# Computer Forensics

**Created by: Peter A. H. Peterson and Dr. Peter Reiher, UCLA {pahp, reiher}@cs.ucla.edu**

**Contents**

## Overview

The purpose of this lab is to introduce you to *basic* forensic analysis theory and practice. You will have an opportunity to investigate disk images of three potentially compromised computers in order to determine what happened to them, potentially recover data, and investigate who might have compromised them. You will perform the analysis yourself, and write brief memos summarizing your findings.

Since practically the work in this lab involves command-line tools, you should be familiar with the Unix shell environment and common shell utilities such as find, grep, strings, etc.

> This exercise will not qualify you to perform Computer Forensic Science in the real world! While the material in this lab is as accurate and useful as possible, Computer Forensic Science in the context of the law and law enforcement has special requirements and considerations that we are not able to address in this exercise. Always make sure that any system analysis you perform is legal.

After successfully completing this lab, you will:

1. be familiar with basic computer forensic theory, including:
    1. Identification methods
    2. Common types of tools
    3. Common locations of data
    4. Data integrity protocols
2. know how to use basic system analysis tools such as:
    1. dd and mount

    2. e2undel
    3. chkrootkit
    4. system logs
    5. strings
    6. hex editors
    7. find and locate
    8. grep
3. have analyzed three computer disk images in order to determine:
    1. the cause of the system's failure. (Was it compromised? If so, how? What happened?)
    2. the identity of the intruder (if one exists).
    3. How the problem could have been prevented.
    4. What needs to be done before returning the system to production. (Is the computer safe to use?)
4. have written up a 1-2 page analysis report on each disk image detailing the information discovered and any recommendations.

## Required Reading

### Computer Forensic Science

Forensic Science (FS) is a cross-discipline science primarily devoted to answering questions in a court of law, and typically involves the use of specialized knowledge to glean obscure but meaningful evidence useful in determining what happened in a partially unknown sequence of events. Popular television shows such as the Crime Scene Investigation (CSI) franchise capture the gist of forensics (albeit in a highly sensationalized manner) and have captivated audiences and citizens excited by illuminating new sources of evidence in criminal investigations. Computer Forensics (CF) is the application of Forensic Science to answer questions in the field of computer systems.

Computer Forensic Science (CFS) has grown tremendously as society becomes increasingly reliant on computer technology. In the past, a clever crime might have required the careful use of fingerprinting powder, tape, a plastic bag, and a magnifying glass to gather evidence, while today many crimes happen in exclusively in cyberspace where such techniques are not directly applicable. Still, the same kinds of questions -- "Who was here?" "What did they leave behind?" -- are every bit as relevant in cyberspace as they are in the real world. In fact while many techniques, such as fingerprinting, are not literally meaningful in cyberspace, most have direct analogues in Computer Forensics and are used to answer the same kinds of questions in court. Like traditional forensics, Computer Forensic scientists must also follow evidence gathering rules and protocols in order to ensure that evidence is gathered legally and will be admissible in a court of law.

Practically speaking, many of the techniques common to Computer Forensics are part of the skill set that good computer scientists, programmers, and system administrators use day to day in order to conclusively determine the facts about any complex and obscure system. When a program crashes, a computer stops responding, or a system is compromised, a sysadmin puts on a "Computer Forensics hat" in order to determine what really happened. In this way, "computer forensics deals with the science of determining computer-related conduct - the who, what, when, where, and how of computer and technology use [1]" whether it is used in a court of law or in the course of a day at work.

There are a number of direct analogues between the methods of traditional Forensic Science and Computer Forensic Science. For example, just as fingerprint or DNA testing can be used to identify suspects, network addresses and encryption keys can be used to identify cyber-criminals. Similarly, just as there are evidence gathering rules for traditional forensics, there are rules guiding computer forensics. An obvious example is that you need a proper warrant or other authorization before investigating private property including computer systems. A less obvious example is that of the integrity of evidence; a police investigator will use plastic gloves and a face mask to ensure that the evidence isn't tainted in the gathering process, and will put the evidence into a labeled and sealed plastic bag. A computer forensics expert must use encryption, documentation, and other safeguards to ensure that the data is not tainted or destroyed.

Finally, while CFS has many techniques used to establish the same kinds of facts, due to the nature of digital systems, very few methods are 100% reliable. While it is impossible at this time to fabricate DNA evidence, a clever criminal could plant stolen DNA at a crime scene. Similarly, a clever computer criminal can forge identifying information with the difficulty of detecting the forgery increasing with the expertise of the forger. A perfect forgery can be impossible to detect. Therefore, it is important to gather as much information as possible and cross-reference it in order to show that the evidence gathered points to a meaningful conclusion.

## Positive Identification : Fingerprints and DNA

In traditional forensics, things such as fingerprinting, DNA testing, security cameras, and eye witnesses are used to positively establish that a suspect was involved (or was *not* involved) in a crime. In CFS, various methods are used to attempt to establish the same kinds of facts.

### Username

Most computer systems today rely on the concept of the user, which is an access account typically protected by a username/password combination, or in some cases, an encryption key. Information on a server is typically logged by user. When a user account is tracked breaking the law (or a policy), it is almost certain that the person who controls the user account is the person responsible. Unfortunately, sometimes login credentials are sometimes stolen, lost, or given away, in which case multiple individuals have access to the same digital "identity". Additionally, usernames can be forged in log files and other records with the appropriate credentials.

### Network Addresses

Every computer on a network has an address, and most data packets on a network (including the Internet) have source and destination addresses encoded into the packet. These addresses can be used to establish where data originated and also where it might be going. Unfortunately, due to the decentralized nature of the Internet, sometimes these source and destination addresses only indicate the next "hop" -- the next node -- which could be the final destination, or might be just another stop along the way. Additionally, the IP address space is divided into several regions, some of which indicate Internet traffic, and some of which indicate internal LAN traffic. Some IP ranges are reserved or not used. When investigating an IP address, one must first identify the kind of address it is.

Unfortunately, source and destination addresses can be forged. For example, a typical DDoS attack uses bogus source addresses on its packets so that when any hosts along the way attempt to respond to the packets, they are misdirected to "innocent" network nodes. Likewise, a user executing a remote exploit might send packets with bogus source addresses to mask the location from which the attack originates. Finally, attackers often log into many servers in sequence, attacking from the last server in the chain. For example, an attacker might log into a compromised server at Yahoo, then from Yahoo into one at AOL, then a personal web server, and finally a compromised server in Korea.

This makes the attack appear to originate from Korea and masks the origin of the attacker, because recovering that information requires logs from all the servers in the chain. This is impossible in practice due to international law and other factors.

Finally, sometimes addresses and identifiable markers are located at a lower level. For example, all Ethernet networking cards include a MAC (Medium Access Control) address which is unique and contains information such as the manufacturer's name. These can often be changed in software, but they are a good introduction to more subtle forms of identification.

### CPU Serial Numbers

In 1999, Intel decided to include a unique serial number into each of it's Pentium III processors. This "Personal Serial Number" (PSN) could be accessed with a special opcode. While the "feature" was supposedly able to be disabled, demonstrations showed that it could still be accessed after having been disabled. An uproar from privacy advocates and European government led to Intel dropping the PSN in 2000. While only a small portion of CPUs ever had PSNs (and still fewer of those are still in use today), the PSN was one of the first "features" that allowed individuals to track and identify individual computer systems.

### Hardware/Software Artifiacts

Sometimes a computer has a more esoteric fingerprint made up of its software configuration, consistent software errors, hardware bugs, or other observable, identifying behaviors. For example, a network card may improperly calculate checksums used to mark packets, or may have an improbable MAC address. Perhaps an exploit package they use has customized markers in it, such as the attacker's pseudonym or hacker's group. These are harder to hide and to forge because they require intimate knowledge of a computer system and can help show a link between data and the source of the data. This kind of sleuthing is analogous to analyzing footprints at a crime scene. If the footprints were made with a limp, investigators will look for suspects with a limp.

### Software Watermarks and GUID

Some software packages incorporate unique software serial numbers into files created with that software. The most common example of this is Microsoft Office, which has been incorporating GUIDs (Globally Unique IDentifier) into files created by Office since 1999. Every installation of MS Office has a unique GUID, so files created with Office can be traced back to that computer (even without CPU serial numbers!). Additionally, early versions of the GUID algorithm used the computer's MAC address as one of the foundations of the GUID; this meant that the GUID linked not only back to a particular installation of Office, but also to a particular physical computer. (A GUID was used to track the author of the Melissa worm in 1999.) Because GUIDs are not easily changed or readily apparent, they often become useful evidence.

### Encryption Keys

Public Key Cryptography and encryption in general is a double-edged sword. On the one hand, strong encryption is a prerequisite for secure, confidential communications. On the other hand, sending and receiving data with the use of a private key ties that information to the individual who holds the private key, since it is considered to be a unique, private cryptography key. For example, if a person has signed a message with a private key or published a public key for use in communications and then is able to decrypt data encrypted with this key, it is *almost certain* that they are the originator of the public keys and messages. This is essentially taking the idea of digital signatures -- where you *want* to prove your identity -- and turning it around in order to positively identify a party that may not wish to be known.

## Tools of the Trade: Magnifying Glasses and Microscopes

Just as traditional forensics has standard information gathering tools such as magnifying glasses, microscopes, swabs, etc., CFS has many common kinds of tools used for digital information gathering, some of which are directly analogous to traditional forensics techniques, while some are unique to the field. Additionally, while there are many commercial applications and suites to perform CFS, we will focus on common open source applications. Interested students should investigate commercial applications on their own; they will not be a part of this lab.

### Disk Imaging

One of the first steps in performing FCS is to take a sector-by-sector copy of the **entire hard disk**, including the boot sector, swap, other partitions, and protected areas (such as vendor-supplied restore partitions). This will be large binary file that contains the exact contents of the disk sectors, including residual data, temp files, and other information. Linux and BSD (including OS X) can mount these binary files as though they were typical block devices. Best practices dictate that these images be created at the first prudent opportunity. After the image is complete, hash digests of the file are taken, and cryptographically validated copies of the disk image are used for all future investigations. All work is performed on copies of these images. In this way, neither the original physical disk, nor the first validated copy are changed.

### Editors / Viewers

Next to traversing the filesystem in a shell, editors, viewers, and monitors are the primary way that FCS investigators view the data under investigation. Your favorite text editor is the FCS equivalent of the "trusty magnifying glass" whereby you open files looking for clues and evidence. Good text editors can also open binary files, displaying any ASCII data that may be embedded in them. Other utilities, such as the Unix utility **strings** will also find all ASCII strings within a file. Hex editors typically display hexadecimal data in one column wth the same data represented in ASCII in a second column. Registry editors are used to edit databases such as the Windows Registry and increasingly, similar databases in the Unix world.

Other viewers include tools to scan live memory (a delicate task we don't have time to cover here), often displayed in a format similar to a hex editor. This can recover important data such as encryption keys that are no longer in disk but are stored in memory. Other useful applications include process monitors (**ps**, **top**), network monitors (**netstat**), filesystem monitors (**lsof**), etc. that are used to show what is happening on a live system. (for Windows monitors, see the TechNet SysInternals Website.) Another class of viewers include debuggers to test applications and inspect core dumps.

It should be noted that many system-wide exploits (also known as "rootkits") automatically replace built in system tools (like process monitors) with versions that don't display malicious software, so the conscientious FCS must take pains to use "clean" versions of the software. This typically involves running the applications from a trusted, read-only media source.

## System Logs and Process Accounting

Most servers and UNIX-like systems such as Linux, BSD, and OS X keep verbose logs, typically in a standard location such as /var/log. These logs cover many aspects of the systems' operation, including kernel operations (/var/log/kern.log), system operations (/var/log/syslog), and various other applications with their own respective logs. Logs are typically rotated (renamed with numbers), compressed, and deleted after a period of time to keep the log files from growing too large.

Some systems have more advanced process accounting installed, whereby every action taken by every user is logged in a private database that adminstrators can view at their discretion. This kind of process accounting often takes a meaningful amount of system resources (including space for logs) so this is generally not used unless specific needs dictate otherwise.

Finally, in the Unix environment, most shells create a hidden file called something like `/home/username/.history` that often contains the last series of commands the user entered. Similarly, tools such a `last` on Unix will show the login session history of the user specified (filtering the binary log file /var/log/wtmp). Modifying or deleting these files are common strategies of system attackers attempting to cover their tracks. (Windows has some analogous facilities but we will not cover them in this exercise.)

## Network Scanning and Monitoring

One way to determine if a system is compromised is to monitor the traffic that it creates and receives. This can involve something as simple as using a port scanner like **nmap** to map the open ports, **iptraf** to monitor connections, or using something more powerful like **tcpdump** or **Wireshark** (formery Ethereal) to filter and view packets. A workstation that is sending data to an outside source or receiving commands from the Internet can be passively monitored with tools like this in order to determine what it is doing and potentially who is responsible. Additional tools can attempt to recreate files from the TCP stream. This kind of network monitoring typically requires a dedicated computer that the network passes through, or a passive ethertap with the monitoring computer attached to it. Finally, the decision to monitor versus immediately archive depends on the situation. If the computer is archived immediately, the investigator can ensure that no evidence is lost, however, if the computer is allowed to continue running, more evidence may be created or discovered.

## Software Scanners

Another way to determine if a system is compromised is to scan the software installed on the system with trusted tools. Antivirus and malware scanners already perform these kind of scans on live systems; in fact, it is conceivable that (after archiving) the investigation of a system might include an antivirus or malware scan in order to determine what may or may not be installed. There are some applications that are specifically designed for the Unix environment that look for known rootkits. These scanning applications are typically self contained and are run from trusted (usually read-only) media to ensure that they are not circumvented by dishonest system utilities (like process monitors modified to ignore malware). The most well known of these is called chkrootkit.

## Data Recovery Software

Most computer savvy people are now aware that when files are "deleted" on a computer they are not typically completely destroyed. While utilities exist to truly "destroy" files (such as **srm** and **shred** on

UNIX-likes and others), typical deletion means removing the file from the directory listing and marking its blocks as free. This is much faster than actually destroying the data on the disk. However, the data remains as long as the blocks are not reused. Sometimes, enough residual information is left in the filesystem to totally or partially reconstruct the file. Many operating systems and filesystems have "undelete" utilities (such as **e2undel** for ext2 filesystems). However, even for systems without these tools, the same principles can be used read data in unallocated space.

Of course, the longer the filesystem is used post-deletion, the less likely it is that the sectors in question will be recoverable. The savvy FCS investigator will attempt to perform an undelete or file recovery process at the **first prudent opportunity**.

### Physical Data Recovery

Ultimately, computers are made out of physical parts. At the "bare metal" level, data is manifested by magnetic flux and the state of transistors. Sometimes disks are damaged or partially erased, or important data in RAM is lost. When this happens, the only recourse is physical recovery. Physical recovery is extremely expensive because of the advanced techniques, expertise, and dedicated hardware that must be employed.

Hard drives that are broken can often be repaired by replacing parts from other identical model drives. These drives are then typically imaged immediately after recovery since recovery is *not* repair. Floppy disks can potentially be repaired, and in many cases, disk contents can often be read despite physical errors (which would cause most operating systems to give up).

Computer RAM will contain residual data after power loss; if the RAM is appropriately chilled and brought to the proper facilities quickly, it may be possible to recover the contents of the memory prior to power loss. In a 2008 research paper, scientists at Princeton showed that liquid nitrogen (and even compressed "air") could maintain data in RAM without power from seconds to minutes. What's more, cryptographic engines regularly keep multiple copies of a key in memory and use "key schedules" for computing keys, both of which can be used as redundant information when reconstructing cryptographic keys.

Similarly, there are advanced techniques for data recovery off disks, by looking into the residual magnetic flux that remains on a hard drive even after "secure deletion". This technique can involve reading "between the tracks", and is roughly analogous to going to a dirt race track and following the tread of one of the cars even though every car (including itself) has driven multiple times over the track you're following.

## Inside The Mattress: Where Data Hides

In addition to typical cleartext and encrypted files, there are many common places in a computer system that should be inspected for data.

- Deleted Files: As discussed above, "undeleted files" can often be recovered using undelete utilities.
- Temporary Files: Many applications create temporary files they use to store data they are processing. Sometimes these temporary files contain sensitive information, like a text document that was being edited, or encryption keys being used to encrypt or decrypt a message. Temporary files live in different places in different operating systems, but in Windows they are often located at C:\, C:\Temp, C:\Windows\Temp, C:\Documents and Settings\Username\Local

Settings\Temp, or wherever the file was being edited. In Unix, temp files are usually either created in the world readable /tmp directory, the current directory, or potentially in a dedicated hidden folder.

- Hidden Files: In Windows, files are hidden by setting an attribute on them. In Unix, a file is hidden if its name begins with a dot. In both paradigms, hidden files are often used by applications to store all kinds of application specific data. For example, in Linux, the Firefox browser keeps its data under the '~/.firefox' directory for each user. Command history and startup scripts in Unix are often 'dotfiles' (UNIX lingo for hidden files) in the user's home directory and can contain important information.
- Caches: Many computer subsystems use caches to improve performance. For example, most web browsers keep a local cache of the last N megabytes downloaded. When requests are made to a remote webserver, the browser attempts to determine if a file has been changed since it was orginally downloaded. If the file hasn't changed, the browser will display the file from the cache rather than using the network to download a new copy. Web caches have grown tremendously as the web has become more important. A good investigator should look at web cache data.
- Spools: Spools are sort of like a reverse cache and can exist in memory or on disk. A spool is a special location that data is buffered into in preparation for a different application or peripheral to process. Spooling is often done because the relationship between the process originating the data and the process using the data is asynchronous. For example, when a program wants to print, it typically writes data into the print spooler, which sends the data to the printer at whatever speed the printer is able to accept it. Another example is a mail spool, which is a file that grows with new mail messages as they arrive. Spools can contain meaningful information if they have not been securely purged.
- RAM: Critical information such as encryption keys, last actions, chat logs, images, and more are often left in RAM after an application closes. As previously mentioned, RAM can sometimes be read after a power off if it is handled properly.
- Backups: entities often spend considerable resources to develop a successful and reliable backup plan including disaster recovery, offsite storage, etc. Unfortunately for them, today's good backups are tomorrow's e-discovery liability. If you can find backups of the entity's important data, it may not matter if they deleted it from the computer.
- Printouts and Notes: As much as we like computers, sometimes you just can't beat plain old paper. Sometimes the information you need is on paper at a person's workstation or elsewhere in their posession. The canonical example of this is the sticky note with the username and password stuck to the monitor, or maybe in the desk, or if they are very clever, under the desk drawer. Paper materials, books, etc., should all be gathered and inspected. That jumble of letters and numbers in the margin of a book might be the cryptographic key you need.
- External Media: Today, digital cameras, iPods, thumb drives, CD/RWs, DVDs, external hard drives and more are common and easily overlooked sources of readily available, concealable, high-capacity storage. Additionally, each device has its own idiosyncrasies that may be important. For example, removable media in a digital camera is obvious -- but what about built in flash memory? Regardless, any such devices should definitely be imaged and archived along with any other storage media if it is legal to do so.
- Swap Space: Swap space (or Virtual Memory) is a portion of the disk dedicated to storing chunks of less-used data on disk in order to free more physical RAM. This is typically done for all user-space memory, regardless of security policy. In recent versions of Windows, the swap space is a file named `C:\pagefile.sys`, while most Unix operating systems use a dedicated disk partition. Regardless, when memory is swapped out to disk, it is as though an application copied that section of RAM to a file for you. Furthermore, swap spaces are typically not cleaned

even if the data is no longer in use; this means that swap usually contains unencrypted application memory, sometimes going back as far as several days or weeks. Swap is a good place to find passwords, application data, browser history, and more. You can use editors, special applications, or other viewers to inspect the swap file.

- Memory Buffers: today's computers are fast in part due to improved buffering which is possible because of the low price of physical memory. This means that portions of memory can have copies of data that you wouldn't necessarily expect. For example, most modern hard drives contain significant amounts of built-in RAM for buffering requests. Similarly, expansion cards such as network, sound, and particularly video cards can have large amounts of onboard RAM that may contain data. Finally, most modern processors have a considerable amount of onboard cache. Potentially all these areas and more could hold data important to an investigation.
- Network Storage: Users often don't consider or realize that the files available on their computer may be physically located somewhere else on the network. Also, sometimes the user's profile information is stored on a networked resource; this can include web caches, passwords, history files etc. In the event that user profiles are stored on the network, it is very possible that this data is also backed up.
- ISP Records: The Internet Service Provider that was the upstream network provider for the computer being investigated might have useful information, such as DNS lookups, traffic and usage patterns, firewall logs, etc. Furthermore, if the investigation involves communication between the local system being investigated and a remote host (such as a web server the suspect accessed), there may be pertinent records at the remote host or anywhere in between. These records can sometimes be subpoenaed and investigated which can help to fill in missing information in local logs.
- Steganography, etc.: Literally speaking, steganography is "concealed writing." While cryptography is used to reversibly transform data to keep its meaning secret, encrypted data is often readily apparent because of its highly-random appearance. However, sometimes knowledge of message transmission is as sensitive as the message contents. Steganographic techniques are used to conceal data so that adversaries are not even aware that it *exists*.

  Plain text written in "invisible ink" is a classic form of steganography. The text is not encrypted, nor is it stashed in an unlikely location. Rather, it is "hidden in plain sight."

  Typically, the use of the word in a digital context refers to intentionally hiding a message in another file. The canonical example is an image file in which pixel values are modified in an algorithmic manner. Ideally, the modification inserts the message into the picture without modifying the visible appearance or leaving identifiable artifacts in the data itself. Other techniques involve hiding data inside audio or video. Although many freely available utilities are available to perform steganographic techniques, it is not clear how often steganography has been used "in the wild".

## Data Integrity : The Evidence Bag

One of the most important issues in evidence gathering is to ensure that the evidence was not tainted when it was gathered, and further that it is not tainted while it is in storage. In the traditional forensics world, this is done with rubber gloves, masks, plastic bags, and documentation. While some of these techniques apply directly to FCS, the nature of digital data requires some additional methods to validate and archive data.

### Chain of Custody

The concept of chain of custody is key in traditional forensics in addition to FCS. It is essentially the requirement that all evidence accesses and transferral of custody be logged and verified such that the process and history of transferral is not vulnerable to legal challenges. For example, imagine what would happen if an investigator "bagged and tagged" a bloody knife at a crime scene, but then loaned it to his buddy, who later brought it back to the police station and gave it to the evidence clerk. There is no trustworthy "chain of custody" any more, because the buddy could have potentially switched knives, or damaged evidence present on the knife, etc. The attorneys for the defendant would be remiss if they didn't try to have the knife thrown out as evidence because it was no longer trustworthy.

Chain of custody relies on proper evidence handling, consistent logging, and verifiable evidence transferral in order to protect the integrity of the evidence. The same process is necessary in FCS, including bagging and tagging of papers, disks, and files, and also to the collection of digital data such as hard drive images.

### Cryptographic Hashes

In traditional forensics, evidence integrity is typically shown through photographs, notes, proper handling, and tamper-proof bags. Tamper-proof bags and secure storage with law enforcement ensure that the evidence is not changed once it has been collected. In FCS data collection, cryptographic hashing is used to show that data is unchanged.

Cryptographic hashes, such asMD5 andSHA-1 are functions that produce a unique fixed-length result for any given input. For example, the MD5 digest hash of the string "Hello, World!" (without quotation marks) is **bea8252ff4e80f41719ea13cdf007273** whereas the MD5 digest hash of the GNU Public License is **a17cb0a873d252440acfdf9b3d0e7fbf**. You can create your own md5 hashes using the UNIX utility md5sum, or with several online utilities.

Hashes are used to create manageable digests of collected data immediately upon collection. Ideally, the investigator would create two disk images from the system being investigated and verify that the hash digest of both images match. This ensures with very good probability that the two images are bit-for-bit the same and represent exactly what is on the physical disk. After reliable disk images are collected (and logged) and any other investigation has or data recovery been completed, the original system is shut down and future work commences only on **copies** of the hashed images. This ensures that the images are reliable and unmodified, and allows the investigator to use potentially destructive techniques on the data without jeopardizing the integrity of the original data.

Any further chain of custody documentation should reflect the hash digest value at the time of the data transfer to show that the data has not been changed since it was originally obtained.

Finally, it should be noted that the strengths and weaknesses of hashing algorithms are constantly under analysis by cryptologists. If a researcher can show that she can generate the same hash for two different pieces of data (known as a *collision*), she has shown that the digest is not cryptographically secure. The threat is that an attacker *might* be able to change the data (e.g., delete the meaningful evidence from a disk image) in such a way that both the original and new digests are identical. In practice this is incredibly difficult, because even if arbitrary collisions can be found, their input texts are likely to be wildly different. Nevertheless, both MD5 and SHA-1 have recently come under a fair amount of scrutiny for some demonstrated attacks; stronger hashes such as SHA-2, SHA-256, etc., are starting to be preferred for hashing when integrity and security are essential.

## The Importance of Documentation

The importance of documentation cannot be overstated, especially if the work being done has a chance of appearing in legal proceedings. If it isn't properly documented, it may not be repeatable, and it is possible that the work will be called into question or outright dismissed on the basis of validity or legality.

All aspects of the investigation should be documented, starting with the initial state of the system when it comes into the investigator's control. The configuration of the system should be described so that it can be recreated, the state of the work area, screen contents, etc. It is also common to use a digital camera to photograph the work area, configuration of the computer, and other details that can be represented in a photograph.

If the work is transferred to a different investigator or inspected by someone else, any undocumented work is likely to be thrown out or at the very least repeated and documented the second time around.

## You Can't Handle The Truth: Legal Protocol

When dealing with law enforcement, CFS includes a significant amount of legal protocol involving rules of evidence, accepted forensic software, documentation, and chain of custody issues that can make the difference between admissible and inadmissible evidence. Furthermore, investigation of data and computer resources without the proper authorization could result in a civil suit against the investigator. This kind of work must not be done without a clear understanding of the protocol, authority, and expectations placed on the investigator.

## Know Thy Enemy / Know Thyself

In any CFS investigation it is critical to both know your enemy and your own abilities. For example, is it important to consider whether your adversary may have installed "booby traps" that will damage evidence (e.g. erase the hard disk on a proper shutdown). Is your adversary a skilled attacker who has gone to great lengths to cover his tracks, or is he a novice who has left a smoking gun? Is the smoking gun you found merely a decoy? Assessing the abilities of your adversary can help you make decisions down the road -- should you first search the images on the computer for evidence of steganography, or should you look at free blocks on the disk for deleted data? The reality is that modern computers have *so much data* that it may not be practical or possible within the time or budget constraints to do a completely thorough job.

It is also equally essential to know your own abilities -- are you able to deal with fragile residual data in volatile RAM? Are you as comfortable monitoring network traffic as you are debugging a core dump? Are you sure you know how to properly handle the evidence? In the commercial world of programming and system administration, a lack of experience often results in an inconclusive internal investigation. But in the legal world, a lack of experience on the part of an CFS investigator can be catastrophic for the legal process.

## The Art of Forensic Science

In the end, Computer Forensic Science is an art as well as a science. On the one hand, it includes the hard science of method, facts, expertise, and data; but it is also an art in that it relies on experience, intuition, creativity, and the ability to make the right decision when there is no obvious way to decide what the best course of action is. In the real world, investigations (both legal and in industry) are often severely limited in terms of time and money. A company might be losing significant revenue every day the computer is out of service, and at a certain point, finding the culprit is less important than getting back online. A legal case might have other time constraints on it such as fixed dates for court proceedings and filings. It is often the combination of science and good judgement that makes the difference between finding an answer or failing.

## Software Tools

### loadimage.sh: load forensics lab images

`loadimage.sh` is the script that loads the compressed disk images from storage into the testbed. These are not automatically loaded to save setup time. To copy an image to your workbench:

```
$ cd /images

$ sudo ./loadimage.sh act1.img
Loading image. Please wait...
`/proj/UCLAClass/236/labs/forensics/images/fdisk.ul' ->
`/images/fdisk.ul'
[ time passes (~10 minutes or so...) ]
$
```

You can select from `act1.img`, `act2.img`, or `act3.img`, for the three parts of the lab.

After this point, the image is decompressed and ready for you to mount it (see below).

The tools `chkrootkit` and `john` are also available in this directory. The `e2undel` source code is in this directory, but the application `e2undel` is already installed in the local path.

### losetup and mount: mount disk images

#### losetup

Mounting a "`dd` image" in Linux requires the use of two utilities: the traditional `mount` command, and the `losetup` command. From the `losetup` man page:

"`losetup` is used to associate loop devices with regular files or block devices, to detach loop devices and to query the status of a loop device."

This means that with `losetup`, you can take a regular file containing a valid filesystem (like a `dd` image) and associate it with a loopback device, allowing it to be accessed as though it were a normal block device (like a hard drive). `losetup` also takes parameters to specify where in the file to begin the loopback device, effectively allowing you to select partitions which exist at an offset within the `dd` image.

> Typically, the starting block number and block size must be extracted from the partition table of the disk. We've already done that for you; you can use the examples here to mount the root partition or swap disk of the forensic images.

To calculate the offset, take the block number and multiply it by the number of bytes per block (512 for this lab). Therefore, to mount sda1, which begins at block 63; 63 * 512 = 32256:

```
$ sudo losetup /dev/loop0 actN.img -o 32256 # associate sda.image with
device /dev/loop0 at offset
$ sudo mount /dev/loop0 sda1  # mount device /dev/loop0 at mountpoint
directory sda1
```

The first argument to `losetup` is the kernel loopback device you are going to attach the image to, and the second argument is the `dd` image file. The -o flag indicates that there will be an offset taken from the `fdisk` output as the start of the partition.

If we wanted to associate sda2 (the swap partition) with a loopback device, we would need to calculate the new offset, which is 2923830 * 512 = 1497000960:

```
$ sudo losetup /dev/loop1 actN.img -o 1497000960 # associate sda.image
with device /dev/loop/1 at offset
```

Typically, you don't want to mount swap; instead you would read it (after setting up the loop device) by editing the file /dev/loop1 with a hex editor (e.g. hexedit) or other low level tools like `grep` or `strings`. `vim` will complain if you try to open a block device as a file; this is because ASCII editors like `vim` are usually not designed for editing the data in block devices.

To disassociate loopback devices and files, make sure any outstanding mounts are unmounted and then execute this command:

```
$ sudo losetup -d /dev/loopN
```

... where N is the number of the loopback device in use.

**mount**

The arguments for `mount` are much simpler: the first argument is the block device (the newly configured /dev/loop device) and the second argument is an empty directory (you create) to be used as the mountpoint for the filesystem.

For example, a typical mount is invoked like this:

```
$ sudo mount /dev/sda1 mountpoint # mount 1st partition of sda at
directory 'mountpoint'
```

Loopback devices work the same way:

```
$ sudo mount /dev/loop1 mountpoint # mount loopback disk at directory
'mountpoint'
```

mount something read-only:

```
$ sudo mount /dev/loop1 mountpoint -o ro # mount loopback disk read only
at directory 'mountpoint'
```

To unmount a disk, execute:

```
$ sudo umount /dev/loop1
```

### e2undel: undelete files from an ext2 filesystem

`e2undel` is a utility to undelete files from ext2 filesystems. `e2undel` works by inspecting blocks marked free to see if they contain the beginning of a file. If they do, `e2undel` will assemble the data into a file and recover it.

The basic syntax of `e2undel` is straightforward, but the interface is unique to say the least. To scan a disk for recoverable files, execute this command:

```
$sudo e2undel -d /dev/loop0 -s /images/recovered -a -t
```

Syntax breakdown:

- -d /dev/loop0 -- specifies the block device to search
- -s /images/recovered -- specifies the directory to save recovered files. Files shouldn't be saved to the disk being searched if it can be helped, because valuable disk blocks may be overwritten by `e2undel`.
- -a -- tells `e2undel` to look everywhere for files, not just in a special `e2undel`-specific journal
- -t -- tells `e2undel` to try and determine what kind of file an inode contains.

The interface of `e2undel` is odd. After the initial search, it displays the found files, ordered by file owner UID and date deleted. The leftmost column holds the newest files, while the rightmost column holds old files. You first select a UID that has recoverable files, and then select an inode (file) to recover. The file is recovered (without a filename) in the save directory specified on the command line. When you're done, you can quit the application and go examine the files you recovered.

Example:

```
$ sudo e2undel -d /dev/loop0 -s /images/recovered -a -t
```

e2undel does not work for ext3 filesystems (the more modern, journaling version of ext2). The ext3 FAQ says it best:

> Q: How can I recover (undelete) deleted files from my ext3 partition?
>
> Actually, you can't! This is what one of the developers, Andreas Dilger, said about it: In order to ensure that ext3 can safely resume an unlink after a crash, it actually zeros out the block

pointers in the inode, whereas ext2 just marks these blocks as unused in the block bitmaps and marks the inode as "deleted" and leaves the block pointers alone. Your only hope is to "grep" the disk for parts of your files that have been deleted and hope for the best.

### strings: search for strings in a file

From the `strings` man page:

> "For each file given, GNU `strings` prints the printable character sequences [some characters are non-printable] that are at least 4 characters long (or the number given with the options below) and are followed by an unprintable character. By default, it only prints the strings from the initialized and loaded sections of object files; for other types of files, it prints the strings from the whole file. `strings` is mainly useful for determining the contents of non-text files."

Example:

```
$ strings /var/log/wtmp

pts/4
pts/4pedro
10.179.38.83
pts/4
ts/4pedro
10.179.38.83
\]I(
pts/21
...
```

In this example, we ran the utility `strings` on the binary logfile `/var/log/wtmp`. As we can see, there are some plaintext strings within the binary log. `strings` can be used to read things like the swap file and other binary files that may contain plaintext information.

### chkrootkit: check for rootkits

None of the images for this lab requires the use of `chkrootkit`. While this is an important tool to understand for use in the real world, it is not necessary for this course. You should read the entry below for your own edification, and you may want to practice using it on DETER, but it is **not** necessary to complete the forensics lab.

chkrootkit is a application containing some small C programs and a bash script that searches a disk image for signs that it is compromised with any number of different possible "rootkits". `chkrootkit` uses mostly standard utilities to make its tests, which makes `chkrootkit` extremely portable across

Unix platforms.

However, because `chkrootkit` relies on tools present on the local computer system, it is critical to make sure that those utilities (such as `awk`, `cut`, `head`, `strings`, `ps`, etc.) are not trojaned. Typically, this is done by running `chkrootkit` from a bootable cdrom or other trusted, write-protected media. In our case, we believe that our work system is secure; it is the disk images we are inspecting that may be compromised. Therefore, it is safe to run `chkrootkit` from the commandline on DETER.

To use `chkrootkit`:

1. Find the current source tarball in the `/images/` directory.
2. Extract the tarball: `tar -xvzf chkrootkit-VERSION.tar.gz`
3. `cd` into the extracted source directory.
4. Execute `make sense`
5. Execute `chkrootkit -r root_directory (where root_directory is the mounted disk image)`
6. Inspect the output of `chkrootkit` for anything suspicious and follow up on it.

Like any anti-malware scanner, just because `chkrootkit` may not find anything does not mean that the system is not compromised -- it merely means that `chkrootkit` *has not detected* any compromise on the system.

**gpg: open-source cryptography**

gpg is GnuPG -- the Free Software version of PGP (Pretty Good Privacy). gpg is really designed to use public key cryptography where each party has one piece of a larger cryptographic key. However, gpg can do other kinds of cryptography, too. The bad guys in Act III use gpg with what are called "symmetric" keys -- it's just a password or passphrase that is the key to encrypting and decrypting a piece of information.

For example:

```
$ echo "the quick brown fox" > brnfox
$ cat brnfox
the quick brown fox
$ gpg --symmetric brnfox
(i enter and confirm "jumped over the lazy dog" as the password)
$ ls brnfox*
brnfox     brnfox.gpg
$ shred -u -z brnfox    # delete the original copy of brnfox
$ gpg brnfox.gpg
gpg: CAST5 encrypted data
gpg: encrypted with 1 passphrase
(i enter the passphrase)
$ cat brnfox
the quick brown fox
```

**john the ripper: brute force user passwords**

**WARNING:** Using `john` or other password crackers is against the rules in many environments (e.g. most university computer labs)! `john` may be detected as malware by your own computer or managed environments you may use. We recommend you only use `john` on DETER unless you really know what you're doing.

John the Ripper is a popular, powerful, and open source password cracker. `john` takes the `/etc/passwd` file (or the /etc/passwd and `/etc/shadow` files if the passwords are shadowed) and attempts to crack them starting with popular and simple passwords and continuing on to progressively more complex passwords.

To use `john` in the Exploits lab image, copy the /etc/passwd file into the `/images/john-1.7.2/run/` directory, cd into it, and execute:

```
$ ./john passwd
```

`john` will attempt to crack the passwords, printing anything it finds to standard out. `john` has many options and other features, which you can explore on your own if you are interested.

**shell tools: less, tail, head, cat, and grep**

Several Unix utilities are invaluable for looking at system logs: `less`, `tail`, `cat`, and `grep`. Additionally, the Unix feature of piping output from one program as input to the next program is especially useful.

**cat**

`cat` (short for concatenate) opens a file and prints it to standard out (which is typically your console). `cat` doesn't have any "brakes" -- it will flood your terminal -- but it is indispensible for sending data into other Unix applications. The command:

```
$ sudo cat /var/log/messages
```

... will print the file /var/log/messages to screen as fast as your connection will allow. `cat` terminates at the end of the file. You can also quit `cat` by pressing `^C` (Control-C).

Most of the time, however, you want to control how much of a file you see. The following tools help you do just that.

**less**

`less` is the better replacement for the Unix file pager `more`. To use `less`, enter:

```
$ less /var/log/messages
```

... or

```
$ cat /var/log/messages | less
```

And you will be greeted by the top of the system log. To move up and down in the file, you can use the arrow keys, or page up, page down, home, and end. You can also search within the loaded file using the / (search forward) and ? (search backward) command, like this:

```
...
xxx.105.166.xxx - - [02/Sep/2007:07:15:32 -0700] "GET /foo/SomePage
HTTP/1.1" 200 15289
xxx.105.166.xxx - - [02/Sep/2007:07:17:23 -0700] "GET /foo/ HTTP/1.1" 200
16557
/SomePage<enter>
```

Note the bottom line, `/SomePage<enter>`. When you press "/" (the search forward key), less will print it at the bottom, and wait for you to enter a search string. When you're finished, press enter. This will jump to the first and highlight all occurances of the string "SomePage". To see the next result, press "/" again and hit enter. In this way, you can cycle through all occurrences of a string in a text file. The command "?" works exactly like "/" but searches backwards. Both ? and / accept regular expressions (also known as regexes) in addition to normal strings -- if you know regexes you can create vastly more expressive search patterns.

Hit q to quit `less`.

**system logs**

Examining system logs is an acquired skill. The first task is always to determine what each column in the log represents. In the case of `/var/log/messages`, it's easy: day, time, hostname, process name and process id (PID), and message.

```
...
Sep  5 21:49:08 localhost postfix/smtpd[19090]: connect from
unknown[xxx.55.121.xxx]
Sep  5 21:49:10 localhost postfix/smtpd[19090]: lost connection after
CONNECT from unknown[xxx.55.121.xxx]
Sep  5 21:49:10 localhost postfix/smtpd[19090]: disconnect from
unknown[xxx.55.121.xxx]
Sep  5 21:49:10 localhost postfix/smtpd[19090]: connect from
unknown[xxx.200.87.xxx]
Sep  5 21:49:33 localhost imapd[19332]: connect from 127.0.0.1
(127.0.0.1)
Sep  5 21:49:33 localhost imapd[19332]: imaps SSL service init from
127.0.0.1
Sep  5 21:49:33 localhost imapd[19332]: Login user=jimbo host=localhost
[127.0.0.1]
Sep  5 21:49:33 localhost imapd[19332]: Logout user=jimbo host=localhost
[127.0.0.1]
Sep  5 21:49:42 localhost postfix/smtpd[17190]: timeout after RCPT from
unknown[xxx.125.227.xxx]
```

```
...
```

In this case, the log refers to mail and IMAP requests. Specifically, a host with no DNS resolution connects to the postfix smtpd (outgoing mail server) at 21:49:08, but disconnects. Another (different) unknown host connects at 21:49:10. Then the local user "jimbo" logs into the IMAP server. Finally, a different server (which must have been previously connected) disconnects from the smtpd.

Compare that to lines from an Apache log:

```
...
xx.105.166.xxx - - [02/Sep/2007:07:14:04 -0700] "GET
/wiki/modern/css/print.css HTTP/1.1" 200 775
xx.105.166.xxx - - [02/Sep/2007:07:14:05 -0700] "GET
/wiki/modern/css/projection.css HTTP/1.1" 200 587
xx.105.166.xxx - - [02/Sep/2007:07:14:05 -0700] "GET
/wiki/modern/img/moin-www.png HTTP/1.1" 200 150
xx.105.166.xxx - - [02/Sep/2007:07:14:05 -0700] "GET
/wiki/modern/img/moin-inter.png HTTP/1.1" 200 214
...
```

In this case, the log format is: IP address, date and time, HTTP request type, HTTP status code, and bytes transferred. This log represents the same user (or less likely, multiple users at the same IP) viewing a page on a wiki.

**tail and head**

`tail` and `head` respectively print out the last 10 and first 10 lines of their input file. Typically, `tail` is used to check the end of a file, but it is also very commonly used to "watch" a log file. Using the command:

```
$ sudo tail -f /var/log/messages
```

... you can watch the messages file grow. ^C quits.

**grep**

`grep` is what makes `cat` useful in this context. `grep` is a filter that uses patterns (including regexes) to filter lines of input. For example, given the snippet of the messages file from before, if a user "pipes" the output of `cat` into `grep` and filters for "xxx.55.121.xxx" like this:

```
$ cat /var/log/messages | grep xxx.55.121.xxx
```

... she will see only lines matching xxx.55.121.xxx:

```
...
Sep  5 21:49:08 localhost postfix/smtpd[19090]: connect from
unknown[xxx.55.121.xxx]
```

```
Sep  5 21:49:10 localhost postfix/smtpd[19090]: lost connection after
CONNECT from unknown[xxx.55.121.xxx]
Sep  5 21:49:10 localhost postfix/smtpd[19090]: disconnect from
unknown[xxx.55.121.xxx]
...
```

If a filter has too much output, just pipe the output from `grep` into `less`, like this:

```
$ cat /var/log/messages | grep kernel | less
```

... and now you can use the features of `less` to examine your result.

As an alternative, you could pipe the output to a file like this:

```
$ cat /var/log/messages | grep kernel > kernel_grep.txt
```

... and you could then use `less` on the file kernel_grep.txt you just created.

`grep` has many advanced features, such as negation (`grep -v somestring`). For more information see "man grep".

**find, xargs, and locate**

**(find files on a system depth-first or via table lookup)**

Users of more "user friendly" operating systems such as Windows and OS X are spoiled when it comes to finding local files, because while the graphical tools like Windows find, Apple's Spotlight Search, and Google Desktop are fast and easy to use, they are generally not nearly as flexible or expressive as the standard Unix utilities for finding files, `find`, `xargs`, and/or `locate`.

**find -- find files on the system**

`find` can be used to search for files of various names and sizes, various modification times, access permissions, and much, much more. However, the syntax for `find` is a black art into which most of its users are barely initiated. We'll discuss the basics here so you can use it. If you want to know more, read the manpage or look online.

The basic command format is "`find [path [expression]`", where 'path' is the directory to start searching in and `expression` is some compound expression made up of *options*, *tests*, *actions*, and *operators*. The expression modifies the search behavior: *options* specify things like how many levels deep to search, whether to follow symlinks, whether to traverse filesystem boundaries, etc. *Tests* specify conditions like matches on the filename, modification date, size, etc. *Actions* can be defined to delete matching files, print the files, or execute arbitrary commands. (The default action is to print the name of any match.) *Operators* are logical operators for combining multiple expressions. Expressions limit results. Accordingly, no expression at all will "match" everything and the default action will print the relative paths of all files that `find` encouters.

You usually don't want to list every file in a subtree. In this case you may want to limit the search with an expression. An expression begins with the first of several expression options that begin with a hyphen (such as `-name` or `-mtime`) or other special characters. The expression can also specify actions to take on any matching files (such as to delete them). Expressions can become very complicated. And like any complicated machine, the more complicated the expression, the more likely it is that `find` will not do exactly what you want. If you need to create expressions beyond the complexity addressed here, please see `man find` or a tutorial online and try examples on your own.

Here are a few simple examples to get you started:

"Find all files ending with .txt, starting in this directory. Use `head` to show me only the first 5 results."

```
$ find . -name "*.txt" | head -n 5
```

"Find all files ending with "~" or ".swp" (common tempfile designations), starting in the subdirectory `public_html`." To do this, we'll use the OR operator `-o`.

```
$ find public_html -name "*.swp" -o -name "*~"
```

"Find all files on the root filesystem not modified in 6 years (2190 days). Start in the root directory (/), and do not descend into other filesystems (such as NFS directories). Use head to only show the first 5 results."

```
$ find / -mount -mtime 2190 | head -n 5
```

Finally, here's an example from `man find`:

```
$ find /    \( -perm -4000 -fprintf /root/suid.txt '%#m %u %p\n' \) , \
             \( -size +100M -fprintf /root/big.txt '%-10s %p\n' \)
```

This translates to: "Traverse the filesystem just once, listing setuid files and directories into /root/suid.txt and large files into /root/big.txt." (`man find`)

The backslashes ("\") in the above example are called "escapes." Backslash escaping tells the shell either to treat the escaped character literally -- not as a a special character -- or vice versa (that a regular character has a special meaning). Above, the parentheses are escaped because they are input for `find`. If they were not escaped, the shell would interpret them in its own way, because parentheses are special for the shell. Similarly, a line break created by pressing enter signifies the end of a command -- unless it is escaped. Here, the line break (visible as the backslash at the end of a line) is escaped to tell the shell that the command is continuing on the next line. Finally, an escaped "n" character, (`\n`) represents a line break in printf-style format strings.

Be careful with advanced uses of `find`, especially if you are deleting files. As you might imagine, unexpected things often happen when shell escapes, quoting and expansion are involved.

Finally, because find performs a depth-first search over potentially the entire filesystem tree, it can take a long time to run. You may have better luck with `locate` depending on what you're trying to do. (See below.)

**xargs -- build command lines from standard input**

In the Unix environment, the output of one program can be easily piped into the input of another program. `xargs` allows you to use the output of one program as *arguments* to another process. This is very useful in conjunction with tools like `find`, `cat`, and `grep`. This is only a brief introduction to `xargs` -- for more information, please see `man xargs`.

The most common use of `xargs` is to take the output from a command and use it as a parameter in a different command. For example:

"Run `strings` on every file on the system that ends with ".pwd" (case insensitive) and sort the results ASCIIbetically."

```
$ find / -iname "*.pwd" | xargs strings $1 | sort
```

Each line of output from `find` is used by `xargs` to construct the `strings` command. After this, the output of `strings` is put into a different order by `sort`.

Here's another example:

"Delete the first four files ending in ".swp" or "~" that have not been modified in 24 days starting in this directory."

```
find . -mount -mtime +24 \( -name "*.swp" -o -name "*~" \) | head -n 4 |
xargs rm $1
```

Note the use of `head` to limit the action. Without `head -n 4`, `xargs` would have removed every matching file.

**locate -- a faster way to search**

`find` does not keep a database of information to use; every time you run a new invocation of `find`, it traverses the entire directory structure looking for matches. This allows you to search the current state of the system, but also lacks the advantages that pre-indexing can provide. `locate` and its helper command `updatedb` are more like Google Desktop than Unix `find` in this sense.

**Before using `locate`, `updatedb` must be run as root to index the entire disk, recording it in a database.** Then, the `locate` command is used with a few simple options to match filenames. This can be *much* faster than using find, especially if you don't know exactly what you're looking for or where it might be. As always, see `man locate` for more information. To update the locate database, run:

```
$ sudo updatedb
```

`sudo` is used to make sure that you can index the entire disk.

The following examples assume that updatedb has been run as root recently:

"Find all files ending in '~' or '.swp'."

```
$ sudo locate *~ *.swp
```

"Find all files ending in '~' or '.swp' and delete them."

```
$ sudo locate *~ *.swp | xargs rm $1
```

`locate` will complete the search for the first term and then the second term. So an example like this:

"Find all files ending in '~' or '.swp' and delete the first 10 results."

```
$ sudo locate *~ *.swp | head | xargs rm $1
```

Will only delete ".swp" files if there were *less* than 10 "~" files (since the "~" files are returned first).

`sudo` is used in the locate commands to make sure that you are able to access all of the `locate` database.

**hexedit -- edit hexadecimal and ASCII**

Hex editors are used to edit and view binary data, and typically show hexadecimal and ASCII representations of the same data side by side.

`hexedit` is a common, freely available hex editor.

Example:

```
$ hexedit filename
```

When you open file in `hexedit`, you're met with a display that looks like this:

```
00000000   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00
00  00 00 00 00  ........................
00000018   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00
00  00 00 00 00  ........................
00000030   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00
00  00 00 00 00  ........................
00000048   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00
00  00 00 00 00  ........................
...
```

The first column is the file location (in hexadecimal), the next section is a number of bytes (in groups of four) represented in hexadecimal, and the last column is the same data represented in ASCII.

You can hit tab to switch between editing hexadecimal and ASCII, and can search (starting from the cursor position) using the "/" key as in `less`. To return to the beginning of the file, enter "<". To quit, hit ^C.

Here's `hexedit` editing a copy of gpl.txt:

```
$ hexedit gpl.txt
00000000   09 09 20 20  20 20 47 4E  55 20 47 45  4E 45 52 41  4C 20 50
55  42 4C 49 43  ..    GNU GENERAL PUBLIC
00000018   20 4C 49 43  45 4E 53 45  0A 09 09 20  20 20 20 20  20 20 56
65  72 73 69 6F   LICENSE...      Versio
00000030   6E 20 32 2C  20 4A 75 6E  65 20 31 39  39 31 0A 0A  20 43 6F
70  79 72 69 67  n 2, June 1991.. Copyrig
00000048   68 74 20 28  43 29 20 31  39 38 39 2C  20 31 39 39  31 20 46
72  65 65 20 53  ht (C) 1989, 1991 Free S
00000060   6F 66 74 77  61 72 65 20  46 6F 75 6E  64 61 74 69  6F 6E 2C
20  49 6E 63 2E  oftware Foundation, Inc.
...
```

As mentioned above, hex editors are really designed for editing and viewing binary data:

Here's `hexedit` editing a copy of `/usr/bin/gcc`:

```
$ hexedit /usr/bin/gcc
00000000   7F 45 4C 46  01 01 01 00  00 00 00 00  00 00 00 00  02 00 03
00  01 00 00 00  .ELF....................
00000018   80 96 04 08  34 00 00 00  00 CA 02 00  00 00 00 00  34 00 20
00  08 00 28 00  ....4...........4. ...(.
00000030   1C 00 1B 00  06 00 00 00  34 00 00 00  34 80 04 08  34 80 04
08  00 01 00 00  ........4...4...4.......
00000048   00 01 00 00  05 00 00 00  04 00 00 00  03 00 00 00  34 01 00
00  34 81 04 08  ................4...4...
...
```

`hexedit` can be used to view the Linux swap file. In our own tests, we found a username and password for a retail website. We sure hope they don't save our credit card information!

For more information, see `man hexedit`.

# Introduction

## Episode 12: "POST Mortem"

You are Chip Holmes, a private security consultant specializing in computer forensics with an office in a crumbling 5th story brownstone somewhere in The City. You consider yourself a digital detective -- a grepping gumshoe if you will -- helping people and organizations (such as companies, coppers, and wealthy individuals) investigate computer systems and files, typically when things have gone horribly wrong or something important is at stake. You had just poured yourself a strong cup of joe, put your feet up on the desk and started reading The Times when your assistant Lefty called and told you to check your messages; some new clients called this morning and of course they all think it's an emergency.

It was a pretty typical morning, all things considered. One client was looking for a binary bloodhound to investigate their server that was accused of being infected with a worm. The second client had some sensitive data stolen and they wanted some help interpreting the evidence. The third client was a little more interesting -- it was a high rolling fat cat who was being extorted by black hat scum from some remote corner of the Internet. Typically, none of the clients had any network logs and you could be darn sure they didn't have an Intrusion Detection System either, so the only information available was on the computers themselves.

You typically use a bootable "live CD" (such as Finnix) to create a disk image and cryptographic hash of each computer's hard drive so you can inspect and manipulate the image without messing with the real thing. It takes a long time, but it's the right way to do it. While you investigate the images, the real computers are powered off and in a vault somewhere, but every hour they remain offline in a "morgue" represents big bucks to the clients. The clients all want to know what happened, and what they should do next. Oh yeah, and they want to know yesterday!

## Assignment Instructions

### Setup

1. If you don't have an account, follow the instructions in the introduction to DETER document.
2. Log into DETER.
3. Create an instance of this exercise by following the instructions here, using `/share/education/ComputerForensics_UCLA/forensics.ns` as your NS File.
   - In the "Idle-Swap" field, enter "1". This tells DETER to swap your experiment out if it is idle for more than one hour.
   - In the "Max. Duration" field, enter "6". This tells DETER to swap the experiment out after six hours.
4. Swap in your new lab.
5. After the experiment has finished swapping in, log in to the node via ssh.

Because these labs are all based off of investigating stored, static disk images, you know that network scanners and the like will not be necessary. Furthermore, while "chain of custody" and related issues are very important in real life, they are not necessary for this lab -- just careful investigation of evidence left on the disk and consideration of what may have happened.

Your analysis should be directly supported by the data on the disk image, or if it is unclear what happened exactly, explain why.

Beyond that, the basic method is:

1. Swap in the experiment.
2. cd into the `/images` directory on the experimental node `workbench`. (This experiment only has one node.)
3. Follow the instructions for loadimage.sh to load a disk image. `loadimage.sh` will copy the dd image from a network server to your workbench machine, but you'll still have to mount it.

4. Use `losetup` and `mount` to mount the partitions. there are sda1 and sda2 directories in the /images directory; these are meant for you to mount the first and second partitions of the disk (root and swap).
5. After you have mounted the partition, cd into that directory and start working! You'll probably find tools like hexedit, e2undel, strings, less, grep, and more useful. (See the tools section of this lab for some ideas.)
6. Use those tools (and any others as you desire) to try and answer the questions above. Find data that may be hiding, and analyze it in conjunction with the data left on the system in order to determine (as best as you can) what happened on the system.

**Important:** Do **not** use the `/var/log/wtmp` file for login information; you don't need it, and the information in that file will be incorrect.

## Tasks

### Act I: The University Server

Your buddy, Bob, is now a professor of Computer Science at some big school here in The City. He got an angry email from the Network Operations Center (NOC) at the University saying that his lab's server was infected with a worm -- the NOC determined this because of a huge spike in Internet traffic which occurred at 4 in the morning. He immediately shut it down and brought it in to be imaged. He doesn't think it was infected, but the University wants independent confirmation before they will put it back online. Bob told you that the machine was just being installed; there were hardly any files on it except for his own account 'bob' and some student accounts 'eric', 'kevin', 'peter' and 'takeda'.

Was the server compromised? If so, how? If not, what happened? What needs to happen before the system is put back into service?

Your assignment for each act is the same.

### Act II: The Missing Numbers

Your cousin Jimmy down at the precinct thought he'd throw you a bone so he sent you this case. It seems like a local punk broke into a server at Yoyodyne Defense, stole a protected spreadsheet chock full of secret numbers, and got caught trying to fence it to an undercover cop via Myspace. He says he got the spreadsheet "from a friend" but the story doesn't check out. The boys at the station seized his computer with an outstanding warrant for "Second Degree Music Piracy" but didn't find any evidence on it, so if they're going to get him for more than "Possession of Stolen Numbers," they need reasonable proof that he actually did it.

Yoyodyne graciously sent a disk image of the server that had the file on it for you to examine. The only other thing they know is that the kid's IP address at home was 207.92.30.41 at the time of the heist.

They know the file was stolen, but they need to know *precisely* how and whether this kid is responsible. Finally, Yoyodyne wants your professional recommendation as to what they need to do before putting the server back into production.

Your assignment for each act is the same.

## Act III: The Wealthy Individual

When you got back after lunch, there was already someone in the office. You could tell from the tux, tails, and British accent that he was a butler, but before you could offer him a drink he handed you the biggest diamond-encrusted hard drive you'd ever seen. Now, it's a fundamental truth that money attracts crooks like Martians attract germs, so you weren't exactly surprised when he told you that his employer's computer had recently been broken into. What *was* surprising was the M.O. -- the crooks deleted the boss' files to show they were serious, and then they encrypted his swiss bank account access codes and held the decryption keys ransom for 1 megabucks.

The boss needs the access codes to do business, but he doesn't want to cave in, either. At this point, he doesn't care about his computer, and he doesn't particularly care how they got in (although he'll give you some extra dough if you figure out who was responsible). He's just hoping that you can find a way to decrypt the codes so he doesn't have to pay those lowlifes. If you can do it, he'll give you a cut of the 2^20 dollars... and that'll keep you in patty melts at Norm's for a long time.

*Hint:* A few of these bank codes are *really* hard to find. Don't sweat it if you can't find every one -- 6 out of 8 is worth full marks -- but of course you'll get some extra credit if you collect 'em all.

Your assignment for each act is the same.

## Your Assignment

Your assignment for each act is the same: investigate the computer systems and develop a recommendation for three clients based on your investigations. The clients would each like a 1-2 page "post mortem" report on their computers detailing all relevant discoveries, including:

1. whether you think the server was indeed compromised
    - if so, how? if not, what *actually* happened?
    - give a blow-by-blow account if possible -- the more detail, the better!
2. whether you think the attacker accessed any sensitive information
3. your recovery of any meaningful data
4. a discussion of what should be done before returning the system to production
    - For example, is it good enough to delete the obvious files? Could the system be trojaned?
5. recommendations as to how they can keep this from happening again
6. an estimate on how long this assignment took you

In order to do this, you'll be inspecting filesystem images. You'll probably want to start by looking at the log files in /var/log on the images. See the setup section for some important lab-specific instructions. This assignment illuminates that computer forensics is sometimes "guesswork" -- if there was someone who knew exactly what happened, you probably wouldn't be there. In real life, some pieces of evidence make certain things obvious, while other pieces of evidence open or close possibilities. In the end, what is important is that your report explains what you *found*, and only from there does it attempt to describe what may have happened. There should be enough evidence in each exercise to make each scenario clear, but if there are things you don't know or can't discover, say what they are, and explain what your thoughts are.

**Please note:** Your report should be representative of something you would return to a client who has contracted you for this work. Be thorough and explain clearly what happened in a format suitable for delivering to a client.

### What can go wrong

- **Undelete Early, Undelete Often!** Make sure you perform undeletion as soon as you set up the loopback device -- if you mount the disk and do any work on it -- including reading files -- you may eliminate important information! (Why?) You can always delete your disk images and rerun loadimage.sh to get "clean" versions of the files.
- **Pathnames are confusing!** Be careful! It is tempting to type `cd /var/log` -- but `/var/log` is where `workbench` keeps its logs -- your forensic logs will probably be located at `/images/sda1/var/logs` -- or wherever you mount them. Make sure you keep your directories straight or you might find "evidence" that DETER has infiltrated your clients' computers!
- **Use Built In Search Tools!** Tools like `less`, `vim`, and `hexedit` have searching facilities, where you can enter string searches. For filesystem searching, consider using `locate` or `find`. This could speed up your recovery process.
- **There Could Be Bugs!** Because these scenarios are obviously manufactured, it is possible that you will encounter something that looks like a "clue" but is actually an mistake from our creation of the lab. We've tried to eliminate all of these, but if you think something may be a "mistake", feel free to ask your instructor.

Good luck!

## Extra Credit

For extra credit in this lab, try using the `john` password cracker on the `/etc/shadow` file for Act 3. You'll receive extra credit for recovering user passwords. You'll also receive extra credit if you recover all 8 missing bank keys.

## Submission Instructions

Make a *single file* (.pdf, .doc, or .txt) containing all three memos and any relevant materials. Submit this file to your instructor.