed, 07
Jan 2004 07:54:30 -0800
Return-Path: <philip@craiger.net>
Received: from lakemtao06.cox.net (68.1.17.115)
  by mta1-vm3.mail.yahoo.com with SMTP; Wed, 07 Jan 2004 07:54:29 -0800
Received: from craiger.net ([68.13.130.154]) by lakemtao06.cox.net
    (InterMail vM.5.01.06.05 201-253-122-130-105-20030824) with ESMTP
     id <20040107155429.VISE24575.lakemtao06.cox.net@craiger.net>;
     Wed, 7 Jan 2004 10:54:29 -0500
Message-ID: <3FFC2BB0.2060702@craiger.net>
Disposition-Notification-To: Philip Craiger <philip@craiger.net>
Date: Wed, 07 Jan 2004 09:54:24 -0600
From: Philip Craiger <philip@craiger.net>
Reply-To:  philip@craiger.net
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.6b)
Gecko/20031205 Thunderbird/0.4
X-Accept-Language: en-us, en
MIME-Version: 1.0
To:  xxxxx.xxxxxxxx@SMU.CA, xxxxxxx xxxxxxx <xxxxxxx@xxxx.cas.usf.edu>
Subject: BIO
Content-Type: text/plain; charset=us-ascii; format=flowed
Content-Transfer-Encoding: 7bit

Hi ya'll,
```

```
I won't be able to make it this Friday; I'm going out of town.

Philip
```

Web-based Email

Investigators are likely to encounter web-mail, the most common of which are Yahoo Mail (my.yahoo.com) and Hotmail (www.hotmail.com). Web-mail messages are stored in *html* format with the extension *html* or *htm* and are thus readable with any web browser. The messages that are downloaded from or uploaded to the Web are stored in the four Windows Temporary Internet Folders (discussed in more detail below).

We conducted a series of experiments to determine the file names associated with each of web-mail messages from Yahoo! Mail and Hotmail. The results of our investigation are displayed in Table 2. These names may be used in conjunction with the *grep* search to identify the use of web-mail messages.

File Content	Yahoo Mail	Hotmail
Login page	login[#].htm	uilogin[#].htm
Home page	Welcome[#].htm	mhome[#].htm
Inbox/Folder	ShowFolder[#].htm	HoTMail[#].htm
View Message	ShowLetter[#].htm	getmsg[#].htm
Compose Message	Compose[#].htm	compose[#].htm

Table 2. Yahoo Mail and Hotmail filenames.

We can use any web browser to view these html files. In Figure 10 we opened a web-mail message from our Temporary Internet Folders (note the message has been sanitized of names).

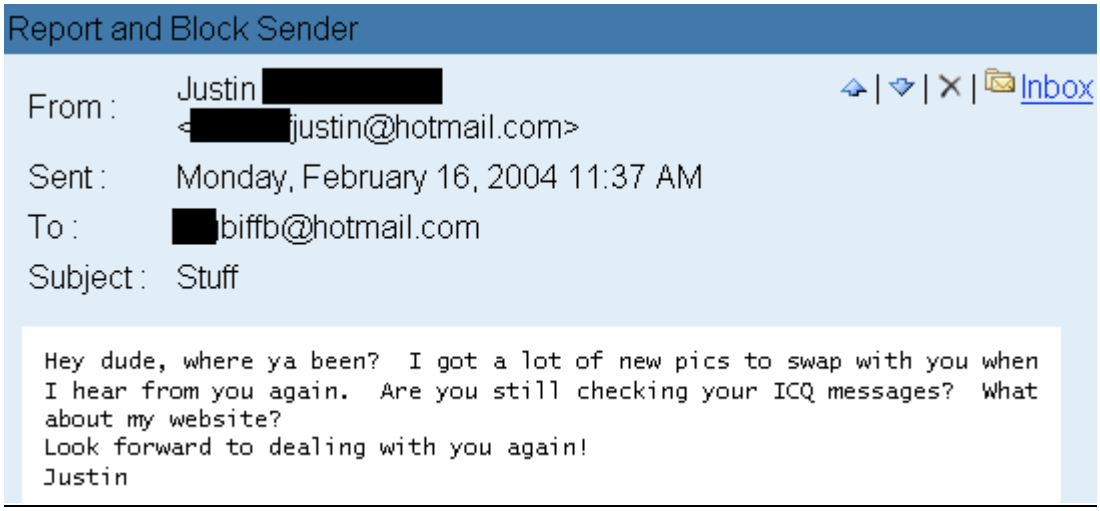

Figure 10. Incoming HotMail Message saved in a Temporary Internet Folder

Unless a suspect took overt means to remove these files from the Temporary Internet Folders, we can easily access these files via a web browser.

The Temporary Internet Folders may contain hundreds of files with an *htm* extension. To find the relevant email-related files the investigator can open up each file in a browser and conduct a manual search, or use `grep` to search for the appropriate file names (see Table 2 above). Note that the terms 'Yahoo' or 'Hotmail' will appear somewhere in the email files. We use this fact in our command below to search for files from Hotmail Mail.

To find all Hotmail files we use the following command line to show all of the related files in the Temporary Internet Folders.

```
linux:~/Temporary Internet Files/Content.IE5 # find . -type f \( -name
'getmsg*.htm' -or -name 'uilogin*.htm' -or -name 'mhome*.htm' -or -name
'HoTMail*.htm' -or -name 'compose*.htm' \)

./31n6dkxa/getmsg[1].htm
./31n6dkxa/compose[1].htm
./2l4z6jel/uilogin[1].htm
./ybcpq9sz/compose[1].htm
./ybcpq9sz/compose[2].htm
./ojavstch/compose[1].htm
./ojavstch/uilogin[1].htm
./ojavstch/getmsg[1].htm
```

The directories with the funny looking names are the Temporary Internet File folders.

The Windows Swap File

A swap file is virtual memory that is used as an extension of the computer systems RAM (whatis.techtarget.com). Typically, the least recently used contents of RAM are paged out (i.e., swapped) to the swap file, and are read back into RAM on an as-needed basis. The swap file can contain evidence that has been previously removed, even if the file was forensically wiped from a hard drive. Unless of course the swap file was forensically wiped, then the file within the swap is unrecoverable, although we may still find copies in unallocated or slack space.

The Windows swap file under Windows 9x/ME is named *win386.swp* and is typically found in the Windows root directory, although this may be changed by the administrator. Under Windows NT/2000/XP the file is named *pagefile.sys* and is typically found in the root directory (e.g., C:\), and again may be changed to a different location.

The swap file is a binary file. We use the `strings` command along with `grep` to extract interesting information of possible evidentiary value. The `strings` command reads in a file and extracts the human-readable text of a certain length, the default of which is four characters. In the example below we extract the human-readable text, and then use `grep` to search for the string 'hacker.' Note that we request two lines of context using the –C argument, which gives us two lines before and after the actual search string. There were approximately 25 hits extracted from the swap file.

```
linux:/jpc/chapter/windows # strings win386.swp | grep -C 2 hacker

// lines deleted for brevity

--
ing latest amitis
/amitis/serv
immortal-hackers.com
```

```
.I$I
llium-e
--
fg32.exe
boot
hacker
syscfg32
BotV0.3
--
echo @echo off >>
ces\firewall
hackers and viruses
ControlSet\
\firewall.exe
--
iiiii
for WiNis       by F-king
root@hacker
\HATREDFIEND
/\/\ENDOFFILE/\/\
```

// lines deleted for brevity

I know what you did with your computer last summer…

Every time a user uses Windows Explorer or Internet Explorer access a file or web site, digital traces of these activities are placed on the hard drive. Most of these artifacts are kept in *index.dat* files. An *index.dat* file is a binary file that tracks user activities: files opened in Window's explorer, web pages opened in Internet Explorer, and so on.

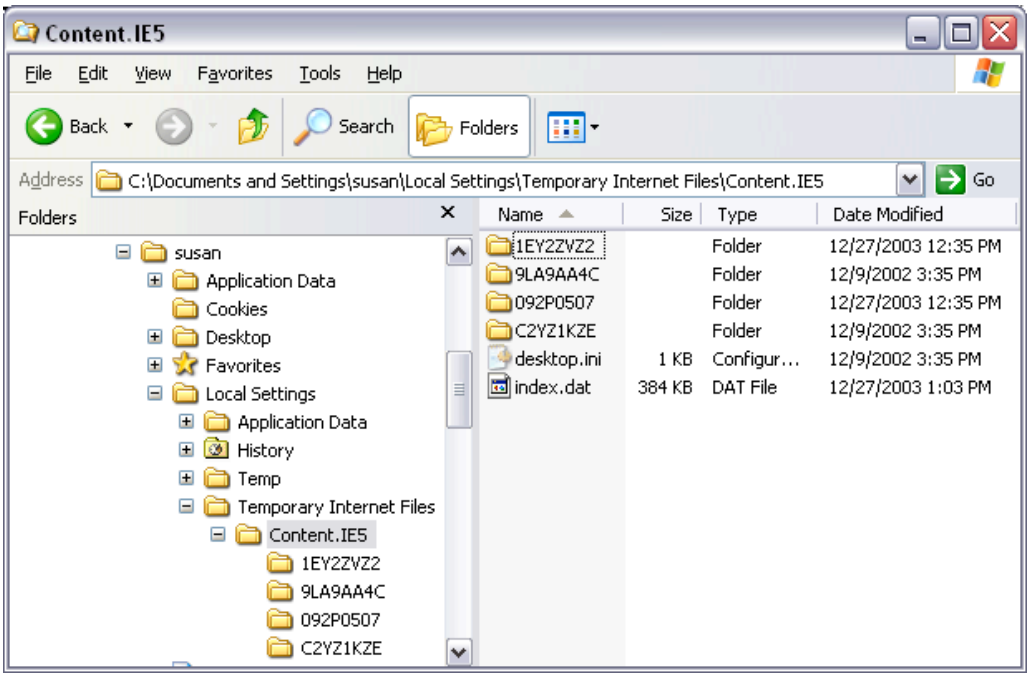

Figure 11. Location of an *index.dat* file in Temporary Internet Folders

Each time a file is accessed via Window's Explorer (as shown in Figure 24) or Internet Explorer, a record is placed in an *index.dat* file.

Window's uses several *index.dat* files to track various activities on the computer. The location of these files will vary depending upon the version of Window's used. Table 3 shows these locations.

Windows	Locations
95/98/ME	\Windows\Temporary Internet Files\Content.IE5\ \Windows\Cookies\ \Windows\History\History.IE5\
NT	\Winnt\Profiles\<username>\Local Settings\Temporary Internet Files\Content.IE5\ \Winnt\Profiles\<username>\Cookies\ \Winnt\Profiles\<username>\Local Settings\History\History.IE5\
2000/XP	\Documents and Settings\<username>\Local Settings\Temporary Internet Files \Content.IE5\ \Documents and Settings\<username>\Cookies\ \Document and Settings\<username>\Local Settings\History\History.IE5\

Table 3: Locations of index.dat files (adapted from Jones, 2003).

Index.dat files are binary files and therefore not in human-readable format. *Pasco* (www.foundstone.com) is a small open source application that parses the contents of *index.dat* files, and outputs the results into a tab delimited file. To illustrate, we ran *pasco* against the *index.dat* file from a system, redirected the output to a file that we subsequently imported into a spreadsheet application.

```
# pasco index.dat > evidence.txt
```

	A	B	C	D
1	History File: index.dat			
2				
3	TYPE	URL	MODIFIED TIME	ACCESS TIME
4	URL	Visited: jpc@file:///C:/Documents%20and%20Settings/jpc/Desktop/chapter/computer.forensic	Sun Dec 28 16:46:52 2003	Sun Dec 28 16:46
5	URL	Visited: jpc@file:///C:/Documents%20and%20Settings/jpc/Desktop/The%20Eagles%20-%20	Wed Jan 7 10:00:59 2004	Wed Jan 7 10:00:
6	URL	Visited: jpc@file:///C:/Documents%20and%20Settings/jpc/Desktop/forensics.grades.fall.03.xl	Sun Dec 21 16:46:03 2003	Sun Dec 21 16:46
7	URL	Visited: jpc@file:///C:/Documents%20and%20Settings/jpc/Desktop/secret.service.pdf	Mon Dec 29 15:08:26 2003	Mon Dec 29 15:08
8	URL	Visited: jpc@file:///C:/Documents%20and%20Settings/jpc/Desktop/chapter/i.encyc/TOC.doc	Mon Dec 29 15:17:09 2003	Mon Dec 29 15:17
9	URL	Visited: jpc@file:///C:/Documents%20and%20Settings/jpc/Desktop/chapter/chapter.12.29.v4.2	Mon Dec 29 15:34:12 2003	Mon Dec 29 15:34
10	URL	Visited: jpc@http://search.msn.com/results.aspx?srch=106&FORM=AS6&q=google+secret+s	Sun Dec 28 16:42:38 2003	Sun Dec 28 16:42
11	URL	Visited: jpc@http://www.google.com/search?hl=en&ie=UTF-8&oe=UTF-8&q=Computers+are	Sun Dec 28 09:44:04 2003	Sun Dec 28 09:44
12	URL	Visited: jpc@http://www.rcfl.org/training.htm	Sun Dec 28 09:43:14 2003	Sun Dec 28 09:43
13	URL	Visited: jpc@http://www.ndaa-apri.org/publications/newsletters/update_volume_15_number_	Sun Dec 28 09:43:25 2003	Sun Dec 28 09:43
14	URL	Visited: jpc@http://www.ndaa.org	Sun Dec 28 09:47:01 2003	Sun Dec 28 09:47
15	URL	Visited: jpc@http://www.ndaa-apri.org/index.html	Sun Dec 28 09:46:32 2003	Sun Dec 28 09:46
16	URL	Visited: jpc@file:///C:/Documents%20and%20Settings/jpc/Desktop/PGPFW658Win32.zip	Sat Dec 20 17:51:26 2003	Sat Dec 20 17:51:

Figure 12. A parsed *index.dat* file

This *index.dat* file in this example was from the *History.IE5* folder (see Table 3). Each row in the spreadsheet is an activity record that includes the type of access, the URL (which can be a regular file or a web site), the modified and access times, filename and directory (latter two not shown).

This particular *index.dat* contains information on files accessed from either Windows Explorer (the default file manager) or Internet Explorer, including:

- Files accessed and opened via Windows Explorer (Rows 4 through 9)
- Keywords used in searches over the Internet (Rows 10 and 11)
- URLs visited via Internet Explorer (Rows 12 through 15)

This *index.dat* file included over 487 files accessed within the last three weeks. An investigator can use the modified and last accessed times to determine the most recent date and time the user downloaded the file (modified time), as well as the last time the user visited the page or file (access time)

Cookies

According to www.cookiecentral.com:

> "Cookies are pieces of information generated by a Web server and stored in the user's computer, ready for future access. Cookies are embedded in the HTML information flowing back and forth between the user's computer and the servers. Cookies were implemented to allow user-side customization of Web information. For example, cookies are used to personalize Web search engines, to allow users to participate in WWW-wide contests (but only once!), and to store shopping lists of items a user has selected while browsing through a virtual shopping mall." http://www.cookiecentral.com/content.phtml?area=2&id=1

Unless a user's browser security is set to high, cookies are automatically – and quietly -- placed on the user's hard drive. Most users may be unaware that these cookies are being placed on their hard drives.

The cookies directory contains the individual cookies as well as an *index.dat* that consists of the activity records for each of the cookies in the directory, as shown in Figure 13. We used Pasco to parse the *index.dat*, the results of which are displayed in Figure 14. Note the activity records contain the cookies URL (from whence it came), the modified and access times, as well as the cookies file name.

```
E:\Documents and Settings\jpc\Cookies>ls
index.dat                       jpc@msn[1].txt
jpc@atdmt[2].txt                jpc@preview[2].txt
jpc@atwola[1].txt               jpc@questionmarket[1].txt
jpc@com[1].txt                  jpc@real[1].txt
jpc@cos[1].txt                  jpc@tucows[1].txt
jpc@dell[1].txt                 jpc@w[1].txt
jpc@ehg-ati.hitbox[1].txt       jpc@winamp[2].txt
jpc@fastclick[2].txt            jpc@www.ati[1].txt
jpc@geocities[1].txt            jpc@www.ipswitch[1].txt
jpc@hitbox[2].txt               jpc@www.pcmag[1].txt
jpc@info.winamp[1].txt          jpc@yahoo[2].txt
jpc@microsoft[1].txt
```

Figure 13 Cookie's directory

```
# galleta "jpc@microsoft[1].txt > ms.cookie.txt
```

	A	B	C	D	E	F
1	Cookie File: jpc@microsoft[1].txt					
2						
3	SITE	VARIABLE	VALUE	CREATION TIME	EXPIRE TIME	FLAGS
4	microsoft.com/	MC1	GUID=3b34e28b8c55e34f9ade8306293468a7&HASH=8be2&LV=20042&V=3	Sat Feb 21 18:46:06 2004	Tue Oct 3 07:00:00 2006	1024

Figure 14. Parsed cookie file

As shown in Figure 14 the cookie is partitioned into six or more pieces of information, including: a) the web site from which the cookie came; b) a variable and c) its associated value (which will differ on its meaning depending upon the web site), d) the creation time of the cookie, and its e) expiration date and time. Here the variable is apparently the GUID (globally unique identifier) for my copy of the Windows OS I'm running, along with a hash of its value.

Care should be taken when interpreting cookies to infer user activity as cookies may be placed on user's hard drive from third-party sources, i.e., from sources other than the web site visited.

Cookies can be found in the following locations:

Windows	Locations
95/98/ME	\Windows\Cookies\
2000/XP	\Documents and Settings\<username>\Cookies\

Table 4. Cookie file locations

Deleted Files and the INFO2 File

Files that are 'deleted' through *My Computer*, *Windows Explorer*, a Windows-compliant program, or any other way *except* from the command line, are removed from their original directory, and a copy placed in either the *Recycled Bin* (FAT32) or *Recycler Bin* (NTFS). A binary file named *INFO2* within each bin tracks important information about the deleted files, and may be an important source of evidence should the contents of deleted files be overwritten.

According to Microsoft (http://support.microsoft.com/default.aspx?scid=kb;en-us;136517&Product=w95), the following occurs when a file is deleted by one of the means described at the beginning of this section:

1. The deleted file is moved to the Recycled/Recycler Bin.
2. The following details are recorded in the INFO2 file for each deleted file:
    a. The index, i.e., the order in which the file was deleted
    b. The date and time the file was deleted
    c. Drive from which the file was deleted
    d. The full path
    e. The file size
3. The deleted file is renamed, using the following syntax:

D<original drive letter of file><#>.<original extension>

The <#> is the order in which the file was placed in the Bin. For example, a file named 'c:\My Documents\Florida.doc' is the first file placed in the Recycle Bin, it's name is 'DC1.doc.' A file named 'Beatles.MP3' deleted next would be renamed 'DC2.MP3.' If the files were on the Z drive, they would be renamed DZ1.doc and DZ2.MP3, respectively.

The length for an INFO2 record under Windows 95/98/ME is 280 bytes and 800 bytes under Windows NT/2000/XP (Sheldon, 2002). When a user empties the Recycled/Recycler bin, the files within the bin are removed, and the INFO2 file's contents are deleted.

Under NTFS each user is tracked by a security ID (or SID) which is how the computer refers internally to each user. This allows user activity to be tracked by the operating system. Under NTFS, every user has a folder named after his/her SID in the Recycler Bin. Each of the folders will have its own INFO2 file.

Figure 15 shows a Recycler Bin's contents from the command line. (We must demonstrate this from the command line as a user's SID is not displayed when viewed with any of Window's utilities). Note the SID-named folder.

```
E:\RECYCLER\S-1-5-21-1409082233-1592454029-839522115-1003>dir /a
 Volume in drive E is whammo
 Volume Serial Number is 787E-00E6

 Directory of E:\RECYCLER\S-1-5-21-1409082233-1592454029-839522115-1003

03/26/2004  09:12 AM    <DIR>          .
03/26/2004  09:12 AM    <DIR>          ..
03/09/2004  07:38 AM           400,896 De10.ppt
03/25/2004  08:08 AM            47,104 De11.xls
02/21/2004  12:56 PM             1,607 De12.lnk
02/14/2004  08:28 PM               630 De13.txt
02/20/2004  07:39 PM           205,595 De14.jpg
01/20/2004  04:05 PM           272,416 De15.EnScript
12/29/2003  03:12 PM         1,091,325 De6.pdf
01/11/2004  09:09 AM           129,577 De7.pdf
03/11/2004  07:22 PM           167,366 De9.torrent
03/26/2004  09:10 AM                65 desktop.ini
03/26/2004  09:12 AM             8,020 INFO2
              11 File(s)      2,324,601 bytes
               2 Dir(s)  24,252,698,624 bytes free
```

Figure 15. Recycler Bin on Windows XP <recycler.tif>

The times displayed are the last modification times for each file, *not* the time the file was created in the bin. Recall the deleted times are kept in the INFO2 file.

We can use *rifuiti* (www.foundstone.com) to interpret the binary contents of the INFO2 file. *Rifuiti* takes as an argument the name of the INFO2 file. We redirect the results to a tab-delimited file that we can then import into a spreadsheet application (as demonstrated in Figure 16).

```
# rifuiti INFO2 > deleted.txt
```

	A	B	C	D	E
1	INFO2 File: INFO2				
2					
3	INDEX	DELETED TIME	DRIVE NUM	PATH	SIZE
4	6	Fri Mar 26 15:11:02 2004	4	E:\Documents and Settings\jpc\Desktop\secret.service.guide.pdf	1093632
5	7	Fri Mar 26 15:11:02 2004	4	E:\Documents and Settings\jpc\Desktop\fed.rules.evide.pdf	131072
6	8	Fri Mar 26 15:11:07 2004	4		118784
7	9	Fri Mar 26 15:11:10 2004	4	E:\Documents and Settings\jpc\Desktop\Mandrakelinux-10.0-Community.torrent	167936
8	10	Fri Mar 26 15:11:18 2004	4	E:\Documents and Settings\jpc\Desktop\HONORS\UseCaseExample.ppt	401408
9	11	Fri Mar 26 15:11:24 2004	4	E:\Documents and Settings\jpc\Desktop\HONORS\HONORS Gradebook.3.23.xls	49152
10	12	Fri Mar 26 15:11:49 2004	4	E:\Documents and Settings\jpc\Desktop\Program\Current_v4_Scripts\Mozilla Firefox.lnk	4096
11	13	Fri Mar 26 15:11:55 2004	4	E:\Documents and Settings\jpc\Desktop\Program\Current_v4_Scripts\CYFOR.attendees.txt	4096
12	14	Fri Mar 26 15:12:01 2004	4	E:\Documents and Settings\jpc\Desktop\Program\Current_v4_Scripts\confiden.html.jpg	208896
13	15	Fri Mar 26 15:12:26 2004	4	E:\Documents and Settings\jpc\Desktop\Program\Current_v4_Scripts\Scripts\Examples\w4_lr	274432

Figure 16. Contents of INFO2 file viewed in a spreadsheet

Of what use is the INFO2 file? Even though its contents are deleted when the bin is emptied, we may be able to recover the contents from unallocated space during a physical analysis. We may even be able to recover the files that were emptied from the bin. (We

deal with the recovering of information from unallocated and slack space later in the chapter.) Note that we are still able to extract important information from the INFO2 file concerning deleted files, including: a) the date and time the file was deleted, b) the drive on which the file was deleted, c) the files original path, and d) the files size.

Application Residual Files

Many Windows applications create *temporary* files that are usually written to the hard drive. These temporary files are usually deleted from the hard drive once the application closes, or the user closes the file manually. As we have previously noted, deleting files only removes the pointer to the file, and marks the clusters occupied by the file as available: the information in the clusters is still recoverable until overwritten. (Note that when temporary files are deleted by applications they are *not* moved to the Recycler/Recycled Bins as described in the previous section).

Microsoft Office applications are very good about creating temporary files. Microsoft Word, for example, creates temporary files that consist of the contents of previous versions of a file. Temporary files are created when Word's *autosave* feature is turned on, or when the user manually saves the file. Word creates temporary files to provide some fault tolerance should Word or the operating system crash. In later versions of Office, if Word finds one of these files it recognizes this as an anomaly, and offers to recover the file.

Figure 17 illustrates several temporary files that Word created as we were editing a document. This explains why an investigator might find dozens of residual copies of the same text in unallocated space.

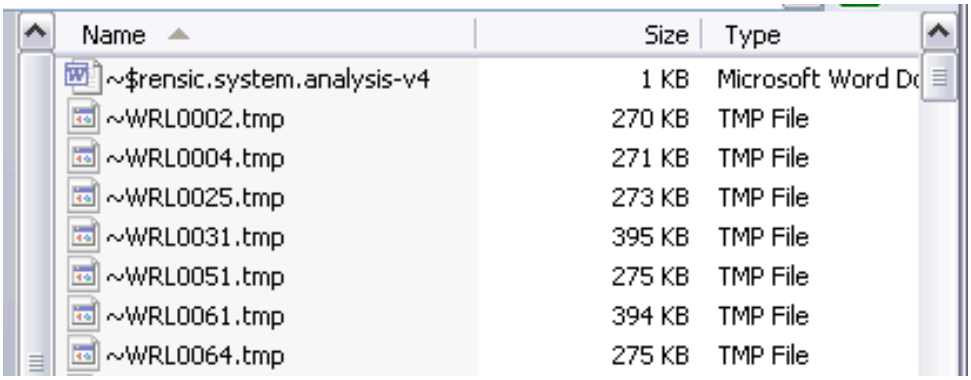

Figure 17. Microsoft Word Temporary Files

If an application or operating system misbehaves it may leave temporary files (usually, but not necessarily, noted with a TMP extension) in allocated space on the hard drive. We can use the `find` command to search for these files.

```
# find / -type f \( -name '*.TMP' -or -name '*.tmp' \)
```

This command will find all files with the *.TMP or *.tmp extension. Of course this only works if the temporary files are in allocated space (i.e., have not been deleted). Once the application closes these temporary files will be deleted by the application. If we need to identify temporary files that have been deleted we can use two procedures. The easiest

method is if we know part of the text of the document, we can use `grep` to search our forensic image (that is, *unmounted* image) for that text using a physical analysis.

Say we are told that subject created a counterfeit document using a word processor on his computer. We are also told the subject has a 'wipe' utility on the computer, and that a cursory examination of the computer showed no copies of the letter. As investigators we assume that the word processor diligently created temporary files of the counterfeit document. We can use parts of the threatening letters in a keyword search to see if we can recover some of the temporary files.

A significant content of a hard disk will be unreadable binary data. Therefore, let's first extract all of the human-readable text from our forensic image. We use the `strings` command to extract the human-readable text from our forensic image.

```
# strings evidence.dd > /evidence/evidence.str

# grep -b -I -C 1 "Philip" evidence.str > /evidence/search.philip
```

`strings` extracts strings of size 4 bytes or greater in the 7-bit ASCII range. We redirect the results to a file, which we then use in our `grep` search. The `-b` flag in the grep search specifies that we want the byte offset printed for each result, i.e., where in the file the match occurred. The `-C 1` specifies that we want one line of context before and after our match.

Figure 18 shows the results five matches at byte offsets 41260, 41404, 41786, 45606, 49673, with one line of context before and after each match. If we wish to further explore text associated with one of the matches we can use the byte offset of the file to extract more of the text.

```
41260:Dr. Philip Craiger
41279-bjbj
--
41346-September 3, 2003 2109 Sully Court        Bakersfield, CA  93311
41404:Dr. Philip Craiger                          |  email:  philip@craiger.net
41478-Associate Professor of Computer Science     |  web:    www.craiger.net
--
41716-Omaha, NE  68182                            |  PGP Key ID: 0x9B600586
41786:Dear Philip:
41799-Thank you for agreeing to serve as one of our contributors to The Handbook
 of Information Security.  Professor Bidgoli and I would like to formally invite
 you.
--
45045-, 554-3181, PKI173B
45065:Dr. Philip Craiger,
45086-HYPERLINK "mailto:pcraiger@mail.unomaha.edu"
--
49642-Computer and Network Forensics
49673:Dr. Philip Craiger
49692-Normal.dot
49703:J. Philip Craiger, Ph.D.
49728-Microsoft Word 10.0
search.philip lines 11-30/30 (END)
```

Figure 18. Results of grep search.<grep.search.results.tif>

Say we are interested in the text at offset 41786, which appears to be a letter of some sort. We can use `dd` to extract the text as demonstrated in Figure 32 below.

```
snoball:/evidence # dd if=evidence.str of=philip.txt bs=1 skip=41700 count=750
750+0 records in
750+0 records out
snoball:/evidence #
```

Figure 19. Command line to extract specific strings

The important arguments are bs, skip, and count. We set our block size to one so that we can ask for a specific number of characters (i.e., `bs=1` is equal to one character). We want to start extracting the data a little before our match at byte offset, so we chose to start at 41700. Note we specify offset 41700 because our block size is one. If we had used the default block size of 512 bytes we would be starting at offset 41700 x 512 or 21,350,400, definitely not what we wanted. The block size also refers to our count: we are extracting 750 characters (not 750 x 512). The 750 characters we extracted are show in Figure 20 below.

```
ebraska @ Omaha
Omaha, NE  68182                                |  PGP Key ID: 0x9B600586
Dear Philip:
Thank you for agreeing to serve as one of our contributors to The Handbook of Inf
ormation Security.  Professor Bidgoli and I would like to formally invite you.
The Handbook will be a three-volume, 2,400-page, 8.5
 x 11
 trim size reference source providing state-of-the-art information concerning the
 information, computer and network security with coverage of the core topics. The
 audience is four-year colleges and universities with Computer Science, MIS, IT,
IS, E-commerce, and Business departments, and public, private, and corporate libr
aries and a diverse group of professionals interested in this fast growing field.
 The Handbook will be avail
philip.txt lines 1-7/7 (END)
```

Figure 20. Results of string extraction

As you can see through several demonstrations `dd` is a very useful tool.

UNICODE

There is a potential problem because text is represented in various formats. ASCII text is represented in 8 bits, which limits the number of characters it can to represent 256, i.e., $2^8 = 256$. UNICODE is a 16-bit character representation that was created to overcome this limitation, and to allow for the ability to represent characters from various countries, given it can now hold 65,536 ($2^{16} = 65,536$). So ASCII text is represented in one byte, whereas UNICODE is represented in two bytes. The difference between the same text represented in ASCII and UNICODE can be seen in Figures 21 and 22.

```
00000010  00 00 00 00 45 3A 5C 44  6F 63 75 6D 65 6E 74 73  ....E:\Documents
00000020  20 61 6E 64 20 53 65 74  74 69 6E 67 73 5C 6A 70   and Settings\jp
00000030  63 5C 44 65 73 6B 74 6F  70 5C 73 65 63 72 65 74  c\Desktop\secret
00000040  2E 73 65 72 76 69 63 65  2E 67 75 69 64 65 2E 70  .service.guide.p
00000050  64 66 00 00 00 00 00 00  00 00 00 00 00 00 00 00  df..............
```

Figure 21. 8-bit ASCII text

```
00000120  A0 4A 47 8D 44 13 C4 01  00 B0 10 00 45 00 3A 00   JG▌D.Ä..°..E.:.
00000130  5C 00 44 00 6F 00 63 00  75 00 6D 00 65 00 6E 00  \.D.o.c.u.m.e.n.
00000140  74 00 73 00 20 00 61 00  6E 00 64 00 20 00 53 00  t.s. .a.n.d. .S.
00000150  65 00 74 00 74 00 69 00  6E 00 67 00 73 00 5C 00  e.t.t.i.n.g.s.\.
00000160  6A 00 70 00 63 00 5C 00  44 00 65 00 73 00 6B 00  j.p.c.\.D.e.s.k.
00000170  74 00 6F 00 70 00 5C 00  73 00 65 00 63 00 72 00  t.o.p.\.s.e.c.r.
00000180  65 00 74 00 2E 00 73 00  65 00 72 00 76 00 69 00  e.t...s.e.r.v.i.
00000190  63 00 65 00 2E 00 67 00  75 00 69 00 64 00 65 00  c.e...g.u.i.d.e.
000001A0  2E 00 70 00 64 00 66 00  00 00 00 00 00 00 00 00  ..p.d.f.........
```

Figure 22. 16-bit UNICODE text

The information for the figures was extracted from an INFO2 file.

The normal strings command as run above will not find the text in Figure 22 given that each character is represented by two bytes. In order to extract UNICODE characters the flags `-b l` (that's a lower-case 'L') must be used.

The command to extract all UNICODE characters from a file named "evidence.txt" would be:

```
# strings -a -b l evidence.txt > /evidence/evidence.unicode
```

Print Spool Files

When a file is printed in Windows it is first converted to an enhanced metafile (EMF) – a graphic image -- before it is printed. This file is written to the hard drive in the root Windows directory under *\system32\spool* directory (Windows XP and 2000). (The default print spool directory can be changed by changing the values in Windows registry). There are two files associated with each file printed: a header file with the extension SHD and the actual graphic image (EMF format) with the extension SHL. The SHD file contains information on the name of the file being printed, the name of the file to which it was printed, and the time and date stamp. The SHL file is the actual graphical file. If we can recover the SHL file we can print it. Collectively, these files are called *print spool files*, and demonstrate that particular documents were once printed from the computer. These files are typically found in the directory:

c:\<Windows root directory>\system32\spool.

If a user creates a document in Windows, prints the document, then uses a forensic wipe utility to write zeros over the clusters composing the file, an investigator can still access the document by recovering the deleted print spool files.

Once the file has been printed Windows will delete the header and EMF file. Until overwritten, the header and EMF file will remain in unallocated space, and are recoverable with the techniques demonstrated later. However, if something happens and the file fails to print, the header and EMF file will remain in the directory. A `grep` search for files ending in '*.SHD' or '*.SHL' will find these files.

Physical Analysis

A logical analysis can only examine the contents of allocated space: It cannot find files intentionally deleted by users, deleted temporary files, deleted print spool files, or other forms of information found in unallocated or slack space. The investigator must conduct a physical analysis to examine ambient information in the unallocated and slack space.

A physical analysis involves analyzing our evidence from a physical perspective, i.e., without regard to a file system. Thus, we do not mount our forensic image, but instead use a hex editor to view the forensic image as a single, flat file. This will permit us access to all categories of disk space: allocated, unallocated, and slack. Clearly this is advantageous as often a great deal of evidence may be found in unallocated and slack space. The draw back is that we are no longer dealing with a file system, and therefore, our analysis can be very tedious and complex, as we will see.

Figures 23 and 24 show physical view of the floppy disk image that we created earlier. (Legend: the far left side of the figure shows the offset from the beginning of the file in hex; the middle section is the contents of the file in hex at the offset; the far right side is the ASCII representation of the contents of the file at that offset.)

```
0002600: e545 414c 2020 2020 4a50 4720 1860 e253  .EAL    JPG .`.S
0002610: 9c2f 9c2f 0000 6b47 972f 0200 7a79 0000  ././..kG./..zy..
0002620: e549 4444 454e 2020 4a50 4720 18be e453  .IDDEN  JPG ...S
0002630: 9c2f 9c2f 0000 0000 212c 3f00 cae6 0100  ././....!,?.....
0002640: 5245 414c 2020 2020 444f 4320 18b6 e953  REAL    DOC ...S
0002650: 9c2f 9c2f 0000 6d51 9c2f 3301 0064 0000  ././..mQ./3..d..
0002660: 5355 5a59 2020 2020 444f 4320 1843 ec53  SUZY    DOC .C.S
0002670: 9c2f 9c2f 0000 6d51 9c2f 6501 2a8a 0000  ././..mQ./e.*...
0002680: 5245 414c 2020 2020 5458 5420 1800 f053  REAL    TXT ...S
0002690: 9c2f 9c2f 0000 6d51 9c2f ab01 a407 0000  ././..mQ./......
00026a0: 4752 4f43 4552 5920 5458 5420 1827 f253  GROCERY TXT .'.S
00026b0: 9c2f 9c2f 0000 6d51 9c2f af01 011a 0000  ././..mQ./......
00026c0: 5245 4745 5844 2020 5a49 5020 1886 f753  REGEXD  ZIP ...S
00026d0: 9c2f 9c2f 0000 6e51 9c2f bd01 5ddf 0100  ././..nQ./..]...
00026e0: 4e45 4544 5320 2020 5044 4620 1897 1054  NEEDS   PDF ...T
00026f0: 9c2f 9c2f 0000 646d 552f ad02 23c5 0600  ././..dmU/..#...
0002700: 4b45 5957 4f52 4420 4749 4620 1819 1a54  KEYWORD GIF ...T
0002710: 9c2f 9c2f 0000 7d51 9c2f 1006 6d56 0000  ././..}Q./..mV..
0002720: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Figure 23. Root Directory of the FAT formatted floppy disk <root.directory.tif>

Figure 23 displays part of the *root directory* of a FAT formatted floppy disk image as viewed from a hex viewer. A root directory is like a table of contents of a file system, it tracks metadata on each file including the filename, file size, time and date of creation, modification, and last access, where the file starts on the disk, etc. On our floppy disk each root directory entry is 32 bytes, or two lines in the Figure 23. The lines at offset 2700 through 2710 are the root directory entry for the file KEYWORD.GIF.

```
0004200: ffd8 ffe0 0010 4a46 4946 0001 0101 0048  ......JFIF.....H
0004210: 0048 0000 fffe 0019 2243 7265 6174 6564  .H......"Created
0004220: 2077 6974 6820 5468 6520 4749 4d50 22ff   with The GIMP".
0004230: db00 4300 0806 0607 0605 0807 0707 0909  ..C.............
0004240: 080a 0c14 0d0c 0b0b 0c19 1213 0f14 1d1a  ................
0004250: 1f1e 1d1a 1c1c 2024 2e27 2022 2c23 1c1c  ...... $.' ",#..
0004260: 2837 292c 3031 3434 341f 2739 3d38 323c  (7),01444.'9=82<
0004270: 2e33 3432 ffdb 0043 0109 0909 0c0b 0c18  .342...C........
```

Figure 24. FAT12 data area: Physical sector 33-logical cluster 2

Figure 24 shows the beginning of the data area which starts at offset 4200h. Note that the contents of this first sector of the data area contain the file signature `FFh D8h FFh` which is a JPEG graphic signature. Note that the data area on a floppy starts are physical sector 33, which translates to logical cluster 2. This will be explained in more detail in the section "Recovering Deleted Files" below.

From an examination of the root directory entry displayed in Figure 23 we can tell that the file at offset 2600-2610 has been deleted. However, Figure 24 shows that the file still resides on the disk. Two questions come to mind. First, how did we know that the file had been deleted from viewing the root directory? Second, can we recover the deleted file displayed in Figure 24?

What Happens when a File is Deleted?

Before we explain what happens to a file when it is deleted it is important to understand parts of a file system and how an operating system tracks files.

A disk formatted with a version of FAT is comprised of a reserved area and a data area. The reserved area consists of the following components:

- The boot record is the 1st sector of the disk.
- 1st file allocation table
- 2nd file allocation table (a backup to the first)
- Root directory
- Data area (on a floppy, it begins at physical sector 33/logical cluster 2. This will differ depending upon the size of the disk).

The root directory entry for a file contains information on the starting cluster of the file. The remaining clusters that compose the file are kept in the file allocation table or FAT. A FAT is merely a singly-linked list, as demonstrated in Figure 25.

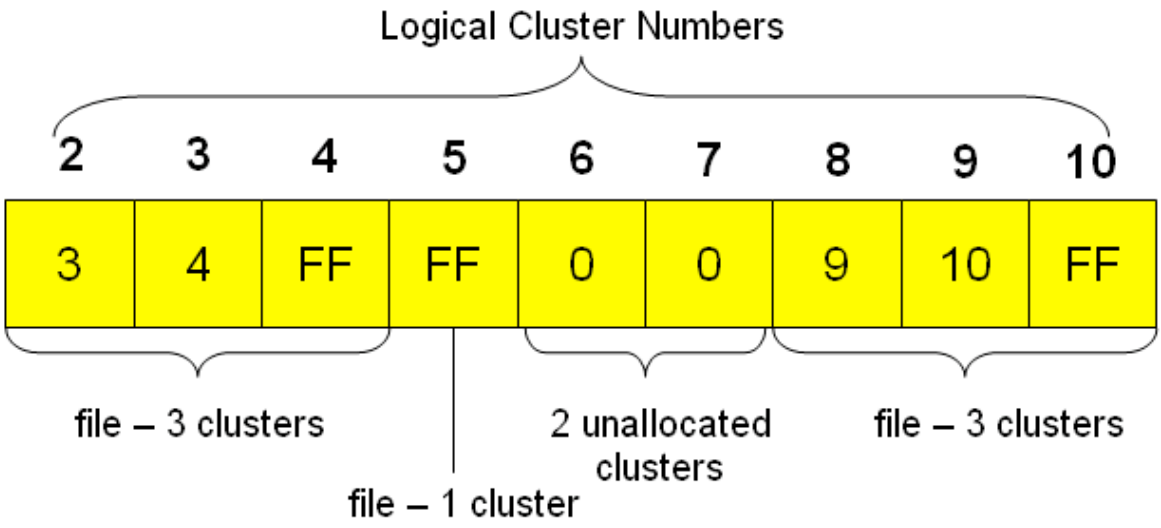

Figure 25. FAT Explanation

Figure 25 illustrates an abstract model of a file allocation table. The numbers above the boxes indicate the FAT entry for a particular *logical* cluster. The numbers in the boxes are pointers to the next cluster in the file. For instance, the FAT entry for cluster two contains a three, which is a pointer indicating that the next cluster in the file can be found at cluster three. FAT entry three contains a four indicating that the next cluster in the file is at logical cluster four. An FF in this instance is an end-of-file marker, indicating that this is the last cluster the file uses. If we assume that this is a FAT12 floppy disk, where each cluster is 512 bytes, we see that the file has a maximum size of 3 x 512bytes or 1,536 bytes. Because clusters two, three, and four are allocated to a file in the FAT, we know that the file resides in allocated space, and should appear in a logical analysis.

FAT cluster entry five has a single FF, indicating that the beginning and ending cluster for this file is logical cluster number five, and has a maximum size of 512 bytes. This file too is in allocated space, and should appear in a logical analysis.

FAT cluster entry six and seven are 0s. This has two possible interpretations. First, it could indicate that the two logical clusters have never been allocated, that is, they have never been used by a file. Second, it could mean that the file was deleted, and thus the clusters comprising the file were set for reallocation by placing 0s in the FAT entries. We can determine which interpretation is correct by viewing the logical cluster six and seven with a hex editor. If these clusters contain any data, we know that the files were deleted. We could also look at the root directory to see if there are any files whose starting cluster is logical cluster six. This would also be supporting evidence that the file was deleted.

Finally, FAT cluster numbers eight, nine, and ten indicate a small file comprising three clusters.

What happens when a file is deleted in a FAT file system?

- The first character of the file's name in the root directory is changed to e5h.
- The FAT entries are set to 0.

The clusters that compose the file are not touched. However, should the operating system decide that the FAT entries and corresponding clusters are needed, say, to save a new file,

then the clusters may be overwritten. That is the reason we freeze the computer: there is the potential for deleted files of possible evidentiary value to be overwritten by the operating system.

Although experiments we have conducted have varied in the exact details that occur, similar operations occur when files are deleted under most, if not all file systems, including NTFS, UNIX, and Linux file systems (EXT2, Reiser, etc.), i.e., a file pointer is modified to indicate the file is deleted, the clusters used by the file are marked as available, with the actual file contents remaining on the disk.

Unallocated Space Revisited

Revisiting Figure 23 above we see that there are two root directory entries whose names begin with an 'e5' (the hex section). These files are at offset 2600h (?EAL.JPG) and offset 2620h (?IDDEN.JPG). This is the character that the operating system places in the first position of the files name to indicate that the file has been marked as deleted. If we look at Figure 24 at offset 4200h (which happens to be logical cluster number two) we see that cluster still contains information located in unallocated space. This is the deleted file ?ATA.JPG. We can (and will) recover this file a little later in the chapter.

Slack Space

Slack space is an interesting phenomenon that can hold a great deal of useful evidence. Recall that the smallest unit that can be allocated to a file is a cluster. If a file is 1 byte in size, and the cluster size for a disk is 32,768 bytes (64 sectors per cluster), then the entire cluster will be reserved for the 1-byte file. Clearly this is a tremendous amount of wasted disk space. On average, the last cluster of each file will only be half full, meaning each file will waste half a cluster. Multiply the number of files on a hard drive by half the cluster size and you will see that there is a tremendous amount of wasted space on any hard drive. The larger the cluster size, the more wasted space.

Cluster size has an important implication for a forensic investigation: Larger cluster sizes mean more slack space. To illustrate slack space, Figure 26 shows a single cluster composed of 64 sectors. Say we create a file composed of 20 characters. Although the physical size of this file is small (160 bytes) it's logical size will be the size of the smallest allocatable unit, which is 32,768 bytes. This means that there are 32,608 bytes wasted (32,768 – 160) in this cluster. Those 32,608 bytes are *file slack*, i.e., slack space.

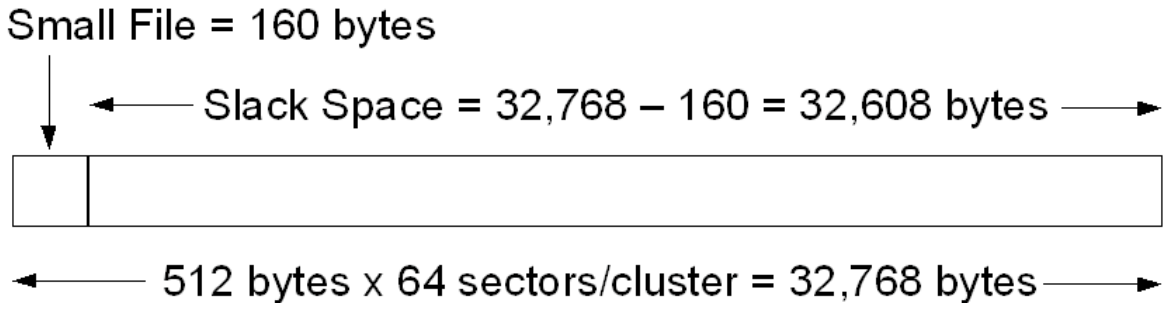

Figure 26. Slack space illustration

File slack becomes most interesting when clusters containing data are reused. Figure 27 illustrates this phenomenon. Say a criminal creates a document that discusses his plans to bomb a building, and then deletes the document. Later the criminal creates a grocery list, and the operating system happens to reuse part of the previously allocated clusters, as demonstrated in the bottom of Figure 27. Part of the original bombing plans document still exists in file slack. A logical analysis would *not* uncover the contents of slack space. The reason is that when we open a file only the contents of the file, up until the end-of-file marker, are retrieved by the operating system. We must conduct a physical analysis to find the contents of slack space contents.

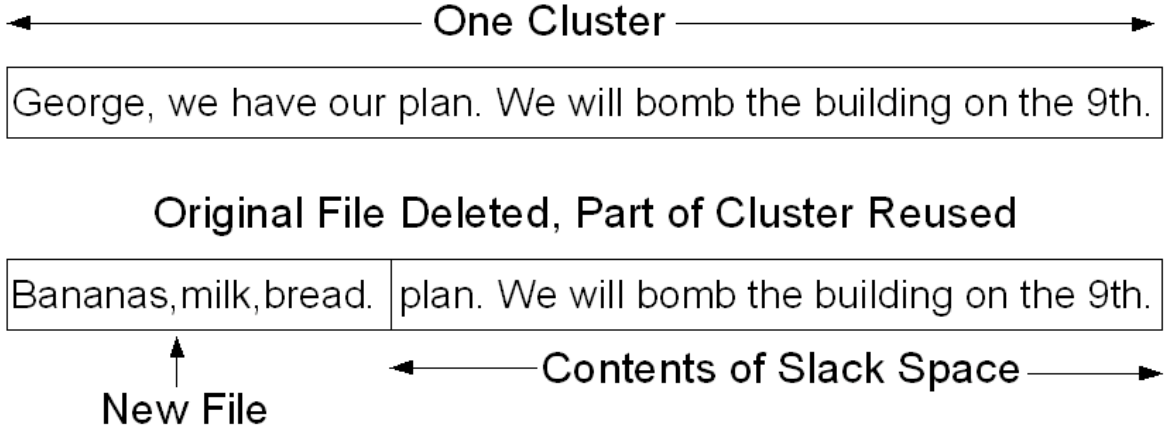

Figure 27. Cluster reuse and slack space

Recovering Deleted Files

As mentioned previously, the delete operation doesn't remove the contents of the file from the media. It only marks the entry in the file table as deleted, and the clusters previously used for the file are marked as available. All of the metadata in the root directory entry remain; including modified, accessed, and created times, attributes, file size, and so on. Also, the deleted file's FAT entries are marked as available by changing the FAT entries to 0. However, the data still remains on the disk. As long as the clusters are not overwritten, file contents can be recovered. And even if some of the clusters were overwritten, there is still the possibility that some of the file contents may still remain.

To demonstrate the ease with which a deleted file can be recovered, a small file was created, saved to a floppy, and then deleted. We created a forensic image of the floppy and then viewed the image with a hex viewer. Note that in this demonstration we do *not* mount the image, but rather simply open the image in a hex viewer.

To recover the file we need two pieces of information: the starting cluster of the file, and the size of the file. (It also helps if the file is not defragmented.) We can find both pieces of information in the root directory entry for the file.

```
0002600: e541 5420 2020 2020 4a50 4720 1872 7070  .AT     JPG .rpp
0002610: 9d2f 9d2f 0000 0c7e 532d 0200 7b7d 0000  ././...~S-..{}..
0002620: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0002630: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0002640: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0002650: 0000 0000 0000 0000 0000 0000 0000 0000  ................
lines 609-614
```

Figure 28. Root directory entry of deleted file

As shown in Figure 23 above, the root directory of a FAT12 formatted floppy disk begins at hex offset 2600h. Note that, except for the first character, we know the remaining characters of the file name. The starting cluster of the file is two, 0002h, as shown by the brackets on the left. The size of the deleted file is indicated by the brackets on the right, 7D7Bh.

(To calculate the starting cluster and file size we must perform a 'byteswap' operation to put the hex characters in their correct order. The byteswap for the starting cluster results in 0002h or 2 in decimal. The byteswap for the file size is 7D7Bh, or converted to decimal 32,123 bytes.)

Once we know the starting cluster and file size we can use `dd` to recover the file. Recall that the data area on a FAT formatted disk starts at logical cluster 2. This translates to physical sector 33 on a FAT12 formatted disk. (The boot sector, two file allocation tables and the root directory comprise the first 32 sectors.) The file size in hex is 7D7Bh, which is 32,123 bytes. Recall that the default block size for `dd` is 512 bytes and each cluster of a FAT12 disk is equal to one sector. To determine the number of clusters to extract we divide 32,123 by 512 = 62.74. Since a file can't use three-quarters of a cluster we round up to 63, which means we may also capture any contents of slack space, should it exist.

```
# dd if=image.dd of=_at.jpg skip=33 count=63
  63+0 records in
  63+0 records out
```

The `skip=` argument specifies at which *physical* sector to begin, here, physical sector 33. The `count=` argument specifies the number of blocks (sectors) to extract. The `dd` operation succeeds, indicating it both read and wrote 63 records (blocks). We can now open the '_at.jpg' file. As we can see from the recovered graphic in Figure 29 we can now complete the file name, the first letter was a 'c' for cat.jpg.

Figure 29: Deleted file recovered manually with `dd`

Dealing with Formatted Drives

If deleting a file doesn't remove all vestiges of a file from media, then surely formatting does, right? Not exactly. There are two types high-level formatting in Windows: A *quick*

format and a *full* format. A quick format performs two operations: a) it zeros out the root directory entries, and b) zeros out the file allocation table entries. The data area is *not* touched. (This is a nice simple experiment we encourage the reader to conduct with a floppy disk). A full format, in contrast, performs the same two operations as the quick format, and in addition it writes the hex character F6h in every sector of the data area. Thus, a disk that has been subjected to a full format will hold no recoverable data, except by experts using expensive procedures such as magnetic force microscopy (MFM).

Given that information let us reconsider our manual recovery of a deleted file. Can we recover the same file after a quick formatting of the disk? Recall that a quick formatting complete overwrites the root directory, therefore, we no long know the starting cluster nor file size of the file. We still have enough information available to us to recover files as long as we know the type of file we wish to recover.

Say we are asked to recover all of the graphical files from a hard drive that has been quick formatted. Given the file signatures (listed in Table 5 below), we can search for these file headers in the image. Once we find the headers, which always occur at the beginning of a cluster, use the following steps to recover the files (only if the files are not fragmented).

1. Search for the file signature(s) within the forensic image.
2. When a file signature is found, note the hex offset.
3. Convert the hex offset to physical sector number.
4. Use the `dd` command as above, using the physical sector number from step 3 above and a substantial `count` size.

File Type	Signature
JPG	`FFh D8h FFh`
BMP	`BM`
GIF	`GIF8[79]a`
PNG	`89h 50h 4Eh 47h`
TIFF	`49h 49h 2Ah 00h`

Table 5: Graphical file signatures

Given we are only guessing at the file size, what happens if we recover too many or too few clusters? Our experiences suggest that it should not hurt to recover more clusters than allocated to a file. Most of the time when we have recovered too little of the file, it is obvious. For instance, we conducted an experiment to the file recovery above. We used the commands (below) to recover too few (50) and too many (100) clusters, and then viewed the resulting files to determine the difference.

```
# dd if=image.dd of=small.jpg skip=33 count=50

# dd if=image.dd of=small.jpg skip=33 count=100
```

Figure 30 shows the results of recovering too few clusters. Note that part of the image is missing, the result of recovering too few clusters from the image. We can correct this by rerunning the command and increasing our count value to recover more clusters.

Recovering too many clusters resulted in the same image as shown in Figure 29 for recovering too many clusters. This is only a single example, and there may be a differences depending upon the type of file recovered.

Figure 30. Manual recovery of too few clusters

Behavioral Timelines: What Happened and When?

Sometimes investigators must create a time line of computer activity based upon file information and other available evidence to determine the sequence of activities occurring during a particular time frame. Examples of questions to be addressed through a timeline include:

- What files were changed? This can be answered thorough the creation of a time line based upon MAC times – modified, access, creation times – described later.
- How were the files changed? Deleting existing files, adding malicious code such as rootkits, or replacing old files with Trojaned versions can change the system. The latter is common, for example, with UNIX systems, where *ps* and *netstat* binaries are replaced with versions that will not report evidence of suspicious activity.
- Can the deleted files and/or other evidence be restored?

Every computer file on a Windows-based file system (FAT-based or NTFS) has associated with it three times: the time the file was created on the current volume (created or *ctime*); the time the file was last modified (modified or *mtime*); and the time the file was last accessed (accessed or *atime*). (Linux/UNIX file systems do not have a created time but a 'changed' time. Additionally, Linux EXT2 file system has a deleted time.) These times provide information regarding the events that occurred on a computer, allowing the investigator to create a scenario that explains a user's or intruder's activities.

From within Windows the MAC times can be accessed by right clicking on the file and selecting *properties* as demonstrated in Figures 31 and 32. A file residing on an NTFS volume in Figure 31 was copied to another volume formatted FAT32, Figure 32. Note the modified times are the same. The created times are different because the created times changed when a file is copied to a new volume. Also, FAT32 only keeps track of last date accessed date, not the time last access. (The reader should also note that we can determine that the volumes use different cluster sizes by looking at the "Size on disk" property.)

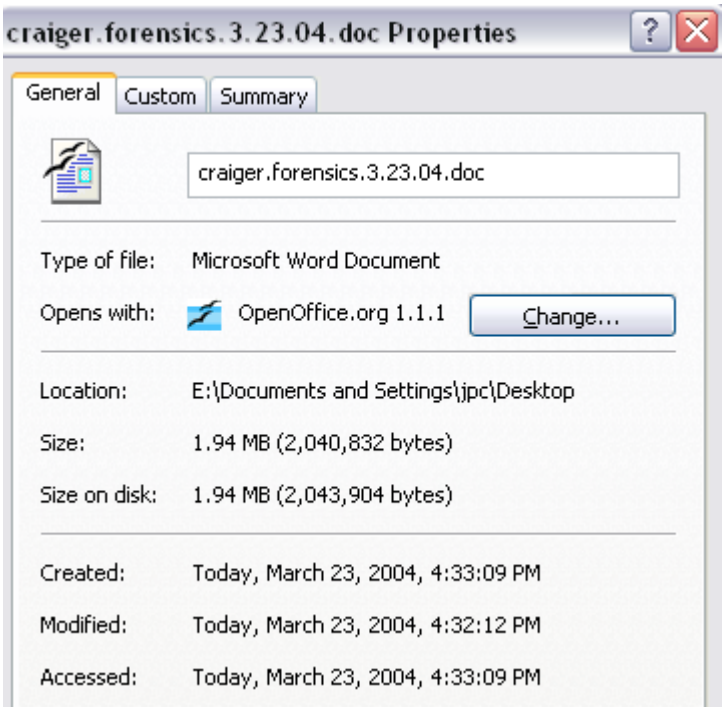

Figure 31: MAC times under NTFS

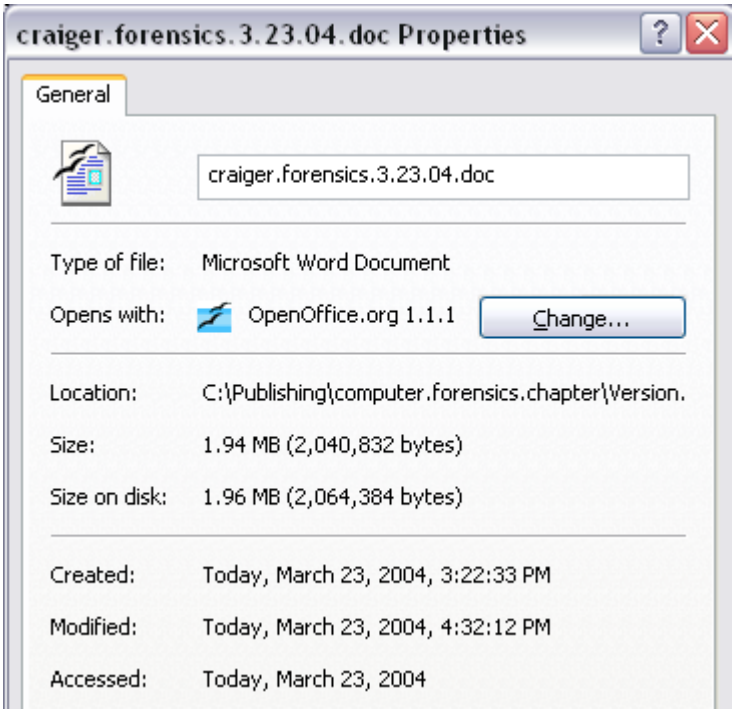

Figure 32. MAC times under FAT32

At the DOS command prompt, the `dir` command displays the files last modification time and date. To view the *created* times of all files in a directory, and to sort the file by date, use the command:

```
C:\> dir <directory name>/* /tc /od.
```

To view by *access* times and sort by date, use:

```
C:\> dir <directory name>/* /ta /od.
```

To view by *modification* times and sort by date, use:

```
C:\> dir <directory name>/* /tw /od.
```

The command's output is a listing of files within directory sorted by date. Unfortunately, this formatting does not allow one to easily determine the activities that occurred on the computer.

We have found the best way to create a timeline is to use tools from the open source forensic toolkit Sleuthkit (www.sleuthkit.org). To illustrate the use of MAC times line, say we wanted to know what files had been created on a system since October 9, 2003. We ran two tools from Sleuthkit, *mac-robber* and *mactime* against a running computer. We ran the command from a Linux box that was connected to the Windows system using a Samba share.

```
# mac-robber /mnt/fred/desktop/ > /evidence/fred.body

# mactime -b /evidence/whammo.body > /evidence/fred.mac
```

The *mac-robber* command extracts all of the time and date information from the files, and the *mactime* command then processes that information by sorting it by date and time and putting the information into a human readable timeline. Here is a small portion of the results:

[thousands of lines deleted for the sake of brevity]

```
Sun Nov 16 2003 11:42:52 530432 ..c /desktop/chapter/cforensics4.doc
Sun Nov 16 2003 11:44:08 530432 m.. /desktop/chapter/cforensics4.doc
Thu Nov 20 2003 17:45:37   4096 ..c /desktop/chapter
Fri Nov 21 2003 09:26:24 475136 ..c /desktop/chapter/cforensics5.doc
Fri Nov 21 2003 09:36:54 475136 m.. /desktop/chapter/cforensics5.doc
Fri Nov 21 2003 09:37:04 475136 ..c /desktop/chapter/cforensics6.doc
                         474624 ..c /desktop/chapter/~WRL0002.tmp
Fri Nov 21 2003 09:42:30 474624 m.. /desktop/chapter/~WRL0002.tmp
Sat Nov 22 2003 20:05:41  79653 m.c /desktop/vmware_drv.o
Sat Nov 22 2003 20:18:50  30158 ..c /desktop/linux_forensics.pdf
Sat Nov 22 2003 20:18:51  30158 m.. /desktop/linux_forensics.pdf
Sat Nov 22 2003 20:18:55  64687 ..c /desktop/SMART Forensics.pdf
Sat Nov 22 2003 20:18:58  64687 m.. /desktop/SMART Forensics.pdf
```

[Hundreds of lines deleted for the sake of brevity]

(Some of the information from the timeline, including the master file table number, file permissions, and links, was been deleted for brevity's sake).

After the date, time, and size (in kilobytes) fields comes a three character field that contains an indication as to whether the associated time is an *m-*, *a-*, or *c-time*, or combination thereof. Note that the MAC changes are organized by date and time. The first line displays the 4th version of this chapter (cforensics4.doc), and is a created time (note the '`..c`'). The next line is the modified time ('`m..`') for the same file indicating that

the file was last saved a couple of minutes after it was created. Note that the last access time for that file is not displayed because it was last accessed after November 22, and therefore doesn't fall within the timeline. Each file will have three times in the timeline.

From the timeline above we can derive several interesting facts. First, the file cforensics6.doc was created on Friday November 21 at 9:37AM. Note that at the same time the Word created a temporary file. Four minutes later, Word updated the modified time on the temporary file. Where is the modified and accessed time for cforensics6.doc? They come later in the timeline and are not displayed.

How are timelines used in computer forensics? MAC times can be used to verify or dispute a use's contention of whether the user created, modified, or accessed a file on a particular date and time. For example, if an employee's Temporary Internet Folders contained pornographic pictures, and the access times on these files coincided with the employee's work schedule, we have evidence that disputes the employee's contention that his Internet surfing habits do not include surfing for porn. (Of course, we have stronger evidence if the source computer system is running a secure version of Windows such as NT, 2000, or XP, which has separate personal directories for each user.)

Collecting Evidence from Live Systems

Thus far we have worked with a subject's computer, running Windows, which has been powered-down and disconnected from a network. In some circumstances it may be difficult or impossible to power-down and isolate a computer from a network. For instance, if a company's only e-commerce server was attacked, management may refuse to isolate the machine because might cost the company more in lost revenue than the attack (Mandia, Prosise & Pepe, 2003; Casey & Seglem, 2002). In this situation the investigator may be forced to work on a **live** system. This presents a problem because live systems are in a constant state of change, thus complicating the collection of evidence and investigation as a whole. Because the system is constantly changing we need to collect any evidence before it is changed, deleted, or overwritten. Not all evidence is subject to change in the same timeframe, however. Some evidence may be relatively stable, such as evidence on CDs or floppies, whereas other evidence may be ephemeral, such as the contents of RAM. These examples demonstrate that computer evidence may have different levels of **volatility**, which suggests that the investigator should prioritize evidence collection procedures by collection the most volatile information first.

Farmer & Venema (1999) proposed a volatility taxonomy, i.e., a measure of the likelihood of change to digital information on a running computer systems. From most to least volatile include:

- Process Register
- Virtual and physical memory
- Network state
- Running processes
- Disks, floppies, tapes
- CD-ROM, paper printouts

There is a correlation between the difficulty of collecting untainted evidence and its volatility. It is not impossible – as far we know – to collect the contents of registers

without changing them. In contrast, printed materials and CD-ROMS are fairly permanent and easy to collect without fear of contamination.

Farmer & Venema (1999) proposed an analogy between Heisenberg's uncertainty principle and the difficulty of working on live systems. Heisenberg's Uncertainty Principle states that attempting to measure both the location and momentum of an atomic particle affects the other; therefore, one can never produce an accurate measure of both at the same time. Similarly, attempts to collect evidence from a live system will change the contents of the system. This principle is demonstrated below.

(We are assuming the live-running server in our subsequent demonstrations is running some version of Linux.)

On Linux systems the file *kcore* (see Figure 34), located in under the */proc* directory, is a virtual file that maps to physical memory (RAM) of the system. The file can be examined using a debugger, or the `strings` command can be used to extract the human readable text from the file.

Similarly */dev/mem* is a logical file associated with physical memory, and */dev/kmem* is associated with kernel virtual memory (Kruse & Heiser, 2002). One may access the contents of physical and kernel memory through */dev/mem* and */dev/kmem*, respectively.

To illustrate how a simple procedure can change a running system we searched for the term 'Heisenberg Uncertainty Principle' within */proc/kcore*. (Note that it is highly unlikely that this term would have been in physical memory prior to searching for the term.) The output of the search (below) shows that the command we used to search for the term shows up, in various formats, several times, indicating that by attempting to collect the evidence we have changed the system.

```
simba:~ # strings /proc/kcore | grep 'Heisenberg Uncertainty Principle'

[everything from here down are the results of the search]

'Heisenberg Uncertainty Principle'
grep 'Heisenberg Uncertainty Principle'
grep 'Heisenberg Uncertainty Principle'
strings /proc/kcore | grep 'Heisenberg Uncertainty Principle'
strings /proc/kcore | grep 'Heisenberg Uncertainty Principle'
strings /proc/kcore | grep 'Heisenberg Uncertainty Principle'
strings /proc/kcore | grep 'Heisenberg Uncertainty Principle'
'Heisenberg Uncertainty Principle'
strings /proc/kcore | grep 'Heisenberg Uncertainty Principle'
simba:~ # strings /proc/kcore | grep 'Heisenberg Uncertainty Principle'
simba:~ # strings /proc/kcore | grep 'Heisenberg Uncertainty Principle
Heisenberg Uncertainty Principle
Heisenberg Uncertainty Principle
Heisenberg Uncertainty Principle
```

We found 14 occurrences of the term 'Heisenberg Uncertainty Principle' within physical memory. This small experiment underscores the susceptibility of contaminating a running computer system through even the simplest interaction. Thus care should be taken to minimize contamination when attempting to recover evidence from a live system. We can do this by having an incident response plan for dealing with live systems, including pre-

written scripts that can be run from a CD or floppy to minimize interacting with the system (e.g., because an typing error was made at the command line, requiring that the command be retyped, a common occurrence under stressful conditions).

Collecting Volatile Evidence

Time is of the essence when collecting evidence from a running computer system. As discussed above, volatile sources of evidence are purged after a brief period of time. Volatile and important sources of evidence on live systems, and the commands used to capture the evidence, include:

- Running processes (`ps` or the */proc* file system)
- Active network connections (`netstat`)
- ARP cache (`arp`)
- List of open files (`lsof`)
- Virtual and physical memory (*/dev/mem*, */dev/kmem*)

Gathering volatile data is more easily accomplished on file systems where everything is a file, which includes Linux, UNIX, and NTFS file systems. For example, all running processes on a Linux system are written out to disk the *proc* file system in the */proc* directory. Figure 33 is truncated example that illustrates the running processes on a Linux system (using the *ps aux* command), and the respective */proc* file system in Figure 34. For each process running in memory (identified by the numbers under the column labeled PID in Figure 33) there is a corresponding directory under the */proc* file system in Figure 34.

```
pc@simba:~> ps aux
USER       PID %CPU %MEM    VSZ  RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0    588  240 ?        S    Jun25   0:05 init [5]
root         2  0.0  0.0      0    0 ?        S    Jun25   0:00 [migration/0]
root         3  0.0  0.0      0    0 ?        SN   Jun25   0:05 [ksoftirqd/0]
root         4  0.0  0.0      0    0 ?        S    Jun25   0:00 [migration/1]
root         5  0.0  0.0      0    0 ?        SN   Jun25   0:03 [ksoftirqd/1]
root         6  0.0  0.0      0    0 ?        S<   Jun25   0:00 [events/0]
root         7  0.0  0.0      0    0 ?        S<   Jun25   0:00 [events/1]
root         8  0.0  0.0      0    0 ?        S<   Jun25   0:00 [kacpid]
root         9  0.0  0.0      0    0 ?        S<   Jun25   0:00 [kblockd/0]
root        10  0.0  0.0      0    0 ?        S<   Jun25   0:00 [kblockd/1]
root        11  0.0  0.0      0    0 ?        S    Jun25   0:00 [kirqd]
root        14  0.0  0.0      0    0 ?        S<   Jun25   0:00 [khelper]
root        15  0.0  0.0      0    0 ?        S    Jun25   0:00 [pdflush]
root        18  0.0  0.0      0    0 ?        S<   Jun25   0:00 [aio/0]
root        17  0.0  0.0      0    0 ?        S    Jun25   0:00 [kswapd0]
root        19  0.0  0.0      0    0 ?        S<   Jun25   0:00 [aio/1]
root       179  0.0  0.0      0    0 ?        S    Jun25   0:00 [kseriod]
root       239  0.0  0.0      0    0 ?        S    Jun25   0:00 [scsi_eh_0]
root       241  0.0  0.0      0    0 ?        S    Jun25   0:00 [scsi_eh_1]
root       246  0.0  0.0      0    0 ?        S    Jun25   0:00 [scsi_eh_2]
root       247  0.0  0.0      0    0 ?        S    Jun25   0:00 [ahc_dv_0]
```

Figure 33. Process list from running Linux system

```
pc@simba:/proc> ls
1       179     2780  6403  6746        bus           ioports     self
10      18      2785  6404  6751        cmdline       irq         slabinfo
11      19      3     6405  6756        config.gz     kallsyms    splash
11478   2       3300  6575  6758        cpufreq       kcore       stat
11566   2051    3626  6576  677         cpuinfo       kmsg        swaps
11613   21989   3714  6608  6774        crypto        loadavg     sys
11615   21990   3880  6669  6775        devices       locks       sysrq-trigge
11620   22057   4     6672  6776        diskstats     mdstat      sysvipc
11623   22059   4930  6674  6778        dma           meminfo     tty
11700   22091   5     6677  7           driver        misc        uptime
14      22093   5503  6716  7301        execdomains   mm          version
15      239     6     6736  8           fb            modules     vmstat
17      241     6087  6737  8865        filesystems   mounts
1718    246     6088  6739  9           fs            mtrr
1719    247     6400  6740  acpi        ide           net
```

Figure 34. Associated /proc file system

The */sbin/arp -v* command displays the contents of the ARP (address resolution protocol) cache. The example in Figure 35 illustrates that the ARP cache on this computer has two MAC addresses under the label titled **HWaddress**. The ARP cache holds MAC addresses (media access control addresses, i.e., the hardware addresses of the network interface cards, not to be confused with MAC times) of computers on the same subnet that have been recently communicating with the computer under investigation. These addresses are purged every so often, thus it is important to gather this information quickly.

```
simba:~ # arp -v
Address                  HWtype  HWaddress           Flags Mask            Iface
whammo.om.cox.net        ether   00:04:75:8B:06:BA   C                     eth0
192.168.0.1              ether   00:09:5B:9F:21:16   C                     eth0
```

Figure 35. ARP output

The netstat command displays network connections and listening ports. Figure 36 displays a portion of the results of running netstat. Note there several established connections (under the "State" heading). The value "LISTEN" under the State heading indicates whether a port is open. Here we see we have three open TCP ports: port 22 (secure shell, SSH), *netbios* (139) and *CIFS* (445), the latter two of which are used with Samba (a service that supports connections between my Linux and Windows machines).

```
simba:~ # netstat -tupan
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:139             0.0.0.0:*               LISTEN      11478/smbd
tcp        0      0 0.0.0.0:445             0.0.0.0:*               LISTEN      11478/smbd
tcp        0      0 192.168.0.3:1733        66.35.250.67:80         TIME_WAIT   -
tcp        0      0 192.168.0.3:1741        66.35.250.67:80         TIME_WAIT   -
tcp        0      0 192.168.0.3:1745        193.136.138.47:80       ESTABLISHED 11620/mozilla-bin
tcp        0      0 192.168.0.3:1746        193.136.138.47:80       ESTABLISHED 11620/mozilla-bin
tcp        0      0 192.168.0.3:1732        66.35.250.62:80         TIME_WAIT   -
tcp        0      0 192.168.0.3:1718        192.168.0.2:445         ESTABLISHED -
tcp        0      0 192.168.0.3:1729        64.233.167.99:80        ESTABLISHED 11620/mozilla-bin
tcp        0      0 192.168.0.3:1736        64.233.167.99:80        ESTABLISHED 11620/mozilla-bin
tcp        0      0 :::22                   :::*                    LISTEN      3626/sshd
udp        0      0 192.168.0.3:137         0.0.0.0:*                           11566/nmbd
udp        0      0 0.0.0.0:137             0.0.0.0:*                           11566/nmbd
udp        0      0 192.168.0.3:138         0.0.0.0:*                           11566/nmbd
udp        0      0 0.0.0.0:138             0.0.0.0:*                           11566/nmbd
simba:~ #
```

Figure 36. View network connections and open ports with *netstat*

Log Files as Digital Evidence

Log files can be very important sources of forensic evidence. A server's log files will contain information about various system resources, processes, and user activities. Protocol analyzers, sniffers, SMTP, DHCP, FTP, and WWW servers, routers, firewalls, and almost any system- or user-driven activity can be collected in a log file. However, if the systems administrator has not enabled logging, then the evidence necessary to associate an intruder with an incident may not exist. Unfortunately, knowledgeable intruders and criminals know this, and one of the first orders of business is to destroy or alter log files to hide their activities.

A second important piece of information, and one sometimes overlooked, involves the system clock. Anything logged to a file has an associated time and date stamp. Time and date stamps enable the investigator to determine the sequence of events that transpired. System clocks, unless explicitly corrected on an occasional basis, can be off anywhere from several seconds to hours. This can cause problems because any correlations between log files from different computers whose system clocks are different make it difficult or impossible to correlate events. A simple solution is to automatically synchronize clocks by having all systems run a daemon, such as the UNIX *ntpd* daemon, to occasionally synchronizes the system time and date with one a government-sponsored (e.g., NIST, National Institute of Standards and Technology) atomic clock. This is transparent to the user and takes little system resources.

Reducing the Potential for Evidence Contamination

If the computer system is on, files will be changing. If the computer system is connected to a network, files will be changing. If we interact with the system, files will change. This is a large, probably unsolvable problem: the need to interact with a live system and gather evidence will cause some form of contamination. The best that can be done then is to limit contamination by limiting interactions with the system. This can be done through planning prior to the incident. It is important, for example, to have an incident response team and an incident response plan that can be executed once an incident has been identified. One aspect of this plan is a pre-defined set of command scripts that can be executed to collect evidence from a running system. Ideally, these scripts should be run to limit the number of errors we are all susceptible to. (FIRE, Forensic Incident Response Environment is a Linux-based bootable CD that includes scripts to recover system information for both Linux and Windows systems: fire.dmzs.com. It is highly recommended).

Here is a simple example of running a predefined script to gather system information and transport it offsite via a network connection.

```
# (ps aux; netstat -tupan; cat /var/log/message) | nc 192.168.1.1 4444
```

These commands collect the running processes (`ps aux`), list of open network connections (`netstat -tupan`), copies the log file *messages* (`cat /var/log/messages`) and uses `netcat` to send them over a network connection to the local forensic machine. The number of commands that could be included within the

processes is unlimited. This form of evidence collection is desirable because the script employed to capture the evidence can be preplanned and tested prior to its use in any real incident. Make sure to test it on various forms of UNIX and Linux as some versions differ just enough that your commands may not work they way they are expected. For instance, on BSD-style UNIX, `ps ef` is the command line equivalent of `ps aux` for capturing running system processes.

Commercial Tools

This chapter would not be complete without a brief mention some of the commercial tools available as of 2004. Unfortunately space limitations guarantee that a discussion of the commercial forensics tools will be incomplete. Realize that this is not an advertisement for any particular tool, but rather a pointer to existing tools that may warrant further examination by a serious investigator.

There are several Windows-based forensics tools that are capable of performing all of the procedures we have covered in this chapter, and many we haven't covered. The two tools we discuss here are Guidance Software's EnCase (Forensic or Enterprise Editions: www.guidancesoftware.com) and Accessdata's Forensic Toolkit (part of the Ultimate Toolkit: www.accessdata.com). We have had formal training and a good deal of experience with both EnCase and Forensic Toolkit; each has their strengths and weaknesses. Ideally an investigator should be trained and have access to several tools. The reason is that it is often desirable to verify the findings from one tool with a different tool to ensure the validity and integrity of the findings.

Forensic Toolkit and EnCase support: imaging; reading multiple file systems; reading multiple image formats; file viewing; advanced string searches; graphical/gallery views; email analysis; compressed file analysis; known file filters/hash analysis; bad file extension determination, electronic discovery; and numerous other capabilities. We have included screenshots of EnCase (Figure 37) and Forensic Toolkit (Figure 38), displaying the results of a string search, to illustrate their graphical interface.

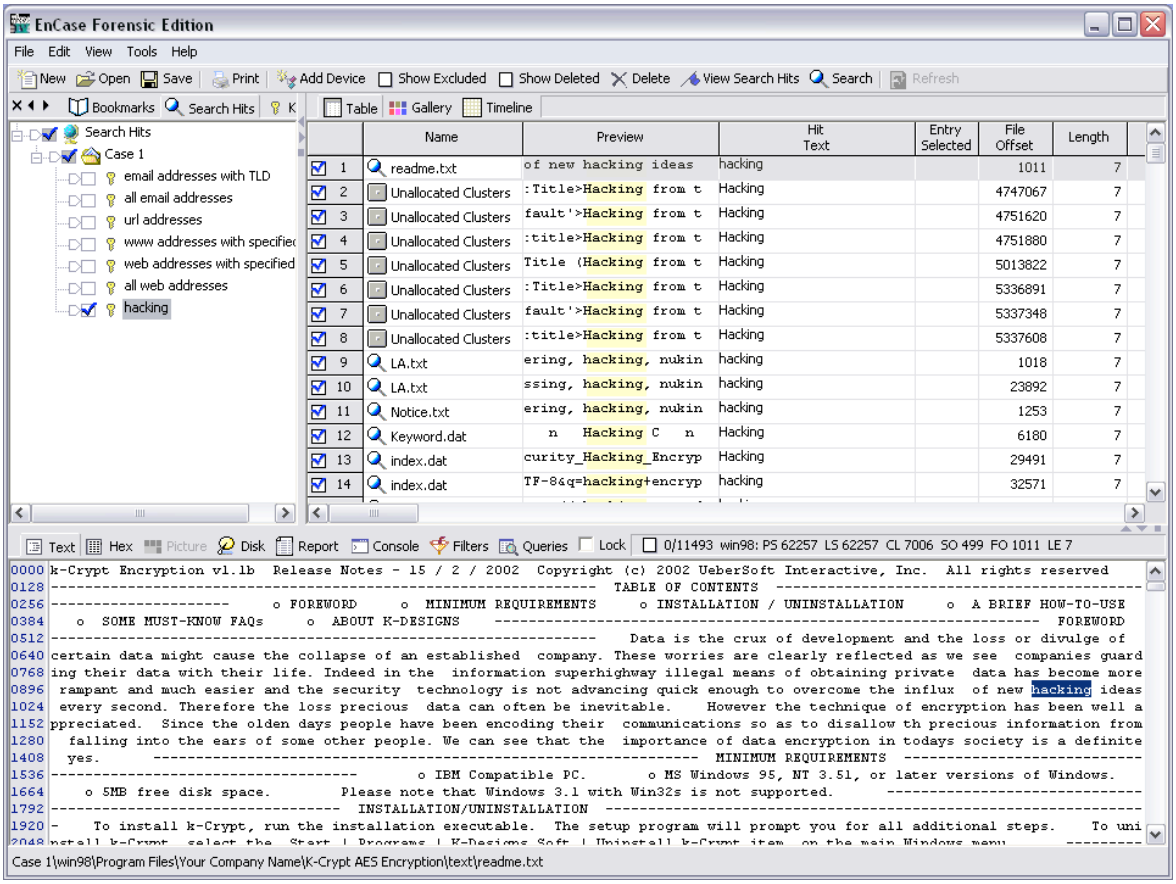

Figure 37. Guidance Software's EnCase Interface

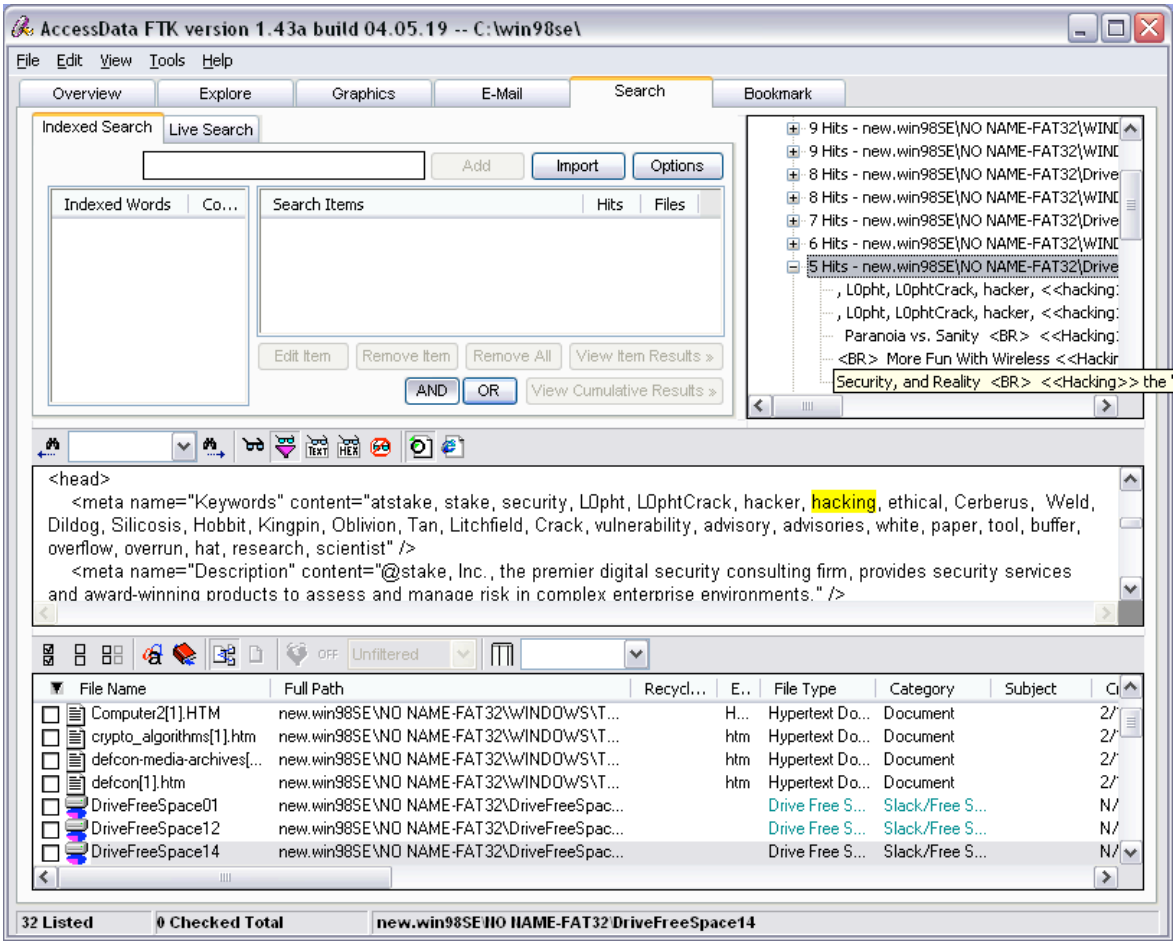

Figure 38: Accessdata's Forensic Toolkit Interface

Other commercial tools available as of 2004, in alphabetical order, include:

- ARS Data's SMART (runs under Linux): http://www.asrdata.com/tools/
- ILook Investigator (law enforcement only): http://www.ilook-forensics.org/
- Maresware Forensic Tools: http://www.dmares.com/maresware
- New Technologies Forensic Suite: http://www.forensics-intl.com/tools.html
- Paraben Forensic Toolks: http://www.paraben-forensics.com/

For the most up-to-date information on the availability of commercial and open-source tools we suggest doing a Google search for 'computer forensic tool.'

Conclusion

This chapter provided a technical introduction and overview of computer forensic procedures. We attempted to cover the fundamental aspects of computer forensics methods and procedures, from acquiring and verifying the evidence through a complete logical and/or physical analysis. Our demonstrations were designed to illustrate fundamental concepts rather than how a particular commercial tool could be used.

Technology changes at an increasing pace, which creates several problems for investigators. For example, devices that may hold evidence have become more diverse, witness cell phones, personal digital assistants (PDAs) such as the Palm® handhelds and

Compaq IPAQ®, Blackberry® wireless email devices, compact flash and smart media, and so on. Investigators must have the necessary hardware and software to make a forensic image and analyze the information obtained from these diverse devices.

Investigators are likely to encounter new types of media on a continual basis. Therefore it is important that investigators be aware of these types of media, including any unique properties that may be important in understanding for the acquisition process. Space limitations prevented us from describing the means of handling the more exotic types of evidence. Nevertheless, the investigator must also have the necessary knowledge, techniques, and tools available to make the forensic images as well as perform thorough logical and physical analysis. The best source for best practices on handling different types of media is the *U.S. Secret Service's Best Practices Guide for Seizing Electronic Evidence* (http://www.secretservice.gov/electronic_evidence.shtml).

Finally, storage technology is becoming exponentially larger and therefore more difficult and time consuming for investigators. For example, it is not uncommon to encounter personal home computers with hard disks in the 200+ gigabyte range. Moreover, terabyte-sized disk arrays are becoming more commonplace. Numerous law enforcement and investigators with whom we have spoken have encountered such devices. This will create problems as the acquired images of a criminal investigation may outstrip an investigators ability to hold and preserve the evidence. Fortunately, technologies such as storage area networks, as they become less costly, may allow law enforcement and industry incident response teams to better deal with this problem.

Glossary

Allocated space – are the clusters allocated to a file and which are tracked by the file system.

Allocation unit – the smallest unit of disk space that may be allocated to a file. Varies by file system.

Bit-stream copy – a bit-for-bit copy of digital evidence.

Block – UNIX terminology for an allocation unit (see allocation unit).

Cluster – Microsoft windows term for allocation unit (see allocation unit).

FAT – Acronym for File Allocation Table. A common form of file system used with Microsoft Windows operating systems.

File allocation table – Part of a FAT file system. A singly-linked list of pointers to clusters comprising a file.

Forensic image – An exact, bit-for-bit copy of media.

Hash – Also known as a message digest, cryptographic hash, or one-way hash. A hash is a hex value, typically 128- or 160-bit, that is unique to the contents of a file.

Hash set – A list of hash values for a set of files.

Known files – Files known to be of no evidentiary value that can be discarded from an analysis. Usually identified through a hash analysis.

MD5 hash – A 128-bit cryptographic hash algorithm created Ron Rivest of MIT.

Notable files – Files of known evidentiary value usually identified through a hash analysis.

NTFS – Microsoft New Technology File System.

Metadata –A file's metadata consists of all information about the file excluding its contents: File name, size, MAC times, starting cluster, permissions, attributes, etc.

Root directory – File table under FAT systems that holds file metadata.

Sector – Hardware unit of measure on a disk, typically 512 bytes. Multiple sectors make up an allocation unit. Individual sections of a disk track.

Slack space – Disk space left over between the end of the data and the end of the last cluster of a file. Slack space may contain residual information.

Unallocated space - the clusters not in use by a file. Where deleted files reside.

Volume – Commonly, another name for a partition on a disk. A hard drive may have up to four primary volumes or partitions.

Wipe – e.g., Forensic wipe. To remove vestiges of information from media by writing a series of characters over the information.

Write-block – a physical device that allows data to be read from a hard drive, but prevents data from being written to it. Typically blocks interrupt 13h.

References

Bigelow, S. (2004). *Troubleshooting, maintaining & repairing PCs*. San Francisco: McGraw-Hill.

Carrier, B. (2002b). *Sleuthkit*. www.sleuthkit.org

Casey, E. (2001). *Digital Evidence and Computer Crime*. Academic Press.

Casey, E. (2002). *Handbook of Computer Crime Investigation: Forensic Tools and Technology*. Academic Press.

Craiger, J.P. (May 2004). *Linux: Portable forensics Toolkit*. Presentation accepted for the 26th Annual Department of Energy Computer Security Training Conference. St. Louis, MO.

Craiger, J.P. & Nicole, A. (Sept, 2002). *An applied course in network forensics*. Presentation for the Workshop for Dependable and Secure Systems. University of Idaho, Moscow, Idaho, Sept 23-35.

Craiger, J.P., & Pollitt, M. (to appear). Computer forensics and law enforcement. In H. Bigdoli (Ed.), *Handbook of Information Security*. John Wiley & Sons.

Dartmouth Institute for Security Technology Studies. (2002). *Law Enforcement Tools And Technologies For Investigating Cyber Attacks: A National Needs Assessment*. Dartmouth College.

Department of Energy. (2003). *First Responder's Guide*. Department of Energy Computer Forensic Laboratory.

Dittrich, D. (2001). *Basic Steps in Forensic Analysis of Unix Systems*. http://staff.washington.edu/dittrich/misc/forensics

Farmer, D., & Venema, W. (Sept., 2000). Forensic computer analysis: An introduction. *Dr. Dobb's Journal.*

Brezinski, D., & Killalea, T. (February, 2002). RFC 3227: Guidelines for Evidence Collection and Archiving. http://rfc3227.x42.com/. Last visited: January 24, 2004.

Farmer, D. (Jan., 2001). Bring out your dead: The ins and outs of data recovery. *Dr. Dobb's Journal*.

Farmer, D., & Venema, W. (April, 2001). Being prepared for intrusion. *Dr. Dobb's Journal*.

Farmer, D. (Oct., 2000). What are MACtimes? *Dr. Dobb's Journal*.

Grance, T., Kent, K., Kim, B. (2004). *National Institute of Standards and Technology Computer Security Incident Handling Guide*. NIST: Gaithersburg, MD.

Heverly, R. & Wright, M. (2004). Cyberspace law and computer forensics. In S. Bosworth and M.E. Kabay (Eds.), *Computer Security Handbook*. New York: Wiley.

Jones, K. J. (2003a). *Forensic Analysis of Internet Explorer Activity Files*. www.foundstone.com.

Jones, K. J. (2003b). *Forensic Analysis of Microsoft Inter Explorer Cookie Files*. www.foundstone.com.

Jones, K. J. (2003c). *Forensic Analysis of Microsoft Windows Recycle Bin Records*. www.foundstone.com.

Jones, K. J., Shema, M., & Johnson, B.C. (2002). *Anti-hacker Toolkit*. San Francisco: Osborne.

Kruse, W.G. III, & Heiser, J.G. (2001). *Computer Forensics: Incident Response Essentials.* Addison-Wesley.

Larson, T. (2002). The other side of civil discovery. In E. Casey (Ed.), *Handbook of Computer Crime Investigation*. Academic Press.

McNamara, J. (2003). *Secrets of computer espionage*. New York: Wiley.

Mohay, G., Anderson, A., Collie, B., De Vel, O., & McKemmish, R. (2003). *Computer and Intrusion Forensics*. Norwood, MA: Artech.

Morris, J. (Janurary 28, 2003). *Forensics on the Windows Platform, Part One*. www.securityfocus.com/printable/infocus/1661

Morris, J. (February 11, 2003). *Forensics on the Windows Platform, Part Two*. www.securityfocus.com/printable/infocus/1665

Mueller, S. (2003). *Upgrading and repairing PCs.* New York: Que.

Nelson, B., Phillips, A., Enfinger, F., & Steuart, C. (2004). *Guide to Computer Forensics And Investigations*. Boston: Thomson.

Parker, D. (1998). *Fighting Computer Crime*. New York: Wiley.

Prosise, K, Mandia, K., & Pepe, M. (2003). *Incident Response: Investigating Computer Crime*. San Francisco: McGraw-Hill.

Rivest, R. (1992). The MD5 Message-Digest Algorithm. http://theory.lcs.mit.edu/~rivest/rfc1321.txt. Last visited 12/23/03.

Sammes, T., & Jenkinson, B. (2000). *Forensic Computing: A Practitioner's Guide*. London: Springer Verlag.

Schulz, E.E., & Shumway, R. (2002). *Incident Response: A Strategic Guide to Handling System and Network Security Breaches*. New Riders.

Seglem, K. K. (2002). Introduction to Digital Evidence Reconstruction using UNIX Systems. In E. Casey (Ed.), *Handbook of Computer Crime Investigation*. Academic Press.

Sheldon, B. (2002). The forensic analysis of Window's systems. In E. Casey (Ed.), *Handbook of Computer Crime Investigation*. Academic Press.

Stephenson, P. (2000). *Investigating Computer-Related Crime*. Boca Raton, FL: CRC Press.

U.S. Department of Justice. (July, 2002.) *Searching and Seizing Computers and Obtaining Electronic Evidence in Criminal Investigations.* Computer Crime and Intellectual Property Section. Criminal Division, United States Department of Justice. http://www.usdoj.gov/criminal/cybercrime/searching.html. Last visited January 14, 2004.

U.S. Secret Service. (2002). Best Practices Guide to Seizing Electronic Evidence, Version 2. http://www.cio.com/securitytools/BPGv2.pdf Last visited: December 29, 2003.

Open-source forensics software. http://www.opensourceforensics.org/tools/unix.html. Last visited Jan 7, 2004.

Author. Location of Outlook Express Files Under Windows XP. http://www.attention-to-details.com/newslog/379-location-outlook-express-files-on.asp#a194

Further Reading

Caloyannides, M.A. (2002). *Computer Forensics and Privacy.* Artech House Computer Security Series.

Caloyannides, M.A. (2003). *Desktop Witness.* Artech House Computer Security Series.

Carrier, B. (2002a). *Autopsy forensic browser*. www.sleuthkit.org. Last visited January 1, 2004.

Carvey, H. (September 5, 2002). Win2K First Responder's Guide. http://www.securityfocus.com/infocus/1624

Casey, E., Larson, T., & Long, T.M. (2002). Network analysis. In E. Casey (Ed.)., *Handbook of Computer Crime Investigation*. Academic Press.

Cheng, D. (Nov. 1, 2001). Freeware forensics tools for UNIX. http://online.securityfocus.com/infocus/1503

Cooper, M., Northcutt, S., & Frederick, K. (2002). *Intrusion Signatures and Analysis.* New Riders.

Department of Energy. (2002). *First Responders Guide*. Department of Energy Computer Forensic Laboratory.

Furnell, S. (2002). *Cybercrime: Vandalizing the Information Society*. Upper Saddle River, NJ: Prentice-Hall.

Grundy, B.J. (2002). *The Law Enforcement Introduction to Linux: A Beginner's Guide*. http://ohiohtcia.org. Last visited 1/24/04.

Hardy, K., & Kreston, S. (2001). *Using Analogy to Explain Computer Forensics: Techniques used to explain computer jargon to courtroom juries*. National District Attorney's Association. www.ndaa.org. Last visited 12/28/03.

Lucas, J., & Moeller, B. (2004). *The Effective Incident Response Team*. Boston, MA: Addison-Wesley.

Marcella, A. J. Jr., & Greenfield, R.S. (2002). *Cyber Forensics: A Field Manual for Collecting, Examining, and Preserving Evidence of Computer Crimes.*

Nemeth, E., Snyder, G., Hein, T.R. (2003). *Linux administration handbook*. Upper Saddle River, NJ: Prentice Hall.

Northcutt, S., & Novak, J. (2001). *Network Intrusion Detection: An Analysts Handbook*. New Riders.

Parker, D. (1998). *Fighting Computer Crime*. New York: Wiley.

Spitzner, L. (2001). *Know Your Enemy: Revealing the Security Tools, Tactics, and Motives of the Blackhat Community*. Honeynet Project.

Stoll, C. (1988). *Cuckoo's Egg: Tracking a Spy through the Maze of Computer Espionage.* New York: Pocket Books.

Tan, J. (2001). *Forensic Readiness*. @stake Research. www.@stake.com.

Vacca, J.R., & Erbschloef, M. (2001). *Computer Forensics: Computer Crime Scene Investigation*. Charles River Press.