Proceedings of the 7th IEEE Workshop on Information Assurance, U.S. Military Academy, West Point, NY, 21-23 June 2006

# Foundations for Visual Forensic Analysis

Sheldon Teerlink, Robert F. Erbacher, *Member, IEEE*

***Abstract*—**Computer forensics is the preservation, analysis, and interpretation of computer data. It is a crucial tool in the arsenal of law enforcement investigators, national security analysts, and corporate computer emergency response teams. There is a need for software that aids investigators in locating data on hard drives left by persons committing illegal activities. Analysts use forensic techniques to analyze insider attacks on organizations and recover data hidden or deleted by disgruntled employees or attackers. Advanced software tools are needed to reduce the tedious efforts of forensic examiners, especially when searching large hard drives. This paper discusses the background, algorithms, fundamentals, and techniques intrinsic to the visual analysis of typical computer forensic data. In terms of the visualization technique itself we discuss a visualization techniques to represent file statistics such as file size, last access date, creation date, last modification date, owner, number of i-nodes for fragmentation, and file type. The user interface to this software allows file searching, pattern matching, and the display of file contents.

***Index Terms*—**Computer Forensics, Visualization, User Interfaces, Software Architecture, Algorithms

## I. Introduction

Computer forensics is the preservation, analysis, and interpretation of computer data. In a world where the number of crimes committed using computers is increasing rapidly, a need exists for advanced forensic software tools. These tools allow investigators to follow digital tracks left by persons committing illegal activities. Plain text documents, log files, or even system files may contain traces of this evidence. More technologically advanced criminals may even conceal information by deleting it, encrypting it, or embedding it inside another file. With the large amount of storage space available on modern hard drives, searching for a single file becomes tedious without the help of specialized forensic tools.

Using visualization techniques to display information about computer data can help forensic specialists direct their searches to suspicious files. Attempts to interpret mass amounts of data that is not correlated or meaningful can waste a great deal of time and require high levels of both patience and tolerance for error. A well quoted phrase, “a picture is worth a thousand words”, directly applies to the assumptions of this research. The human visual system has the ability to interpret and comprehend pictures, video, and charts much faster than reading a description of the same material. This is a result of the fact that the human brain performs some processing early in the chain of processing visual input.[5]. This is a result of the human visual system’s ability to examine graphics in parallel [4] but text only serially.

Using this concept of visual perception, the researchers developed a graphical user interface (GUI) that displays file information visually. A user is able to query a specific directory and see statistics such as file size, access date, creation date, modification date, owner, and file type. Pixel intensity and color represent file type data, with each pixel representing an individual file. By clicking on the display and traversing the associated menus, a user can obtain information about a suspect file in more detail.

Viewing information about multiple files and understanding the relationship between them aids in forensic analysis. The user interface for this software allows file searching, pattern matching, and display of file contents. Each of these options provides a deeper analysis of the data stored on the hard drive and results in a flexible tool for locating criminal evidence.

This paper presents background as to the need for advanced forensics tools, the developed visualization capabilities, and the results of initial user studies comparing the visualization with Linux-based command line investigations. In addition to merely discussing the techniques abstractly, the details of the algorithms for the visualization techniques are presented. Additionally, details as to how to configure the environment for user testing are provided. These details will greatly aid additional researcher in performing significant research.

## II. Data Hiding and Concealment

Most computers under investigation contain hidden data in one form or another [2]. It may be password protected, encrypted, compressed, renamed, placed in an unusual location, appended to another file, or may fail to show up in a directory listing because system programs were modified. Recent consumer hard drives are quite large, and when full, contain tens of thousands of files. An average size hard drive for a home user today is about 60GB. When servers and non-traditional computer users are considered, the amount of storage easily meets or exceeds 100GB. It is easy to imagine how daunting a task it is to comb such a large hard drive for evidence without the help of any special forensic tools.

Figure 1 shows the 20 largest file types on a typical hard drive. This data represents a typical home user’s Windows 2000 desktop machine and reveals the types of files occupying the most space. This Windows 2000 machine has a 40GB hard drive with 15GB of it in use. These 20 file types consume approximately seventy percent of the used storage. It is safe to assume that a full 100GB hard drive would have similar ratios to the ones found on our Windows machine; or as is more

Sheldon Teerlink, Access Data, Lindon, UT 84042. steerlink@accessdata.com
Robert F. Erbacher, Computer Science Department, Utah State University, Logan, UT 843212, Robert.Erbacher@usu.edu, Phone: 435-797-3291, Fax: 435-797-3265

likely a larger percentage of the files would consist of images and sound files. These images and sound files could be rapidly reduced, leaving essentially the same setup as we have identified..

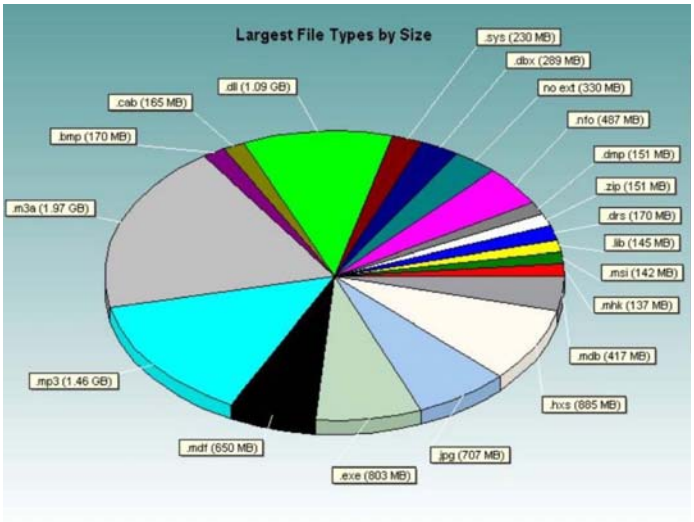


**Figure 1:** Pie chart showing the 20 largest file types on a typical Windows2000 machine, occupying 70% of the total storage.

Knowing the type and number of files stored on a hard drive can ease the search process. This knowledge helps an investigator determine how effective string matching will be or what applications he will need to open certain files. In the system mentioned above, the majority of the consumed storage space is in data files. Included are .m3a, .mdf, .mp3, .dbx, .zip, .mhk, .mdb, .jpg, and, .hxs. The next most common files include the shared libraries (.dll and .lib), followed by executables (.exe), files with no extension, and dump files (.dmp). Many of the unfamiliar file endings like .drs and .mhk are application specific. For example, .mhk files are data files for Riven, the predecessor to the game Myst. This knowledge during an investigation can speed the analysis process.

Many renowned security organizations such as the SysAdmin, Audit, Network, and Security (SANS) Institute and the Computer Emergency Response Team (CERT) Coordination Center offer guidelines and information about software that investigators can use to aid their search of hard drives and recovery of hidden data [3], [9]. Once files are found, it is a simple case of opening the file in a text editor. If the file is encrypted, a password cracking program called L0phtCrack (LC 5) or similar can be used to retrieve the secret key [1]. Criminals are clever, but if there is a way to hide the data without completely destroying it, there is a way to locate it. It is simply a matter of how much time it will take to reveal it. While commercial tools do exist, such as the Forensic Tool Kit, these tools require the analyst to search directories of recovered files or look at source code for files. Though powerful, these tools can be time consuming and frustrating to examiners who are unfamiliar with the data hiding techniques or file formats used in a particular case. The purpose of this research is to provide novel techniques to reduce this time requirement and improve the efficiency of forensic analysts.

## III. Software

### A. Capabilities

In this section, the capabilities of the implemented test system are presented. While the focus is primarily on the visual characteristics of the environment, the interactive metaphors incorporated into the environment are the contribution that will truly make it useful and allow an effective forensic exploration process.

The goal of this project is to locate suspect files on a large hard drive. A visualization display has been developed that renders data from a selected region of the file system residing on the hard drive of interest. Typically, this is a directory of files or a directory containing both files and subdirectories.

The visualization environment allows implementation of two proposed visualization techniques, a hierarchical visualization and a non-hierarchical visualization. An investigator or user of the software can switch between display methods, thus altering the visual representation of the selected hard drive region. Each visual display or representation is interactive and sensitive to mouse clicks. Selecting a file on the display by clicking on it with the mouse pops up text information, such as file type, file name, permissions, owner, group, access time, modify time, and creation time. Sliding the mouse over the display and clicking on files of interest allows fast and easy access to file information helpful in the investigation. Mouse and menu navigation allow the investigator to open files directly from the visualization display with an application of their choosing. When the user encounters a file with a name and type that do not match, he can open the file immediately and view its contents.

Our system also offers the ability to view the contents of archived or compressed files in the same way all other files are viewed. The idea here is that the user can select the archived or compressed file from the visualization and, in effect, zoom into the file to see what files are contained within. This feature is called archive file zooming. Each file in the compressed or archived file is colored according to a predetermined scheme.

If the user wishes to extract the files, it is possible directly from the visualization using the mouse and menu pop-ups. Files in the archived or compressed directory can be queried and opened. One additional coloring tag is used in both proposed visualization techniques to mark altered system files. The idea behind tagging altered system files is similar to the concept used by the commercial product Tripwire [6]. Tripwire allows a system administrator to create a baseline md5 digest for selected files including system commands and system libraries. At periodic intervals, say every day or week, the administrator compares the baseline against a current md5 digest to determine if any files have been altered. Altered files may be an indication of system compromise, to which the administrator can act accordingly.

Our system uses a database containing md5 digests for system files based on operating system (OS) and kernel versions. For example, the md5 digest of the ls system command for the Linux Redhat 9 OS running kernel version 2.4.20-8 is ’dbc1a18b2e447e0e0f7c139b1cc79454’. If this 128 bit key queried from the database does not match the md5

digest of the ls command on the investigated system, the block or rectangle is colored with a hatch pattern; see figure 2. Identifying altered system files can help the investigator to further direct the search for evidence.

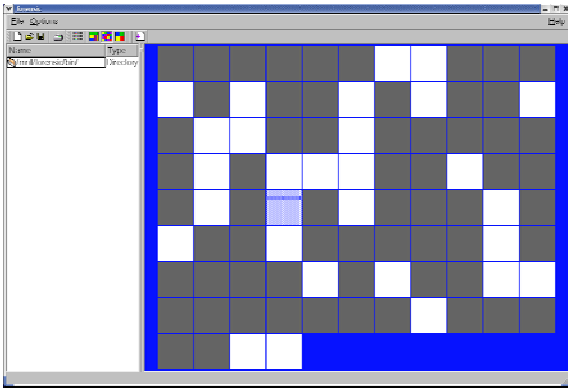

**Figure 2:** Square block view revealing an altered system file in the fourth column and fifth row

### *B. Visualization Methods*

For this research, two visualization techniques were deployed for the representation of file statistics relevant to forensic analysis. These techniques are designed around intrinsically different metaphors. The first metaphor represents file information without regard to directory structure or hierarchical information; the block diagram visualization. The detail is thrown away. Each file is simply represented as a small square box with its intensity controlled by a user selectable parameter. Using this technique is better for examining individual directories, but may lack necessary information for the forensic examiner. This technique does allow filtering based on selectable parameters, for instance figure 3 shows an example filtered on file size in which the intensity of the block is controlled by file size. By controlling parameters affecting the visual emphasis, analysts can quickly adapt the visualization to their current needs.

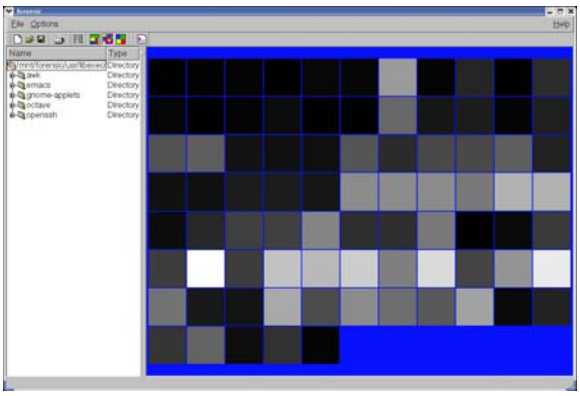

**Figure 3:** Square block diagram filtered on file size. The light-colored blocks are the larger files and the darker-colored blocks are the smaller.

The second visualization metaphor incorporates hierarchical information; namely Tree-maps, figure 4 [8]. This reduces the amount of information that can be represented, but incorporates critical information related to a file's position within the hierarchy that is lacking in the first metaphor.

Schneiderman [8] explains that tree-maps are a 2D space-filling algorithm for complex tree structures. They are designed for human perception by displaying the entire tree structure in one screen. Each file is a shaded box that adheres to a chosen coloring scheme that highlights file and directory boundaries. Box size is determined by two parameters; the size of the user selected display region and percentage of the selected directory the file occupies. Subdirectories are likewise displayed, subdividing each region until individual files alone are represented. Other file directory representations, such as Windows Explorer use nodes and edges rotated on their side, and always require scrolling up and down to view the complex structure. Tree-maps facilitate easy recognition of the largest files because they take up the most space in the 2D display. The method of using tree-maps to visualize data storage and directory structure greatly reduces the time it takes to locate large files in a tree structure that is several levels deep with tens of thousands of files.

**Figure 4:** Tree map view showing modified file recency.

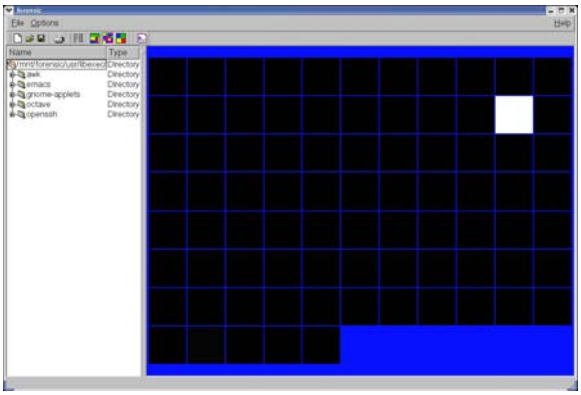

**Figure 5:** Square block diagram filtered on file modify time. The white block represents a file modified more recently than other files in /usr/libexec.

Tree-maps are primarily designed to emphasize large files. However, Schneiderman does point out that a user can drag a mouse over the display and click on a shaded box to query the system for the file name or other information. Such additions may enhance the usefulness of tree-maps, but stand-alone tree-maps for computer forensics contain many weaknesses. Small files and directories are hidden among larger files and may not even show up on the display. An investigator may be looking for a simple file on a massive hard drive (100+GB). If the file

is small, or if the disk contains numerous files, a single file will hardly stand out. For the researchers' purposes, stand-alone tree-maps require enhancement that provides the user with advanced filtering and display techniques.

With either of the aforementioned techniques, correct visual emphasis will make anomalous files stand out like a sore thumb, figure 5. While this example is demonstrated with file modifications time, the metaphor will apply similarly to any attribute available within the data set.

## IV. Architecture

We used Linux as the operating platform for the forensic software mainly because of its native support for needed functionality and support of numerous file systems. Our software does not run natively on the Windows operating system, but reads and processes FAT and NTFS partitions; though with emulation environments such as Cygwin it is likely possible to execute the tool on any platform. One of the key libraries to the development of the needed forensic capabilities was the magic library that allows the software to determine the type of a file from a database of nearly a thousand different formats. The magic library does this by reading a certain number of bytes from the file to extract a magic key, usually the first few bytes of the file. This magic key is used to determine if the file is of a known type such as a spreadsheet, a JPEG image, or a compressed file. Using magic and other built-in libraries greatly reduces development time.

The Qt [7] GUI API was selected for building the user interface. Qt is platform independent and compiles directly to the same level as the native windowing system. One of the huge advantages of Qt is obviously speed because communication is not being directed through an extra layer of abstraction. Qt supports OpenGL windows for advanced visualization as well as a number of data structures and file reading capabilities that make it much more than only a GUI API. When the time comes to add advanced visualization schemes the support for seamless OpenGL integration within our environment will greatly aid future development.

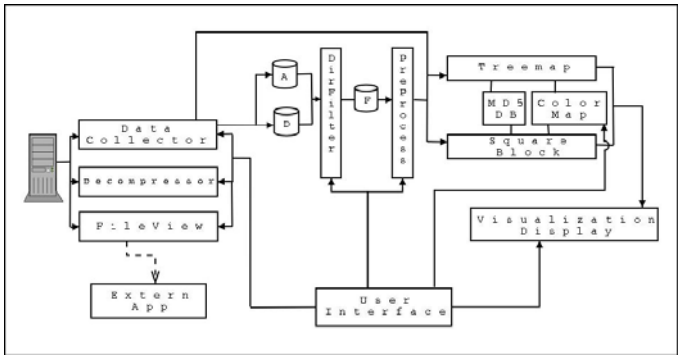


**Figure 6:** High-level view of the forensic software architecture.

Figure 6 shows a high-level view of the system components and their inter-connections. The DataCollector object reads file information from the hard drive image and prepares the data for visualization.

The library libmagic contains the algorithms for determining various file types, the object Stat collects time stamps on the files that have the most recent activity, and the object MD5 computes the md5 hash of each file. Time stamps are collected to aid the visualization rendering process and are discussed in detail in the algorithms section.

Once attribute data is acquired from the hard drive image, it is stored in a data structure of type ForensicFileInfo that contains the file type, md5 hash, and attribute information. All the slow file I/O operations are clustered at the beginning of the pipeline, so the user waits initially for the software to load but does not have to wait during analysis. Containers A and D store the ForensicFileInfo for every file in the image and decompressed file respectively. As the data travels through the pipeline, the user can filter the data so only a subset of the data is under analysis at any given time. For example, the user may only want to view the /usr/games directory and exclude everything else. Container F stores the ForensicFileInfo only for the directories under examination. Our main motivation for filtering is the improved speed we get by ignoring a large portion of the files. Object PreProcess prepares the ForensicFileInfo for visualization by using some simple algorithms, discussed in the next section, to convert certain attributes from the temporal domain to the spatial domain. As mentioned before, our methods use time-based file attributes to create filtered tree-maps and square block diagrams.

During the rendering process, objects Tree-map and SquareBlock make use of the user defined coloring schemes and cryptographic hash database (MD5DB) to create a meaningful visualization. Once the visualization is rendered to the screen, a user can query it for information and make requests to decompress or open files. Our current decompression engine only operates on gzip and tar files, but it could easily be swapped for a more comprehensive decompression module in the future. ForensicFileInfo obtained from a decompressed or nonarchived file is stored in container D. Storing unarchived data in a separate container allows the user to switch between visualizations generated from containers A and D without initializing the DataCollector to restore the overwritten data in A. When a request to open a file is made, it is handled by the FileView object. The FileView object verifies there is an appropriate application to open the file, forks a process, and hands it to the external application using the Linux command execl().

## V. Algorithms

Our first algorithm is contained in the Stat object of the DataCollector. Its sole responsibility is to record the most recent or maximum value access, modify, and creation time of the image files. Time stamps are unsigned integers representing the number of seconds since midnight on January 1, 1970. Obviously, a larger integer time stamp represents a more recent file. These values are used to render the visualizations based on the last time that file activity occurred. It may be several months before a computer is analyzed, and we do not want this fact showing up in the visualizations. We

want to view the computer system as it existed at time t, where t is the last time the computer was powered down.

| Seconds | Conversion | Significant Value |
|---|---|---|
| 60 | 1 min | 77 |
| 900 | 15 min | 59 |
| 1800 | 30 min | 45 |
| 3600 | 1 hour | 34 |
| 21600 | 6 hours | 26 |
| 86400 | 1 day | 20 |
| 345600 | 4 days | 15 |
| 604800 | 7 days | 11 |
| 1209600 | 14 days | 8 |
| 2419200 | 28 days | 6 |
| 7257600 | 3 months | 4 |
| 14515200 | 6 months | 3 |
| 29030400 | 1 year | 2 |
| 58060800 | 2 years | 1 |

**Table 1:** How time stamps are converted to significant values.

These maximum value time stamps are used by the PreProcess module to convert file times to the spatial domain. The idea behind the conversion is grouping time stamps into categories of temporal importance. Files are considered more important if they have recent time stamps. The older the time stamp, the less visible the file is in the visualizations. For example, let us consider three files in a directory. One was accessed two days ago, the second was accessed a month ago, and the third was accessed two years ago. We want the file accessed two days ago to show up most visibly in the visualization because it is most relevant. At the same time we do not want to deal with the difference between files accessed six months ago and those accessed seven months ago. It would be nice to group files accessed in the same significant period of time. This means all files accessed between nine months and one year ago would all map to the same significance value. Higher significance values are given to files accessed or modified more recently. Table 1 shows how the mapping of significant values works.

If we consider analysis of the modification time of files, the first column in the table gives the number of seconds since the file was modified from time t. Column two is the amount of time using different (more comprehensible) units, and column three is the significant value. It is easy to see that a file modified less than a minute ago receives a significant value of 77 while a file modified more than a year ago but less than two years receives a value of 2. These significant values are used to associate visual acuity, i.e., adjust the applied grayscale level based on the significance value. When preprocessing is run on file size, the file's size becomes its significant value. Thus, the size of a file will be the primary factor in identifying the files visual acuity within the visual display. Gray tones may vary in a range between 0 and 255, so, naturally, we scale the significance values within these bounds. Significant values are generated for as many distinct periods as necessary.

Currently there are fourteen significance periods. Starting from one, the size of each significant value is a third larger than its predecessor. These values are generated using the algorithm below. If two squares are seen side by side, one is easily distinguished from the other when it is a third larger or a third brighter; i.e. the 30% increase was chosen arbitrarily but designed to ensure ease of differentiation.

```
seed = 1;
for(i=0; i<TIME DOMAINS; i++)
     {
     next = ceil(seed * 1.3);
     seed = next;
     }
```

Drawing the square block visualization so the files remain square and large as possible, we use the following algorithm.

```
width<= height ? s = width : s = height;
while ((width/s)*(height/s)<n && s > 0 )
     s--;
if( s > 0 )
     {
     x_off = ( width % s ) / 2.0;
     total_x= ( width - (2*x_off) ) / s;
     total_y = ceil( n / total_x );
     y_off=(height-(s * total_y)) / 2.0;
     }
```

Dimensions of the drawing area are given by height and width. s is the length of one side of the square. Using a coordinate system centered in the top left corner that grows down and to the right, x off and y off are the x and y locations of where the first square is drawn. The remaining two variables, total x and total y, contain the number of square blocks in the x and y directions, respectively. In some cases, it may not be possible to draw all the files in the given real estate; hence, we verify this is not the case using the conditional statement if( s > 0 ).

## VI. DATA

For our user evaluation we used two different data sets for searching, one for each method. The goal of the data is twofold. First, we want a relatively substantial search space, and, second, we want to simulate a typical directory structure found on a home system. A search space was selected that was large enough to occupy an investigator's time but small enough to have a high probability of locating files; i.e. since we wished to perform tests with multiple subjects we had time constraints not presented to real analysts in typical scenarios. For these two reasons, a search space of 2GB was used. The Linux directory structure was selected to match the development environment; a Windows file system could have served as a substitute. Our test data was generated by creating four new user accounts and filling them with typical data found in most user accounts. This was achieved by logging in as the new user and operating the system for a time. Suspect files were then created and hidden in the user accounts, as well as other directories. These suspect files were accessed, modified, and handled in ways representative of a criminal trying to conceal them, such as name changing, directory relocation, and compressing. Next a mountable file system was created of the hard drive, for each case, as follows:

```
dd if=/dev/zero of=/data/caseX.image
count=4458220 bs=512
mke2fs /data/caseX.image
```

After creating an ext2 file system, it was mounted at /mnt/forensic using the mount command:

```
mount -o loop -t ext2
/data/caseX.image/mnt/forensic
```

With root privilege, selected directories were copied to the new file system including the suspect files. Options -p and -r were used to preserve time stamps and copy subdirectories.

Two user accounts were copied for each case. After copying 2GB's of system and other files, caseX.image was unmounted using the umount command and remounted read only.

```
umount /data/caseX.image
mount -o ro,loop -t ext2 /data/caseX.image
/mnt/forensic
```

Now that the new file system is mounted read only, it can be analyzed without modifying any of the data. It is noted here that the forensic process step of creating an exact image of the hard drive with forensic hardware was not used due to the lack of any court presentations.

Both data sets are nearly identical to each other, so one does not contain files less concealed than another, thus skewing the results of the experiment. Nevertheless, the placement of the hidden files varies between sets. Information about suspect files gleaned from one method cannot be transferred to the second. The data set is a scaled down version of a Linux file system. Directories used in include: `/bin`, `/sbin`, `/lost+found`, `/usr`, `/lib`, `/root`, `/dev`, `/home`, and `/tftpboot`. Each data set contains an altered system file (changed md5 value), a renamed media file, and a renamed office document. Table 2 shows the details of the placed files.

| Method | Original File | Hidden File | Attribute |
|---|---|---|---|
| LCS | /bin/ls | /bin/ls | md5 mismatch |
| LCS | marijuana.jpg | /lib/libdth.so.420 | renamed |
| LCS | delivery.xls | /home/escobar/backgammon.gz | renamed |
| FSS | /sbin/halt | /sbin/halt | md5 mismatch |
| FSS | hidout.jpg | /usr/games/phantom menace.avi | renamed |
| FSS | ospina.doc | /home/villabos/happy days.bmp | renamed |

**Table 2:** Locations of each hidden or altered file used in the different search methods; Linux Command Search (LCS) or Forensic Software Search (FSS).

## VII. Evaluation

A preliminary evaluation of the effectiveness of the developed techniques was conducted through a controlled, human-computer interaction experiment. In this experiment, a human subject was tasked with looking for three altered or hidden files on a hard drive using the two specified methods. When the subjects began each method, they were instructed to look for an unknown number of files, some of which are related to drug trafficking. The first method was to use traditional Linux commands such as ls, cd, grep, file, md5sum, stat, and find. The second method was to use the developed visualization techniques. During the study, each subject recorded three pieces of information: the time the study began, the discovery time and name of each suspect file, and the time the study ended. The ability to determine if one method was superior to another was anticipated as represented by the discovery of more files in less time.

Half the subjects began the experiment using the built-in system commands and finished using the visualization software. The other half of the subjects performed the same tests but in reverse order starting with the visualization software. By altering the methods the subjects started with, it was hoped that the researchers would be able to determine if the use of the first system affected the use of the second. Initially, six subjects were selected for this experiment, each of whom had general computer knowledge and varying levels of experience with the Linux operating system. Before participating in the study each subject filled out a pre-test questionnaire to categorize their abilities and identify how they would hide files or conceal evidence of a crime on their own computer and how they would search for hidden or concealed files on another's computer.

The experiments were run for 30 minutes for each of the two methods. The subjects were then asked to fill out a questionnaire to help the researchers understand which system was easier for them to use and which provided greater aid in locating hidden or altered files. The subjects were also allowed to write other comments about the experiment that could be used to make improvements to the software or visualization methods. Comparisons between these preliminary results were made to determine whether the visualization techniques developed would aid investigators in locating files faster than traditional UNIX commands.

### A. Results

After performing the user experiments, several analyses were performed to determine the effectiveness of the results and their impact. First, the efficiency and effectiveness of the techniques were compared. Second, the impact on the results of performing one experiment before the other was examined. Finally, the different search techniques employed by the test subjects and their effectiveness were compared and contrasted.

| Skill | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| directory traversal / view contents (cd, ls ) | x | x | x | x | x | x |
| vi, emacs | x | x | x | x | x | x |
| shell scripting | x | x | - | x | x | x |
| writing / compiling C/C++ | x | x | x | x | x | x |
| other software development (e.g. Java ) | x | - | - | x | - | x |
| regular expressions | x | x | x | x | x | x |
| security (e.g. iptables,tcpdump) | - | - | - | - | x | x |
| package / library management | - | - | - | - | x | x |
| kernel management | - | - | - | - | x | x |
| filesystem mounting | x | x | - | - | x | x |
| networking (e.g. Samba, NFS, ports) | x | - | - | - | x | x |
| grep | x | x | x | x | x | x |
| md5sum | x | x | x | - | x | x |
| stat | x | x | x | x | - | x |
| file | x | x | x | - | x | x |
| find | x | x | x | - | x | x |

**Table 3:** Raw skills of subjects who participated in our experiment.

### B. Tester's Abilities

Table 3 outlines the Linux abilities of each test subject. This

information was collected in the first survey before the subjects began any analysis. There is not strong evidence here to suggest that subjects with advanced Linux skills performed better in the experiment using either method. Subjects 3497 and 6324 have different skill sets, but both found all files using the forensic software. Subject 9121, who stated he possessed all the skills, only found one file with each method. We expected that subjects with advanced skills would be more successful locating the files, but this was not the case. However, it appears some of the subjects were able to make the skill to success transition where others were not.

Some subjects carried skills that attributed to their success that we were unable to measure. A point for future work is to identify a skill set or knowledge base that caters to success.

### C. *Time and Quantity*

The first question was to answer whether the forensic visualization techniques were capable of helping the subjects locate more hidden files in less time than using the simple command line search techniques. Ultimately, each subject using the forensic visualization technique was able to locate a number of files greater than or equal to the number they located using the Linux-based command search. Only one tester located the same number of files using both methods. All other subjects located an additional file using the forensic visualization techniques over the traditional Linux-based command search. The results illustrate that on average, 53 percent more files were located using the developed forensic visualization techniques. This suggests that organizing information in a way that supports clustering and outlier detection increases the probability of discovering suspect files, though this finding needs to be supported by further research with a larger number of subjects.

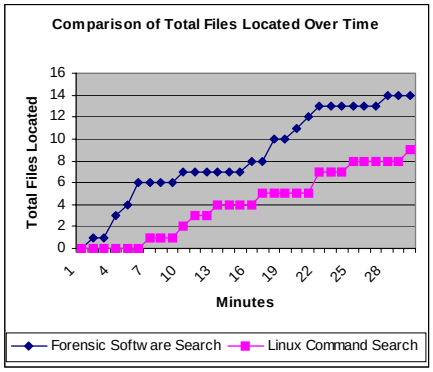


**Figure 7:** This plot shows the number of files located over time assuming all 6 subjects were searching simultaneously.

Once started, the subjects took an average of 13.7 minutes between files using the command search. Using the forensic visualization techniques, this value was greatly reduced to 8.8 minutes. A 35 percent reduction in time was realized using the forensic visualization techniques. Concerning time, another supporting statistic shows that the time to locate the first file was 57 percent faster using the forensic visualization techniques. This shows that the subjects were easily able to use the visualization techniques and achieve results in just a few minutes. Figure 7 shows the relationship between the number of files found over time assuming all the subjects were searching simultaneously. This plot shows that at any given time during the study, more files were identified using the visualizations than with the Linux-based command search.

An interesting point, amid all the data regarding the speed and success of the forensic visualization techniques, is that the renamed media file was never located using the Linux-based command search. File /lib/libdth.so.420 was a .jpg hiding among a sea of shared libraries. It could have been detected using the command 'file /lib/*', which would have listed out the type of each file in the directory. A search of the output would yield a single line stating that the file was really a JPEG image and not a shared library as its extension implied.

Contrast this process with the easy to view visualization created by the forensic software, essentially similar to figure 5. By filtering on modification time, the square block visualization scheme yields a field of mostly dark squares with a lone white square representing a file with recent activity (i.e., it has been recently modified). Clicking on the white square pops up a message box containing the file's details, including its name and type. In a real world scenario, this means time stamps can be used to identify suspect files on the premise that they have witnessed recent activity. This suggests the developed visualization techniques are an effective method for rapidly identifying outlier files.

### D. *Interaction of Methods*

Another issue that needed to be resolved was whether a human subject's performance depended on the order in which each method was applied. In essence, did the first method of searching affect the second, or are the results independent of order? This was determined by examining changes in number of files located and changes in mean time to locate a file.

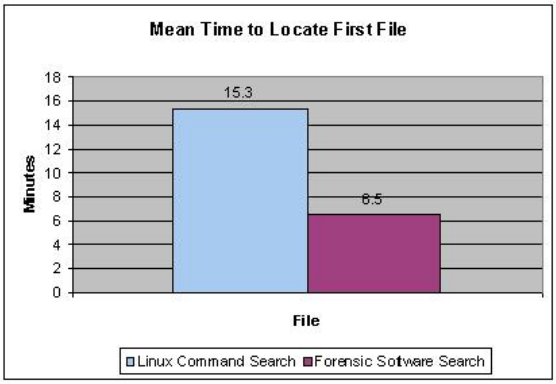


**Figure 8:** Mean time to locate the first file using different search techniques.

Figures 8 and 9 show the average number of minutes it takes to locate files using both methods. More specifically, figure 8 shows the mean time to locate a first file using each of the techniques. This mean value is a combined metric over all test subjects, whether they used the command-based test or the visualization first. No matter what order the different tests were performed it is clear the visualization resulted in an

enormous reduction in analysis time. Figure 9 shows the mean time to find an additional file from the previous one, averaged over all steps. Once started, the subjects took an average of 13.7 minutes between files using the command search. Using the forensic software, this value was greatly reduced to 8.8 minutes. A 35 percent reduction in time was realized using the forensic search software.

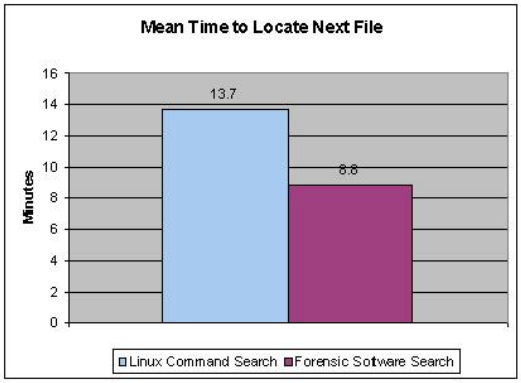


**Figure 9:** Mean time to locate consecutive files.

Figures 10 and 11 identify the number of files found and the mean time to locate them based on each of the methods. The plot in Figure 10 does not show any evidence to indicate that the first method had a strong effect on the second, either positively or negatively. However, Figure 11 shows that the time to locate a file using the second method is reduced, regardless of the technique used. While this preliminary data does not support an overwhelming argument about the interaction of each method, it does lead to the preliminary conclusion that higher performance is associated with the second technique applied by a particular user. Since both data sets were nearly identical, it is safe to suppose the subjects became increasingly more familiar with the directory structure between techniques, regardless of their order of application.

## VIII. Conclusion

In this paper, we discussed the critical needs, challenges, and background associated with computer forensics. In association with these challenges, we discuss the capabilities, algorithms, and techniques associated with our visualization environment for the visual resolution of the identified challenges. Finally, we discuss new results of user studies performed through the application of the developed visualization capabilities. The visualization display was developed and renders data from a selected region of the file system residing on the hard drive of interest.

The use of filtered tree-maps in computer forensics as proposed here is novel and offers many advantages over traditional tree-maps. Additionally, the use of square blocks in the manner proposed has not been examined. This system not only uses visualization to represent a file system, but also is specifically designed around the forensic process. The goal of this project is to locate suspect files on a large hard drive. To this end we put as much emphasis on the interaction techniques as on the visualization techniques.

There are several widely used computer forensics toolkits, namely: ILook [10], Encase [11], and Sleuthkit [12]. The extent of visualization incorporated into these tools is essentially explorer type interfaces. Thus, our work on visualization goes far beyond what these tools provide. However, these tools are very good at providing scripting and other low level analysis tools. Our goal is not to compete with these tools but rather, in the future, to integrate their results into the visualizations and thus make a more usable and effective set of capabilities.

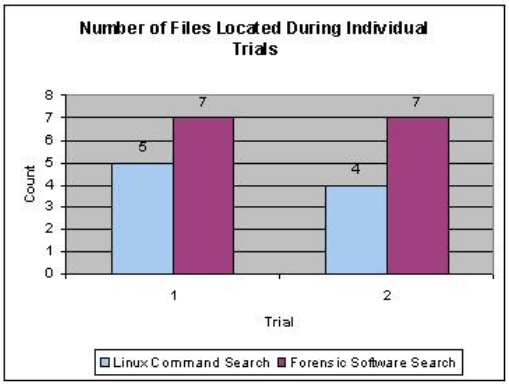


**Figure 10:** Number of files located for each trial and method.

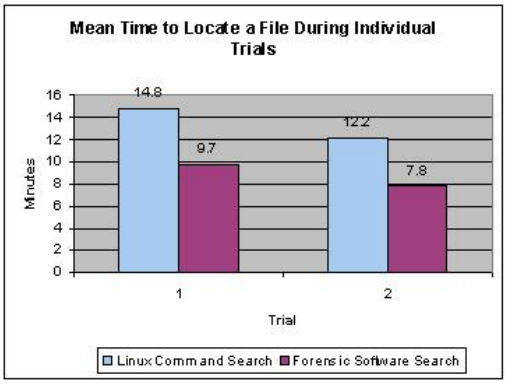


**Figure 11:** Mean time to locate consecutive files for each trial and method.

## IX. References

[1] AtStake Corporation. (2005). Available: http://www.atstake .com
[2] Casey, E. (2006). Investigating sophisticated security breaches. *Communication of the ACM, 49*(2), 48-55.
[3] Computer Emergency Response Team. (2005). Available: http://www.cert.org
[4] Kelsey, C. A. (1997). Detection of Vision Information. In W. R. Hendee & P.N.T. Wells (Eds.), *The perception of visual information (Second Edition)(* p 51). New York: Springer_Verlag.
[5] Neisser, U. Cognitive Psychology. New York: Appleton-Century-Crofts; 1967.
[6] Tripwire, Inc, Available: http://www.tripwire.com/resources/datasheets.cfm, September, 2004.
[7] Qt by trolltech. http://www.trolltech.com, October 2004.
[8] Schneiderman B., *Tree Visualization with Tree-Maps: 2-d Space-Filling Approach*, ACM Transactions on Graphics, vol. 11(1), January, 1992, pp. 92-99.
[9] SysAdmin, Audit, Networking, and Security (SANS) Institute. (2005). Available: http://www.sans.org
[10] http://www.ilook-forensics.org/iLookv8.html
[11] http://www.guidancesoftware.com/products/ef_index.asp
[12] http://www.sleuthkit.org/index.php