Thomas Hiller

CIS480 Senior Project

Report

May 18, 2006

For years, “Law and Order” has been a television show filed with interesting characters and plot twist. A character that is becoming more and more common on the show is the computer lab technician that analyzes different suspects’ computers for evidence of a crime. This referred to as computer forensics. When most people hear the term 'forensics', especially when referred to law enforcement, most think of scientist using different techniques to recover DNA, fingerprints, or other physical evidence to be used against a suspect in a trial. Conceptually, computer forensics is no different, only the methods and the types of evidence. Because this topic is so fascinating, it was my chose to do my senior capstone at SUNY Potsdam on.

The originally proposed project was vastly different from the end result. The intent was to create scenarios similar to ones that may be encountered by law enforcement, by taking five hard drives (Three of known file systems, one NTFS, one FAT32, and one EXT2 and two with unknown files systems) and extract as much information out of them as if it was to be presented in a trial. Immediately there were problems. With no real trial, there is no scope for trying to gather evidence. I would end up with a scope too narrow or too broad. After reading through an issue of Communication of the ACM (Casey and Carrier), the project outline was rewritten. Topics like rootkits and timestamps, which are more interesting than simple file recovery, were necessary in my project to have a more complete view of the current state of computer forensics. To have systems that have been altered by these different ways of changing or hiding information, the aspect of, not only law enforcement, but also the suspect’s as well. The different topics covered in my project are the data preservation, timestamps, different methods of hiding data, and rootkits.

Before starting on different issues covered in this project, it should be stated what was not covered. Forensic analysis of networks was not covered, the project focused on local systems. The reason why not to cover network forensics was lack of time and resources and there is nowhere on campus where an in-depth network analysis could take place legally. Stenography and Cryptography are

not covered either. Both of these would be better studied as individual projects instead of being included with computer forensics. One of the largest issues not covered is that of legality. Law differs from area to area and over time. Unless reviewing every law in the entire world regarding computer crime, which is impossible, discussing legality would give a false view of computer forensics and the law.

The integrity of data is crucial to be entered as evidence, because of this; there is a growing market for forensic products. To ensure data is in it's originally form, there must be nothing written to the media. Many different hardware devices are available that completely remove the write enable line from the interface. These devices range from $100 dollars to many thousands of dollars. Once writing has been disables, the media can be read to create a drive image to be analyzed. Many different software titles are available that extract useful information from these images. The degree of effectiveness usually depends on the cost of the software, which can also cost thousands of dollars. There was no budget for this project. The Computer Science department had an IBM ThinkPad A22 laptop and a forty-gigabyte external USB hard drive for my use with this project. Because there were no funds to purchase the professional quality hardware or software, free alternatives were used. A software write lock was used to replace the hardware equivalent. The software lock is not fool proof and still has the potential of writing to the media. The software used was a collection of free programs that were not the same quality as the thousand dollar forensic software suites, but for the purposes of this project, they were good enough. Most of these free programs were found on a live bootable Linux distribution named Helix (e-fense). Each of the programs used will be discussed later.

The first step in the forensic analysis of a drive is preserving the data. Doing so requires taking a disk image. The one of the best free tools to do this is a rewrite of the Unix command 'dd'. 'dcfldd' was written for Department of Defense Computer Forensics Lab (Kornblum) and includes the ability to perform user defined degrees of MD5 hash verification. This is one of the programs found on the Helix distribution, as is a GUI interface named Automated image & restore, or AIR. There are instances when a system cannot be turned off or have down time of any other sort, for this a live forensic analysis must be

completed. In a live analysis, the major difference is an image must be taken of the physical memory. 'dd' can create such an image using 'if=\\.\PhysicalMemory' as the input parameter. A version of 'dd' exists for both Linux and Windows, both are included on the Helix distribution. The images just described are raw data dumps, but other formats exist. Each commercial forensic software maker has a proprietary file format and different free programs offer their own file formats, some data dumps and some compressed. These compressed images are smaller but have size limitations and it is difficult to recover if there are errors in the file (PyFlag). Because of these facts, uncompressed data images were used for this project.

Timestamps are used to keep track of when a file was created, when it was last modified, and when it was last access. Timestamps are changed for different reasons. Some files are time sensitive and are shared by different people or groups that span time zones. Changing timestamps for these people is crucial to keeping things organized. Other, more illegitimate reasons for changing timestamps include an intruder disguising that fact that they have viewed or altered a file. In Windows, there are many different programs that can be used to change timestamps. The program used in this project was Date Edit, from Ninotech (Ninotech). This program is easy to install with a clean interface and allows a user to change the any of the three timestamps for a file. In Linux and Unix, there is a command named 'touch'. 'touch' is a command line program that can change either the last accessed or last modified timestamps of a file but defaults to changing both. If the file specified does not exist, an empty file is created ('touch' man pages). Timestamps are stored in the file's metadata, because of this, there is not real way to detect timestamps. Only through logging can a timestamp change be detected. Logs are not always reliable though; these are usually altered to hide the tracks of an attacker.

The hiding of data can either be trivially easy or can be so complex the operating system works against itself. Starting with the trivially easy. Files may be hidden in plain sight by simply changing the extension of the file. Any other person browsing through a directory would probably over look the file, thinking it is something else. In one case used in this project, a Windows folder contained three JPEG

images in it, one of apples, one of bananas and one of oranges. Windows will create a thumbnail image of a file with a known image extension. The only thumbnail created was of 'oranges.jpg'. 'Apples.jpg' was changed to 'apples.sys', so Windows thought it is a system file and did not try to generate a thumbnail of it. 'Banana.jpg' has been renamed to 'Banana.txt', Windows believed it was a text file and again did not try to generate a thumbnail of it. Only the file extension and what it is associated with has been changed, the file itself has not. If opened with a program that can view images, the image appears, as it should. There are a few different ways to discover these hidden files. There are programs that search through a file system and look for different file types, regardless of file extension. One of which is 'Retriever' and is included on the Helix Linux distribution. Sometimes just instinct that can reveal a hidden file. If a file looks out of place or is too big or too small for the file type, it is may be something hidden. More complicated methods can be used to hide files. Starting with Windows ME edition, hidden operating system folders have provided ways to hide information. These folders do not appear when browsing directories. Though, showing hidden files and folders and unchecking 'Hide protected operating system files' in the 'Folder Options' menu can easily uncover these folders (Xtra Limited). One of the most devious ways to hide files is to attach it to another file using Alternate Data Streams, or ADS. ADS was developed for Macintosh computers to network with Windows NT machines with NTFS file systems. To create an ADS stream, one must only open a command line terminal and open the program they wish to create the file with followed the file that it is being attached to, a colon, and then the name of the hidden file. An example is:

notepad C:\windows\iexplorer.exe:hiddenmessage.txt

Now the file 'hiddenmessage.txt' is attached to 'iexplorer.exe', and cannot be seen by normal means. There is a free program that exist that finds open streams named LADS, but LADS only finds open streams and does not delete them. To do so, follow the following steps in a command line:

>ren *filename* temp.exe

>cat temp.exe > *filename*

->del temp.exe

(Cook). Other ways to hide data include stenography and cryptography, which were not covered in this project, and rootkits, which are discussed next.

Rootkits are an increasingly dangerous method of hiding information. A rootkit is a piece of software that runs on an operating system and gives privileges to a user that they normally should not have them. A rootkit not only can be used to give a user administrative power, but also hide a file or process, even from the operating system. There are three type of rootkits: kernel rootkit, which acts like a driver adding code to the operating system's kernel, a library rootkit, these filter information between the operating system and the user, and application rootkits, which are similar to library rootkits but work on the application level. The most famous case of roots was in 2005, when Sony/BMG music placed rootkits on music CD to prevent piracy; unknowingly to the public (Radiojon). One of the first groups of people many would think rootkits would be used by would be a virus and spyware writer. This is currently a problem, but this trend started more recently. Another person who may use rootkits are employees at companies that want to steal information through keyloggers and accessing protected data or hide data from coworkers and administrators. One possibility that may not have crossed your mind is the idea of a person running a rootkit on his or her own system. Why? One reason is to run cheat programs while playing online games. Because there is big money is some of these online worlds, some game companies scan running processes to look for these cheat programs. Rootkits are a way around this. For this project, the FU rootkit was used. This is a simple proof of concept rootkit, and is less malicious than most other rootkits. FU has the ability to hide process and load hidden drivers. A test complete was to open a program in the background, MS Paint was chosen. FU was run and told to hide the process id of MS paint. Once done, the program was still running and worked perfectly normal but once task manager was opened, it was nowhere to be found. Another test was to hide as many processes as possible; this included processed required by Windows. Though all of the hidden processes were still

running, Windows could not find then and crashed. This is a good demonstration of how much power rootkits can have over an operating system. Most malicious rootkits are intended for Windows NT based systems, but there are rootkits for Unix and Linux. Three rootkits were chosen to be tested on Linux: SucKIT, a kernel rootkits for Linux 2.4 versions, a proof of concept rootkit that is loaded through /dev/kmem, so a kernel does not need kernel module support, adore-ng, a kernel rootkit for Linux 2.4 and 2.6 and is different because it creates a kernel level Virtual File System (VFS) to hide files and processes, and enyelkm, the most up to date (as of April 25, 2006) version of the common Loadable Kernel Module, or LKM, type rootkit. After two distributions of Linux and five different kernels, none of these rootkits would either compile or load properly. Even if they worked, there is an underlying problem; root privileges are needed to load these rootkits and development tools are needed to compile them on the system. Though these are factors that help prevent Unix and Linux rootkits, they still are potential harmful. Rootkits are hard to detect by nature. There are different programs that exist, designed to detect and remove rootkits. This is a game of cat and mouse, every time a rootkit can be uncovered, a new exploit is found to hide it. The future of rootkits is frightening, a proof of concept rootkit has been created that can dwell within a virtual machine and then infect the host. This rootkit can hide itself in two different operating systems; this may make detection and removal next to impossible (Wichmann).

This project provided an eye opening experience to how technology can be exploited for illegitimate reasons. Hobby operating systems is the first major project I was going to enter, once out of school with more free time. The topics researched in this project give new viewpoints for security and use within these small operating systems. These are also lessons learned for various career paths. This project directly led to being hired as a person with background in computer forensics. I was told after giving the speech on my project that this is a rapidly growing area of computing. Other career paths could have also taken information from this project; network administrator is one of the most obvious. As an administrator, everyone must be considered as a potential risky to the integrity of the security of the network. This project reviles some of the potential tools one could use to compromise the system's

integrity. Overall, the lesson is nothing is absolute and to protect oneself, you must stay one step a head of the people who are after you.

Bibliography

Cook, Rick. (September 16, 2005). Alternate Data Streams: Threat or Menace?. Retrieved May 18, 2006, from http://www.awprofessional.com/articles/article.asp?p=413685&seqNum=1.

Carrier, Brian D.. (February 2006). Risk of Live Digital Forensic Analysis. Communications of the ACM, Volume 49 No. 2, 56-61.

Casey, Eoghan. (February 2006). Investigating Sophisticated Security Breaches. Communications of the ACM, Volume 49 No. 2, 48-54.

e-fense. (2005). Helix [Linux Distribution]. http://www.e-fense.com/helix/.

Kornblum, Jesse. (February 12, 2006). dcfldd. Retrieved May 18, 2006, from http://dcfldd.sourceforge.net/.

Ninotech. (2002). Date Edit 4.0 [computer software]. http://home.worldonline.dk/ninotech/

PyFlag. (July 5, 2005). Advanced Open Standard Forensics Format. Retrieved May 18, 2006, from http://pyflag.sourceforge.net/Documentation/articles/forensic_format.html.

Radiojon. (May 11, 2006). Rootkit - Wikipedia, the free encyclopedia. Retrieved May 18, 2006, from http://en.wikipedia.org/wiki/Rootkit.

Wichmann, Rainer. (2002). Linux Kernel Rootkits. Retrieved May 18, 2006, from http://la-samhna.de/library/rootkits/index.html.

Xtra Limited. (May 14, 2006). Help: How to Show System Files. Retrieved May 18, 2006, from http://www.xtra.co.nz/help/0,,4155-1916458,00.html