Master Thesis
Computer Science
Thesis no: MCS-2008-27
August 2008

Computer Forensics
Digital Evidence with Emphasis on Time

Jens Olsson

Department of
Computer Science
School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

This thesis is submitted to the Department of Computer Science, School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author(s):
Jens Olsson
Address: Gustaf Arnolds Gata 18B; 372 37 Ronneby; Sweden
E-mail: jens * rby.se (replace * with @ before sending e-mail)

University advisor(s):
Martin Boldt
Department of Computer Science

Department of
Computer Science
School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

ABSTRACT

Computer Forensics is mainly about investigating crimes where computers has been involved. There are many tools available to aid the investigator with this task. We have created a prototype of a completely new type of tool where all evidences are indexed by its time variable and plotted on a timeline. We believed that this way would make it easier and more intuitive to find coherent evidence and would make it faster to work with for the investigator. We have performed a user test where a group of people has evaluated our prototype tool against a modern commercial computer forensic tool and the results of this test are much better than we expected. The results show that users completed the task much faster and that the results were more correct. They also experienced that the prototype were more intuitive to use and that it was easier to find evidence that was coherent in time.

Keywords: cyber forensics, digital crime, e-fraud.

ACKNOWLEDGMENTS

First and foremost I would like to thanks Martin Boldt for being my supervisor during this work. Martin has contributed with many ideas and helped keeping me on track when I got buried into development details. Martin has also helped me with all kinds of things around the thesis writing.

My second thanks goes to my girlfriend, Anna-Karin Luiro for all support and understanding as I have spent many late nights on completing this work.

My third thanks goes to each of the people who participated in the user test sessions when trying out the prototype developed compared to the competition.

My last thanks goes to my cat, Tina, who have kept me company on my desk beside my computer during the majority of the late nights I have been writing and developing on this project.

CONTENTS

LIST OF FIGURES

INTRODUCTION

A. History of the digital society

"Computers have moved into every nook and cranny of our daily lives. Whether or not you personally know anything about it, you invoke computers when you make a bank withdrawal, when you buy groceries at the supermarket, and even when you drive your car" [10].

Meanwhile an increasing number of people are getting personal computers at home, in school and at work. About half of the Americans already own a computer [10]. In Sweden over 83% have access to the Internet at home [11] and around 77% have their own e-mail address [12]. The amount of Internet-connected people in the world over time is spectacular. In 1995, Internet had around 16 million users connected and in June 2007, 1173 million people were connected. In other words approximately 18% of the world's population was connected in June 2007 [13].

More advanced computer users and families own multiple computers and build their own home LAN (Local Area Network) to share files and their broadband connection to the Internet. Many use a broadband router with a built-in switch as a central point for the network [16] and many of these routers come with wireless capabilities, allowing computers to connect via Wireless LAN to the network and the Internet. There have been issues with the security of these wireless routers. Some of them have been completely open by default; enabling neighbors to access each other's network without a key. Key-based security on early devices has also proven to be weak and easy to compromise [15].

At companies it is common that every employee have a personal computer assigned to them. The company also often host their website and having central servers for handling the employees e-mail and sharing files within the company. Many companies also offer their employees the possibility to work from home, connecting from their home computer to the company network via a VPN connection (Virtual Private Network). A VPN connection provides the employees with an encrypted channel, preventing anyone who might listen to the traffic on the way between the employee's computer and the companies LAN [10][16].

Mobile phones are getting very common today. In year 2007 it was found that around 84% of the Americans owned a cell phone and around 6% owned a smart phone [14] (a smart phone is a cell phone with a large amount of features like, web browsing, word processing, e-mail etc.). Both standard feature phones and smart phones are getting very advanced. Today they behave almost like a personal computer offering a large variety of information services to its users. Many phones also offer built-in camera and sound recorder/player. By contrast to personal computers, mobile phones are completely wireless so one can access the Internet from anywhere. It is even possible to communicate with computers and servers while in movement, for example in a car or at a train [15].

Other devices that seem to become replaced by small computers are cameras and CD players. Today traditional cameras have almost been completely replaced with digital cameras. Digital cameras contain digital memory and can store images directly in computer format; it can often also store video. The memory used in the cameras is usually standard memory cards and can hold arbitrary computer files. The MP3 player seems to replace the CD player in the future. MP3 players often use memory cards as well, but it is also common that they use built-in memory. Like digital cameras most MP3 players can also store arbitrary files in addition to music [17].

Computers are introduced everywhere and within crime is no exception. While more and more things in our daily lives is getting handled by computers they naturally automatically get involved in many crimes, but they also introduce a new dimension to crime because skilled criminals can for example be located in Russia and compromise a computer in China over the Internet.

B. History of computer forensics

The forensic science begun around year 250 BC when Archimedes proved that a crown that was claimed to be of pure gold was in fact not entirely made of gold, he did this without damaging the crown [9]. Digital forensics by contrast is quite a newcomer and begun around 1970 when students at universities in protest destroyed some computers [2]. Simultaneously people begun to gain unauthorized access to mainframes (large computers serving many people). At the time computer crimes were hard to solve because existing laws did not seem applicable on them [2].

Today computers are involved in many crimes. Information technology is actively used by criminals to keep records, communicate and committing crime. It has even occurred that criminals have gained remote access to court computers and changed their criminal records [2]. Computer forensics is also getting useful in non-technical crimes. Since many people own a computer today, chances are vast that evidence for traditional crimes, for example a murderer or theft, can be found on a computer somewhere, maybe with the victim or the accused. Maybe the criminal has spoken to friends about it on instant messaging or via e-mail? Maybe the accused have talked to the victim over the Internet before the crime?

C. The challenges of computer forensics

To investigate computer crime, the examiner has to investigate the contents of computers involved to find evidence. There exist a variety of software specially designed to help the examiner with this task. There are many techniques that can be used by the criminal to hide information from the examiner that these programs have to defeat. Different software is the better in finding evidence hidden in different ways [5].

In year 1994, the hard-drive in a typical personal computer contained a maximum of 4.5 GB [6] (1.000.000.000 letters) of data (a hard-drive is the part inside a computer holding information for long time). This allowed a personal computer to contain the same amount of information as approximately 4500 books (we assume that a typical book is 500 pages and each page contains 2000 letters). Today a hard-drive in a typical computer can be up to 1000 GB allowing it to store around 1.000.000 books with the same assumptions [6].

This is by far more than you can find at many libraries. To make it even more complicated, any amount of computers may connect to each other in networks and via the Internet interchanging information. In networks we can also find servers with hard drive racks with storage capacity ten or hundreds or even thousands of times larger than typical home and office computers. These huge hard drives are then accessed and modified by a large number of people, locally, via an internal network or from the Internet.

Many users also own cell phones, handheld computers, USB sticks, USB hard drives, MP3 players, digital cameras and digital memory cards. All could possibly contain evidence that is key pieces of a forensic puzzle. Because of their size, some of them are as small as a thumbnail, they are very easy to hide and can therefore be the preferred storage location by criminals [2][8].

All this together, the evolution of digital storage space and new digital devices appearing everywhere in our lives, probably gives some feel about the enormous complexity and huge amount of information a computer forensics examiner is challenged with when investigating a computer crime today.

The examination software available is an indispensible toolbox for the examiner to manage the amount of information. For personal computers there are a number of large applications available. Most software for scanning personal computers has the ability to access information from a large variety of different systems, recover deleted files and search for encrypted files (files protected in a way that they are unreadable unless decrypted again with a key). In many applications there also exist special viewers, which enables for instance viewing of images as thumbnails, e-mail conversations in chronological order, show what documents have been opened lately and which web pages that have been visited [5].

D. Some major obstacles

There are some things that cause much trouble for the forensic examiner and the developers of digital forensic software. We describe some of them below.

1) Encryption

A large obstacle when examining hard drives is when the investigator encounters encrypted files. There are both weak and strong encryption solutions available. Using specially designed software for the task it is usually possible to decrypt the weak encryptions. Some of the weak encryptions need to be breached by finding the password. There are many ways of approaching this. One way is to try words that are of interest to the criminal. Another way is to scan the hard drive for cohesive phrases in the hope that the password was stored somewhere on the hard drive. If nothing else works the last solution is the have a computer test all possible passwords until one succeed. There are software specially designed for this and it is possible to create a system where the password-trying job can be distributed to a large array of computers (nodes) connected to a network. By doing this an almost unlimited password trying speed can be achieved. The only thing limiting the speed is the number of nodes that are connected to the network. One well-known cluster setup working in this way is called a Beowulf cluster [2].

Apart from weak encryption there are also strong encryptions available where one example is PGP. PGP is commonly used to encrypt e-mail and files saved on the computer. PGP is so strong so that a brute force attack is virtually impossible even with a huge Beowulf cluster. There may however be other ways to defeat this encryption. For example if the encrypted data is a file in a file system it is very possible that the plaintext of the file that was encrypted was just deleted from the system, leaving the actual data. The same data would then still be located in unencrypted form on the disk. Another solution would be to find the key, which might also be stored somewhere on the disk or possibly written down on paper [2].

2) Stenography

Stenography can be used to hide information in a way that it cannot easily be spotted for example hiding a document in an image file. An image file is usually built up based on an array of pixels (small colored dots) where each pixel has an eight-bit value specifying the amount of red, green and blue. There is software that can take an arbitrary file and dissolve it into binary bits and place them as the least significant bit of these red, green and blue values. This method will make a slight change of the color but it is so small that the human eye cannot see it. A digital image of 3 megapixels can by this method contain up to 1,125 megabytes of stenography data. There is special software available that automatically can help the criminals put this data into images however there are also special software that can help the forensics examiner to find this kind of files [34].

3) ATA Security Mode Feature Set

The majority of all ATA and Serial ATA hard drives today have a feature set called the security mode feature set. This is actually a number of ATA commands that can be sent to hard drives from the operating system or BIOS to enable security on the drive. It is possible to set a user password and a master password on the drive. The master password can be used to reset the user password while the user password protects all data on the drive. When the security features are enabled on an ATA drive its data will not be accessible after a power cycle and the user password will have to be supplied to unlock. There are two security modes to choose among, high and maximum. In the high mode both the user and master password can be used to gain access to the data whereas in maximum mode only the user password can be used. The master password can then only be used to make the disk working again by doing a complete secure wipe of the entire hard drive. The ATA security mode feature set is a very simple but quite powerful way of protecting the data of a hard drive. To retrieve the data the drive either have to be sent to a recovery center or special devices to unlock the drives are required. One device claimed to do the job for all drive vendors currently available is made by a company called Vogon and it is named Password Cracker POD [33].

E. Our work

The different software available has different ways if displaying the evidence found. Most of these applications have different views for different kinds of information. For example Forensics Toolkit have a special view for browsing e-mail, a special view for images displayed as thumbnails, a special view for file browsing etc. The increase of hard disk size as we have discussed earlier is obviously putting a very high pressure on the performance of the evidence browsing in these programs. It has to be intuitive and helpful to the examiner to find relevant evidence. We think that this will be the single most important aspect of a computer forensics tool in the future and we think that software can do a great deal of the manual work currently done to combine related evidence. We have concentrated on timestamps, which exist at least one in most computer files and objects. We think that if we can combine evidences that are coherent in time, the investigator might find evidence that happened at the same time much easier and even find evidence that would otherwise not be found at all. We are planning to create a prototype software tool that can scan a hard drive image for timestamps and list them on a graphical timeline where the examiner should be able to see what evidence of different times occur at the same time. We will also evaluate this tool against a modern computer forensics tool to see if it has any advantages compared to it. We will do this by performing a user test where test subjects will try to solve a fictive case with each tool.

BACKGROUND

Computer forensics, digital forensics and cyber forensics are all terms describing the task of investigating computer crime, all literature seem to agree on that. How investigations of computer crime are supposed to be conducted and which parts of the investigation are part of the computer forensics field is slightly different described in various literature.

In Incident Response & Computer Forensics they divide the process into seven steps: *Pre-incident preparation, detection of the incident, initial response, formulate response strategy, investigate the incident, reporting and resolution* [1].

In Digital Evidence and Computer Crime twelve steps are described. *Incident alert or accusation, assessment of worth, incident/crime scene protocols, identification of seizure, preservation, recovery, harvesting, reduction, organization and search, analysis, reporting, persuasion and testimony* [2].

In Cyber Forensics, computer forensics is described to deal with *the preservation, identification, extraction, and documentation of computer evidence.* [3].

In Computer and Intrusion Forensics four steps used by law enforcement are described. The first two steps are identifying the sources of digital evidence and to preserve them. The later two is to make a forensic analysis of the digital evidence (Extract, Process, Interpret) and to present the result including expert opinions and testimonies. If the first two steps are combined to one a simplified model called CFSAP (Computer forensics – Secure, Analyze, Present) is created [4].

CFSAP seems quite straightforward and on a level, high enough to be applicable on most organizations, we therefore base the rest of this chapter on it and below we will go into more detail on the three steps.

A. Computer Forensics – Secure, Analyze, Present (CFSAP)

1) Secure

This step is about recognizing what evidence has to be collected and to prepare it for investigation so that the damage or changes to the evidence will be minimal [4].

To recognize which evidence there are to collect often requires an in-depth understanding of the environment where the evidence is gathered as well as good computer skills [4].

The next step is to ensure the integrity of the evidence. The aim is usually to try to make an identical copy of the evidence so that it can be analyzed without destroying the original evidence. If the evidence is on a powered off system this can be as simple as removing the hard drive from the system and making a 1:1 copy to another hard drive. It can also be to gather and secure the original evidence hardware and to make sure this is done without altering any of its data [4].

To do this a special device called a write blocker is commonly used. Write blockers exist both as software and as hardware devices. A hardware write blocker is a small device that is connected between the HDD interface on the computer and the actual HDD. The write blocker analyzes commands sent from the HDD controller on the motherboard to the hard drive and filters all commands that instruct the drive to change its content. Hardware write-blockers are generally preferred over software write blockers. This is because software write-blockers easily can be bypassed by software. Hardware writes blockers are located outside and are not affected by software running on the computer [5].

Regardless of whether the original hardware has been gathered or an exact copy of it, the next step is to make more copies. These copies are usually verified by MD5 or SHA1 hash algorithm against the original to ensure their authenticity. The original is then locked away so that nobody has access to it [2].

2) Analyze

This step is composed of 3 tasks. The preparation, processing and interpretation of the data preserved [4].

During the preparation task the examiner make a copy of the master copy. The master copy is a copy of the original data, which is authenticated (for example verified against the original with a hash algorithm like MD5 or SHA1). The examiner makes a copy of the master copy so that it is possible to start over without access to the original if changes to the data have to be made during the examination [4].

During the processing task the goal is to find the relevant data and to extract it so that it can be presented as evidence. To do this the examiner recovers deleted data and searches for encrypted files, they also use pattern-matching techniques to search to find data corresponding to events. In this step a large variety of software is commonly used. A walkthrough of some of these tools is 0. Tools below[4].

During the interpretation task the actual human work begins. Here the examiner has to lay the puzzle, connecting pieces together and find out their meaning. Things like the reliability of the data and which people that can be connected to the data have to be considered [4].

Even though the analyze part is divided into different tasks it seem reasonable that much of the analyze work is done iteratively as each task can give input to another.

3) Present

The final step is to present the result of the examination, maybe in front of a jury or judge but it can also be to a board of a company, for example in a company-internal investigation. In this step an especially important skill the examiner need to possess is the ability to explain the technical aspects of the investigation in a way that it is easy to understand for people who does not know anything about the technology. It is also very important that all steps done to extract and find the evidence have been carefully documented so the procedures can be explained and reproduces if required [4].

B. Tools

There already exist a solid base of software for conducting computer forensic investigations. Following is a brief description of the most common tools.

- EnCase, made by Guidance Software is considered to be the largest digital investigation software on the market. It is written for Windows but can access a variety of file systems from a large amount of operating systems [5].
- Forensic Toolkit (FTK), made by AccessData is another forensic tool for Windows systems. FTK is also able to access different file systems but its strength lies in it's ability to search. It uses a sorted index to speed up searches and it contains numerous internal viewers for different file types [5].
- ILook Investigator, developed by IRS Criminal Investigation Division Electronic Crimes Program, is a tool running on Windows. ILook handles raw disk images as well as some widely used commercial formats. It enables the examiner to browse files in various categories. Encrypted files, deleted files, password protected files and files with invalid signature compared to its extension, just to name a few. It has the advantage of enabling the image creation of complicated systems such as servers with advanced RAID configurations etc. Unfortunately ILook is only licensed to law enforcement agencies, which make it impossible to get for companies making internal investigations etc. [7].
- ProDiscover, made by Pathways, is also a tool for windows. ProDiscover is also able to analyze a number of file systems. The license also contains the source code, so that investigators can identify exactly what the software is doing.
- SMART, a Linux based alternative. Developed by the same person who originally wrote EnCase (EnCase was called "Expert Witness" at the time). SMART makes use of the wide support for file systems on Linux and have advanced features like recovering deleted files, keyword search, and listing of image files [5].
- Sleuth kit/Autopsy is a collection of UNIX tools and commands and a graphical front end for these. Sleuth kid is also able to analyze numerous file systems and recover deleted files. It also has a feature where file accesses can listed chronologically [5].

PROBLEM DESCRIPTION

The computer forensics software available seems to have many different tools to browse for different kinds of information. This ought to make it necessary doing a lot of searching in different areas and to combine evidence with each other by hand, evidence of different types but which are coherent in time. With storage capacities as large as they are today in typical computers this will obviously be a very time consuming task. In many cases it is likely to be so time consuming that it cannot be done thoroughly within the law-enforcements budget for the case.

We think that finding innovative and different ways of displaying evidence to make it easier for examiners and to make it faster to conduct digital investigations. It is also likely that different ways to visualize can help examiners find different types or relations between evidence that would not have been found using other tools.

We have created a tool that extracts all timestamps from a disk image and lists them on a timeline for the examiner. This will help the examiner to find evidence located near each other in time, regardless of its evidence type. This could for example make it possible for the examiner to see e-mail messages, IM conversations and web browser history on the same timeline. The difficulty here is to find all timestamps from different evidence types and to implement readers for them in the software. Another difficulty is to create a user interface that is quick to use, intuitive and makes it possible to see the relation between the evidence gathered.

RESEARCH MOTIVATION

We want to concentrate on the time parameter of evidence because it is a variable that can be found in most evidence. It is like a least common denominator, with some exceptions of course, there are some evidence types that will not comply. For example e-mail messages and chat conversations (Windows Messenger, ICQ and Trillian etc) all contain a timestamp for each message sent and received. All files on most modern computers contain timestamps when the file was first created, when it was last changed and when it was last read [2]. Considering that different files are used for different things this is probably a great source of useful information about what happened and when it happened. It is also possible to get timestamps for last opened documents, executed software, visited web pages, all settings and data saved to the Windows registry etc. This is just to name a few examples and is just a small subset of all evidence types containing a timestamp that could be gathered. We think that if we combine all these evidence types in a database and sort it on time, we would create a new effective way of organizing and browsing computer evidence. This will also enable the examiner to choose an interesting time where maybe other interesting event in the investigation has occurred. The examiner is then able to see all events that happened on the particular computer (and possibly other computers if indexed in the same database) on and around that time. The examiner can browse back and forth from this time to find what happened right before and right after. We think that this will make it possible to find evidence faster and to easily spot evidence patterns that would not even have been found with the current methods. To prove this we have made a proof-of-concept application that scans a hard-drive for items and indexes them in a database with its timestamp as the key. The application will also show a visualization of the evidence that were indexed. Allowing the examiner to browse back and forth at the timeline.

RESEARCH QUESTIONS

We have defined the following research questions that we will try to answer with help of the prototype:

1. How can a computer program scanning a HDD for timestamps be implemented?

We will determine how a software program which purpose is to scan a computer hard-drive for timestamps could be designed. We will focus on how to handle the different data types as well as how to make it scalable so that support for more file types can be added in the future.

2. How can an application displaying a timeline with evidence be implemented?

We will investigate how a software program with the purpose of displaying the information gathered by the scanner on a timeline could be designed. Focus will be on how the data should be stored to allow for quick access as well as how the graphical presentation should be laid-out to allow for easy and fast browsing.

3. In what ways could such a tool be advantageous over modern ways of browsing evidence?

There are a large amount of computer forensic tools on the market today. We will test how a tool like this will perform compared against one of the modern tools available?

A. Evaluation of results

To evaluate if this prototype was a success, we have given test subjects the assignment to work on a computer we provided. On this computer they will be able to interact with the prototype software trying to solve a task given to them. They will all have a disk image provided and the prototype application ready to browse the evidence. The prototype application has already gathered all evidence it could find on the image and sorted it all by the timestamps. The test subjects will be divided into two groups. The test subjects in one group will use CyberForensics TimeLab, the prototype application that has been built in connection with this thesis. The test subjects in the other group will use Forensic Toolkit (FTK), which is a well-established computer forensics tool.

RESEARCH METHOD

First of all we have done a literature study to investigate where we can find timestamps in a windows system. We will also go into detail on the file formats and file systems so that we can understand how to interpret the information in these.

We have then developed a prototype scanner utility that will scan disk images for timestamps and output them in XML format so that they can easily be imported by other tools. We have also developed a prototype viewer that is able to load XML files generated by the scanner utility. After we have developed these two tools we are able to answer research questions one and two.

To answer research question three we have performed a user test with six test subjects who have tried to solve a fictive case with the prototype viewer and six test subjects have tried to solve the same case with a well-known professional computer forensics tool. The result of this user study has given us a good basis for answering research question number three.

TECHNICAL PRESTUDY

Before starting the actual implementation of the prototype software we have to make an in-depth investigation of which timestamps there are to collect, how we can find them and how we can decode them. This is a detailed technical prestudy, which will serve as a base for designing and implementing the actual evidence extraction part of the prototype.

We have focused the prototype on systems running Windows but log files gathered from other systems, mainly server systems, have also be used. Our goal has also been to make the design of the prototype so scalable so that it would be easy to implement support for other file systems and data formats in the future. To prepare for this and to give a broad overview we have also included detailed information about other file systems in the prestudy. Furthermore to some extent evidence from different files from other client and server systems are covered in the report although we have only implemented support for a small subset of them in the prototype.

First of all we have to identify the sources, where we can find evidence of which we can determine its time. Then we have to list all the kinds of evidence we can gather from the sources. We will put an emphasis on where the timestamp information can be found and in which way it is coded, because that is the core of what needs to be handled in the scanner prototype.

A. Files in different file systems

The majority of all operating systems use files in a file system to store data. There are many different brands of file systems available on most operating systems, some systems use unique, custom-built file systems while other operating systems use common and widely used file systems. The main cause of a file system is to organize data in a files and folders hierarchy. Additional to giving files paths in the hierarchy and a size, most file systems allow meta-data about each file to be read and written (meta-data is extra information, for example author, image size and sound file length). It is common that simple properties like time for last-changed, created and last-accessed, as well as if it is possible to read, write or execute different files is specified by the file system. It is also common for file systems to hold information about which access different users and groups should have on different files. Some file systems even allow extra data streams, additional to the content, to be contained within a file [5].

In Windows environments a file system called FAT (File Allocation Table) and another called NTFS (Originated from Windows NT) is mainly used for hard drives. Apart from in Windows FAT is also commonly used in embedded devices for example digital cameras, mobile phones and mp3 players. It is also a very common format for memory cards. FAT comes in different flavors: FAT12, FAT16 and FAT32. The number after FAT specifies how many bits are used to store unit addresses (we call the smallest amount of space possible to allocate a unit), hence a higher number will support larger storage volumes and smaller unit sizes. FAT12 is mainly used for floppy disks today whereas the others are used for hard drives and other larger storage devices. All files in a FAT file system contain the time when it was created, last modified and last read. The timestamps are stored as actual year (seven bits), month (four bits), day (five bits), hour (five bits), minute (six bits) and second (five bits). To store seconds at one second accuracy six bits would have been required so the five bits are only enough for saving the seconds with 2s accuracy. The year field starts counting at 1980. The accuracy is different for each time stamp. The last access timestamp only contains the date. The last modified timestamp contain both date and time and the creation time even include an extra byte giving an accuracy down to 1/100 second. An overview of this can be seen in Figure 1 [5][8].

```
//This is a conceptual overview, actual positioning vary

struct Date { //Used for create, modify and access
      unsigned int year       : 7; //Starts counting on 1980
      unsigned int month      : 4;
      unsigned int day        : 5;
};
struct Time { //Used for create and modify
      unsigned int hour : 5;
      unsigned int minute     : 6;
      unsigned int second     : 5;
};
struct TimeFiner { //Used for create only
      unsigned int hundredth  : 8;
};
```

Figure 1: Timestamp format in FAT16 and FAT32

NTFS is an alternative to FAT and is mainly used in internal hard drives. NTFS supports very large drives and exist in a number of versions, where the latter also natively support features like encryption end compression. It also contains support for disk quota (limiting users disk space) and enables an administrator to specify which directories different users and groups should have access to and what kind of access. NTFS like FAT stores information about when a file was created as well as last changed and read but it also stores an additional timestamp. This additional timestamp specifies when the metadata for the file was last changed [5][8]. NTFS seem to store timestamps using a standard Windows structure called FILETIME. FILETIME is a 64 bit unsigned integer counting the number of 1/10000000 seconds from 1601-01-01. We could not find any reference for this but according to our tests by comparing the time extracted via a disk image with the representation in Windows we are very certain that it is correct. A representation of this timestamp format can be seen in Figure 2.

```
struct WindowsTimestamp {
      unsigned int timestamp  : 64; //Number of seconds * 10000000
                                    //since 1601-01-01 00:00:00
};
```

Figure 2: Timestamp format in NTFS

On Unix system UFS1 and UFS2 can be found. These file systems exist in different modified and improved versions in many operating systems. Some examples of operating systems supporting this file system are: Apple OSX, Sun Solaris, OpenBSD, NetBSD, FreeBSD and HP-UX. Timestamps in UFS are composed of a 32 bit unsigned integers, counting the number of seconds since 1970-01-01 00:00:00 GMT [5]. For completeness we illustrate this timestamp in Figure 3.

On Linux systems today you can find EXT2 and EXT3 file systems [8]. We refer to both of them as just Ext. EXT3 is just EXT2 with a journal added. EXT3 is perfectly compatible with EXT2 however if an EXT3 volume is mounted with EXT2 drivers, the advanced journaling features will not be in effect. Ext contains 4 timestamps, last access, last modification, last change and deletion time. Timestamps in Ext is like in UFS composed of a 32 bit unsigned integer, counting the number of seconds since 1970-01-01 00:00:00 GMT [5].

```
struct UnixTimestamp {
      unsigned int timestamp  : 32; //Number of seconds since
                                    //1970-01-01 00:00:00 GMT
};
```

Figure 3: Timestamp format in Unix systems

On the Macintosh a file system called HFS (Hierarchical File System) and its sequel, HFS+ (Mac OS Extended) is used, the later introduced in OS 8.1 [8]. Since HFS is legacy we will not go into detail on what timestamps there are to find however we will look into HFS+. In HFS+ each timestamp is stored as a 32bit unsigned integer holding the number of seconds since 1904-01-01 00:00:00 GMT, The time is specified in GMT by contrast to original HFS who would specify the time in the computers local time zone. HFS+ contain timestamps for the creation, last content modification, last attribute modification, last accessed and when the file was last backed up. See Figure 4 for an illustration [18].

```
struct HFSPlusTimestamp {
      unsigned int timestamp  : 32; //Number of seconds since
                                    //1904-01-01 00:00:00 GMT
};
```

Figure 4: Timestamp format in HFS+

On the CD-ROM you usually find a file system called ISO9660. ISO9660 is a standard file system for CR-ROMs which implements the least common denominator for most file systems, leaving it to system developers to add their own extensions to the format to make it comply with the systems native file system (like special attributes and meta-data). In the ISO9660 format four different timestamps exist for each file. It contains timestamps for the creation and last modification of a file as well as the uncommon timestamps effective and expiration time. The effective and expiration timestamps specifies when a certain file can be first used and when it is obsolete and should not be used any more. The timestamps in ISO9660 is composed of seventeen bytes. The bytes contain information as following: number of years after 1900, month, day, hour, minute, second and finally offset from GMT in fifteen second intervals. By contrast to other timestamp format, this format saves all values except the time zone information in ASCII whereas the other formats use binary representation. A detailed description of the format can be seen in Figure 5 [19].

On the DVD a file system called UDF (Universal Disk Format) is used. This file system format can be seen as an extension of ISO9660, improving the performance and removing some of its limitations. It also enables the use of much larger media. The same format is used for expressing timestamps as in ISO9660 and the different timestamps stored in UDF are also the same except that the last modification timestamp does not exist in UDF [20].

```
struct ISO9660AndUDFTimestamp {
      char[4] year;
      char[2] month;
      char[2] day;
      char[2] hour;
      char[2] minute;
      char[2] second;
      char[2] hundredthSecond;
      char    gmtOffset;
};
```

Figure 5: Timestamp format in ISO9660 and UDF

B. Windows registry keys

On early windows systems all application settings were stored in a central text-file called win.ini located in the Windows folder. Later the application settings were moved to individual text files ending with .ini because of a 64K file limit of the win.ini file. The text-file system had many drawbacks where some examples are slow performance and administration issues [21].

In Windows 3.1 the Windows Registry was introduced. The registry is a so-called hierarchical database created to organize all system and application settings into a structured layout. The registry was build as a tree, structure almost like the directories in a file system. By contrast to a common file system, instead of containing files, each directory contains settings. At the time the complete registry was stored in a signle file called Reg.dat located in the Windows folder. In Windows 95, 98 and ME the registry was moved to two files, User.dat and System.dat located in the Windows folder. If user profiles were enabled every user also got their own User.dat file, containing personal settings, located in their profile folder. In Windows ME an additional Classes.dat file was added. In Windows NT, 2000, XP and Vista the registry was split into even more files, most stored in Windows\System32\Config. In the Config folder you can find a file named "Default", which stores the default settings, "SAM" which stores accounts securely, "SECURITY" which stores security related keys. Finally there are the "SOFTWARE" and "SYSTEM" files, which stores settings for all installed programs and the settings of the Windows system respectively. User specific settings are stored in a file called Ntuser.dat, located in the users profile directory [21][22][31].

The registry contains almost all settings in windows, from the control panel and information about connected hardware. It also contains application specific settings and data, for example recently open files, recently visited URLs. The windows registry is like a goldmine for the forensic examiner [21].

The logical structure of the registry is not mapped directly to the hive files, each hive file represent one or mores sub paths in the registry hierarchy. In the root of the registry there are 5 main categories. HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS and HKEY_CURRENT_CONFIG. Below each of these there are a tree of so-called Keys. Keys could somewhat be considered equivalent with folders in a file system, each having a name in the hierarchy. Each Key can contain sub keys, but keys can also contain value elements. A value element is a named object containing a value of a specific type. There are many value types in the registry. Some examples of value types are binary, dword and string [24].

The different registry files are generally called hives. These hive files are stored in a special binary format and held in memory although parts not recently used are paged out to disk to save memory. The binary file format is not easily readable by humans without a special reader or editor and by using the Windows Registry Editor not all meta-data are accessible. Something that makes the registry incredibly important in computer forensics is that each key has a last-write timestamp, which specifies when the key was last written. This is something that is not showed by the original registry editor provided by Windows.

This timestamp specifies the number of nanoseconds since 1601-01-01 00:00:00 and is in the same format as the FILETIME structure described in Figure 2 [22][23][31].

There is no official detailed information about the internal structures of the registry available. Although there are individuals and organizations that have reverse engineered the file format of the registry hives and sorted out most of how the majority of the data structures within the registry works. The Samba project and the Wine project are two large contributors to this success [26].

Since the registry in Windows NT/XP is what we have added support for in the prototype we have chosen to go a little bit deeper into this version of the Windows registry. In the rest of this chapter we will refer to the registry version used in Windows NT/XP as just "the registry". The registry is divided into blocks of 4096 bytes each and there are two kinds of blocks. There are the "regf" block and "hbin" blocks. The regf block is placed at position zero of the registry file and seems to be a header block specifying generic information about the registry hive. The regf block contains some important information. For example it contains an offset to the root entry and the size of the binary data. A detailed overview of the regf block can be seen in Figure 6 [31].

```
struct regfHeader {
      char[4]     signature;        //always "regf"
      int         unknown1 : 32;
      int         unknown2 : 32;    //Seem to be equal to unknown1
      WindowsTimestamp lastWriteTime;
      int         unknown3 : 32;    //Seem to be always 1?
      int         unknown4 : 32;    //Seem to be always 3?
      int         unknown5 : 32;    //Seem to be always 0?
      int         unknown6 : 32;    //Seem to be always 1?
      int         offsetRoot : 32;  //First key record
      int         dataBlockSize : 32;
      int         unknown7 : 32;    //Seem to be always 1?
      int         checksum : 32;    //Sim of all dwords from
                                    //0x0 to 0x1BF
};
```

Figure 6: Header of the regf block [31]

The binary data is located in another kind of blocks called hbin blocks. These blocks are always a multiple of 4096 bytes and each has a header containing among other things a pointer to the first and the next hbin block in the chain. Apart from the header each hbin block contain a list of "records". Each record begins with a size defined as a 16bit signed integer and right after the size follows data of a length specified by the size. If the size is negative the block should be considered as free space of "–record size" bytes [31].

```
struct hbinHeader {
      char[4]     signature;        //Always "hbin"
      int         offsetToFistBlock;
      int         offsetToNextBlock;
      char[16]    unknown;
      int         blockSize;
};
```

Figure 7: Header of the hbin block [31]

There are 3 record types that are important to us. There is the nk-record, which is actually representing a registry key. The structure contains the key name and a timestamp specifying when the key was last written to disk. It also contains a pointer to a list of sub-keys and a pointer to a value-list. The values are stored in another record called a vk-record. The vk-record contains a name and type of the value. It also contains a pointer and a length, which together describes where the data of the value is located [31].

```
struct nkRecord {
      char[2]     signature;  //Always "nk"
      int         keyType:16; //0x2C = rootkey, 0x20 = other key
      WindowsTimestamp lastWriteTime;
      int         unknown1          : 32;
      int         parentKeyOffset   : 32;
      int         numberOfSubkeys   : 32;
      int         subkeysLfRecordOffset : 32;
      int         numberOfValues    : 32;
      int         valueListOffset   : 32;
      int         skRecordOffset    : 32;
      int         classNameOffset   : 32;
      int         unknown2          : 32;
      int         nameLength        : 16;
      int         classNameLength   : 16;
      char[nameLength] keyName;
};
```

Figure 8: An nk-record [31]

```
struct valueList {
      int:32      value[];    //A list of value offsets
};
```

Figure 9: A value-list [31]

```
struct vkRecord {
      char[2]     signature;        //Always "vk"
      int         nameLength  : 16;
      int         dataLength  : 32;
      int         dataOffset  : 32;
      int         dataType    : 32;
      int         flags       : 16;
      int         unknown     : 16;
      char[nameLength] name;
};
```

Figure 10: A vk-record [31]

C. Workstation system data files and log files

There are many applications, services and functions on a workstation that creates log entries in various files and in the majority of the cases, each of these log entries come with a timestamp.

In a Windows NT/2000/XP/Vista system there are three interesting system log files called Appevent.evt, Secevent.evt and Sysevent.evt. These files are stored in the windows\system32\config directory. Appevent.evt contains log entries about applications usage. Secevent.evt contains log entries about security, for example logins and logoffs. Sysevent.evt contains system events such as shutdowns. These log files are stored in a

special binary format and cannot easily be viewed with a standard text viewer. Each log entry contains a date, time, username, computer name, an id of the event and a source. Each log event also has a category and a type. There is also a textual description of each event as well as additional binary data, which is optional. The timestamp in the log entries consist of a four-byte integer counting the number of seconds since 1970-01-01 00:00:00 GMT. The event logs are composed of a quite simple structure. They start with a header structure of typically 48 bytes and following the header are a number of log records adhering to another structure. These are of varying size but the size is specified within the beginning of the structure, which makes it possible to step through them all. [2][27][28][29].

In windows systems starting with Microsoft Windows 95B, it was possible to show folders as a list of image thumbnails. To achieve this with enough performance Microsoft stored thumbnails of all image files from a folder in a special hidden file residing in each folder called Thumbs.db. Apart from storing a thumbnail for each image, Thumbs.db both contains a list of all filenames as well as their modification dates. The thumbnails are stored in BMP apart from some minor changes in the header. This caching method is very interesting because it is common that image files that are deleted are still left in the cache file, allowing the examiner to find a copy of evidence that the criminal have deleted [27].

Another item that contains timestamps in windows systems are the shortcut files. Shortcuts in windows are used to make a link to an executable or a document from another folder. If executed, the shortcut will open the referred program or document. Shortcuts are defined with the special file extension .LNK. The shortcut files are using a binary file format and contains among other things, the complete path of the linked executable or document. It also contains a copy of the last modify, last access and create timestamp of the target file. Another important aspect of the shortcut files is that Windows holds a list of recently opened documents, called the “Recent”-folder; this folder is placed in the Windows folder on earlier versions of Windows. In later versions it is located in the users folder inside “Documents and Settings”. All windows links begin with the magic 32 bit number 0x4C. The timestamps are always located at offset 28, 36 and 44 of the file and they correspond to the creation time, modification time, and access time respectively. Each timestamp is 64 bits in size. [27][30]. We haven’t been able to find a reference of how these timestamps are encoded but as we have written earlier, in Windows there is a structure called FILETIME described in Figure 2, which is 64 bits. It would seem reasonable to believe that this type is also used in the LNK files because it is 64 bits. We tried this and according to our testing, if the timestamps are treated as FILETIME structures the date and time matches the target file.

D. Server log files

Servers are often a log file paradise, since servers often does not even have a monitor and are often monitored and configured remotely via remote command line. It is therefore very important with log files, to ease the identifying and solving of errors and miss configurations. Another very important aspect of log files is also to identify and track any attacks or break-ins that might occur.

Web server software is very common on servers and the web-server software usually writes to a number of log files. The well-known web-server software, Apache, uses two log files by default on most installations. Even though the Apache web server allows its administrator to configure it to create custom log files, there are two log files that are enabled by default. Those are the access and error log files, which both of them are stored in plain text. A new entry is written to the access log file each time a visitor requests a file from the server. The default log entry contains the IP address and often the host name of the visitor. It also contain which file was retrieved and if it was found or not. If the user have logged on via HTTP authentication, the username is also written to the log file. The time of the request is written inside square brackets “[” and “]” using the following format [Day/Month/Year:Hour:Minute:Second +/-TimeZone]. For example: [24/Nov/2007:18:22:22 +0100]. The error log lists page requests that have resulted in an

error. These errors could be “file not found” or server-side scripts that encounter errors or hangs. For some reason the error log uses a different timestamp format comparing to the access log. In the error log the timestamp is formatted as follows: “[Weekday Month Day Hour:Minute:Second Year]” Even though the time zone here is not specified, it seem to be specified in local time and not GMT [25].

Serving files for other computers is a very common task for servers. Many servers use FTP, Samba or NFS for this purpose. FTP often logs its file transfers to a file called xferlog. This log file commonly contains information about successful and failed login attempts. The amount logged and how the lines are formatted might be different in different servers although the default ftp server in both FreeBSD 6.2 and OpenBSD 4.0 write log entries that begin with a timestamp in the format “Month Day Hour:Minute:Second”, for example: “Oct 24 00:43:07”. The year is not included but it seems reasonable to assume that unless log entries from more than a year ago is needed, it would not cause any problems. And even though log entries from more than a year are needed it would be possible to find out the year as long as there are some traffic on the server generating log entries, hence all entries are in chronological order and if January is found after December it has to be a new year in between. The last year of the log file could be gathered from the files last modified timestamp from the file system [2],

Samba is commonly used in UNIX and clones for the purpose of sharing files with computers running Windows. The logging capabilities of Samba are configurable but on OpenBSD the default is to maintain a log file for each client. The default is also just to monitor user logons and user logoffs this can however easily be changed to write log entries for each file transferred [32].

Serving E-Mail is also a common task for many servers. In most UNIX systems and clones, Sendmail or Postfix is used to handle the transfer of mail from the client computer to the server as well as from the server to the recipient e-mail server. Both Sendmail and Postfix adds log entries to a file called maillog, usually stored in the /var/log directory. Each entry in this file begins with a timestamp in the format “Month Day Hour:Minute:Second” for example “Nov 24 16:34:06”. The maillog often contain information about e-mail sent from remote systems to local accounts. If the server also handles users outbound e-mail, which is common, the log file will include information about e-mails sent from the users to remote addresses. In addition to when a mail has been sent it is possible to determine the originator and recipient address of each message [1].

E. Communication evidence

Electronic mail clients can contain a huge amount e-mail messages, each can potentially be counted as time-bound evidence. We have already discussed mail servers and what logs they keep, we will now concentrate on the e-mail clients connecting to these servers. There are both web-based and client-based e-mail readers. Many e-mail services even combine the two ways of accessing e-mail by providing both as alternatives. Because of the large variety of web-mail services we will not cover them in this thesis although it would be very possible to extract evidence, containing timestamps, from most of these services as well. Traditional client-based e-mail readers use 3 standard protocols today. For downloading received e-mail from the e-mail server a protocol called POP3 is used, but it is being replaced with another protocol called IMAP. For transmitting e-mail messages another protocol called SMPT is used. SMTP is also used to forward mail between different e-mail servers.

When an E-mail is sent via for example SMTP, first of all the users client software sets the date and time of the e-mail transmission in the header before transmitting the e-mail. When the service providers server receives the e-mail, it adds from which IP address it was sent as well as a timestamp. It is then typically sent to the recipient's service provider's server. This server will also add a timestamp along with the IP address and DNS name of the senders service providers e-mail server. Finally the recipient connects to his/her service provider's server and retrieves the e-mail. The e-mail retrieved now contains IP address of the sender, the sender's service provider, the recipient's service provider and each of these together with timestamps. The servers involved in the communication also keep log files of the e-mail transmission typically from whom and to who and what time. This can be very useful to authenticate evidence found at a recipient computer, that an e-mail has actually been sent. It is also common that the e-mails travel thru some additional mail servers between the senders and recipients service provider. Those are also added to the e-mail message header in the same way as described above [8].

REALIZATION

In this chapter we will present an overview of the software design. We will discuss the Scanner and Viewer par separately and we will also discuss the interface that is used to interconnect the two parts.

A. Design Decisions

The prototype is divided into two main parts, completely separated from each other. These two parts are the Scanner and the Viewer. The purpose of the scanner is to scan a hard drive or an arbitrary amount of evidence recursively and identifying and storing the time stamps found. The scanner outputs XML code so that the viewer can read, index and display as a graphical timeline based on the timestamps found by the scanner. The scanner is written in the programming language Perl because of two reasons. First: it is good suited for text and data extraction, which is what this is all about, second: it is portable and can run on many different kinds of hardware and systems which would make it possible to keep the scanner while replacing the graphical viewer when porting to a new system. The viewer is built in C# because Windows is considered to be the largest operating system in use for the audience of the application. It is also a language that will give a short time to market because of its large amount of ready-made functionality.

B. The Scanner

The scanner is composed of four parts: the main executable, a data controller, a number of handlers for different file formats and utilities. At execution the scanner is supplied with a list of evidence files.

The main executable of the scanner walks through this list and tries to identify the content of each file. It does this by querying the handlers one by one asking them if they can handle the specific content. It continues until either a handler response true or when all handlers are queried.

Each handler has a function called validate and another function called Extract. The validate function is used by the main executable to query the handler whether it can handle a certain file. The extract function is called at first when the main executable know that the handler is suited for handling the specific format and will then go through the data structure searching for timestamps. It will also search for more data object that it should send back to the main executable for inclusion in the evidence list. This will make the scanner able to for example recurs through a whole hard drive, by first identifying the MBR, then the individual partitions, then the files on the partition and so on.

The handlers are closely coupled to source objects. A source object is a data source. All evidence is transformed into source objects which purpose is to maintain information about the location of the evidence. The source objects contain a chain of byte offsets and byte lengths as well as the origin. The origin can be either a file on the host system or another source object. This is one of the key concepts for allowing the recursive searching. This would make it possible to begin with a large HDD image, recurs into a partition with a NTFS file system, find a disk image in that file system containing a FAT32 file system and start going through the files. A detailed UML diagram of the design is available in Figure 11.

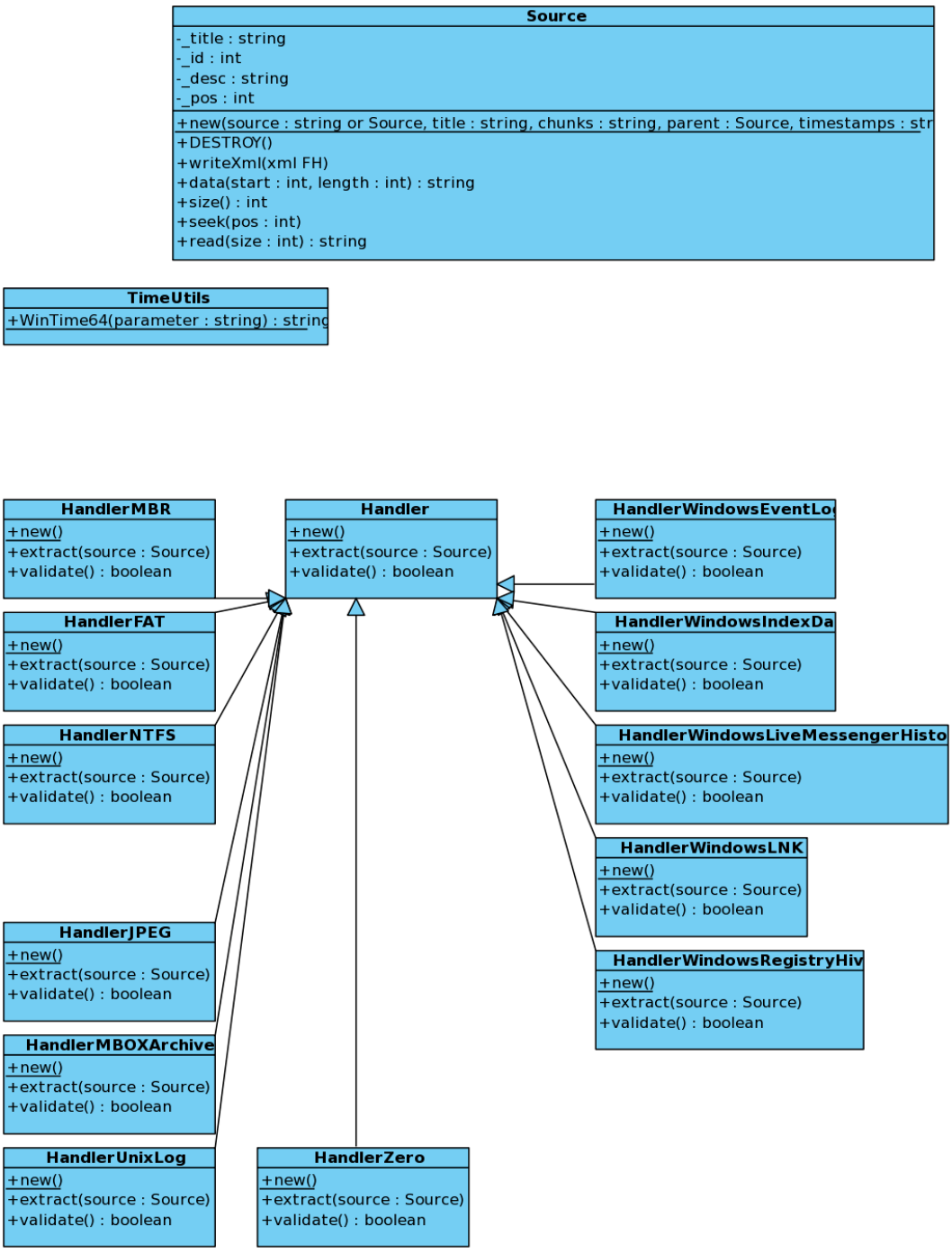

Figure 11: UML design of the scanner

The handlers that are made available in the prototype are:

FAT – Handles both FAT16 and FAT32 volumes. Extracting all files together with their Read, Access and Write timestamps.

JPEG – Handles image files in JPEG format. Extracts the dates contained in the EXIF part. This often contains the time of photo shooting and the time it was transferred to a computer.

MBOXArchive – Handles e-mail messages in MBOX format. This format is used in Unix machines as well as in Mozilla Thunderbird and Eudora to name a few graphical clients. The send timestamp, the receive timestamp and timestamps from the mail servers on the way between the sender and recipient are extracted. These in-the-middle

timestamps can be used for example to verify against the server logs that an e-mail message has really been sent. It is much more difficult for computer criminals to alter their traces on many servers than on just one.

MBR – This handles the MBR on a disk image. As the MBR does not contain any timestamps none could be extracted. It will however go through the partitions and extract these as source objects for other handlers to extract.

NTFS – Handles NTFS partitions. Extracting all files together with their Read, Access, Write and MFTChange timestamps.

WindowsEventLog – Handles windows system event logs, each log entry in these log files contain a time stamp and details about the event.

WindowsIndexDat – Handles the index.dat history files created by Microsoft Internet Explorer. It will extract timestamps together will URL for each visit. It might even extract some deleted entries if lucky.

WindowsLiveMessengerHistory – Handles the conversation history logs created by Windows Live Messenger. Each message has a timestamp and the message sent as well as sender and receiver.

WindowsLNK – Handles shortcuts in Windows. Each shortcut file contains a copy of the Read, Access and Write timestamps of the file pointed to by the shortcut file.

WindowsRegistryHive – Handles registry hives. Each registry key is extracted and the keys contain a LastWriteTime telling when it was last changed. These hive files can be found in the windows folder and in each users folder, but they are also commonly found in the system restore points often created by windows computers.

UnixLog – Unix has a large amount of log files in text format that begins with a date/time and ends with a description of the event. This handler parses those files.

```
<!ELEMENT EvidenceCollection (Evidence*)>

<!ELEMENT Evidence (Chunk+,Timestamp*,Data*)>
<!ATTLIST Evidence title CDATA #REQUIRED>
<!ATTLIST Evidence type CDATA #REQUIRED>
<!ATTLIST Evidence id CDATA #REQUIRED>
<!ATTLIST Evidence parent CDATA "">

<!ELEMENT Chunk EMPTY>
<!ATTLIST Chunk from CDATA #REQUIRED>
<!ATTLIST Chunk to CDATA #REQUIRED>

<!ELEMENT Timestamp EMPTY>
<!ATTLIST Timestamp type CDATA #REQUIRED>
<!ATTLIST Timestamp value CDATA #REQUIRED>

<!ELEMENT Data EMPTY>
<!ATTLIST Data name CDATA #REQUIRED>
<!ATTLIST Data value CDATA #REQUIRED>
```

Figure 12: Document Type Description (DTD) for scanner to viewer interface

C. The Scanner to Viewer interface

The interface between the scanner and the viewer is an XML file created by the viewer. The XML file is very straightforward, it simply contains a list of all <Evidence> tags where each contain a number of <Timestamp> tags describing all timestamps found in the evidence file. It also optionally contains a number of <Data> tags, which add special extracted information about the evidence. Further, a number of <Chunk> tags specifying where exactly in the parent evidence the evidence was found. This is specified by offset and length. Evidence added manually have a parent of "0" while evidence found by the software have the parent set to the evidence it was automatically found in. This makes it possible if one have the XML file and the image file, to extract the data part for particular evidence from the image file without actually having to embed the full evidence data into the XML file. In Figure 12 you can see the Document Type Definition (DTD) for the format used between the scanner and viewer and in Figure 13 you can see a small sample of how the evidence collection is coded.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl"
href="CyberForensicsTimeLab.xsl"?>
<!DOCTYPE EvidenceCollection SYSTEM "CyberForensicsTimeLab.dtd">
<!--
Created by CyberForensics TimeLab.
Copyright(C) 2008 Jens Olsson.
-->
<EvidenceCollection>
      <Evidence title="Volumes/NTFS.img" type="MBR" id="1"
parent="">
            <Chunk from="0" to="5368709120" />
      </Evidence>
      <Evidence title="Partition?0?" type="NTFS" id="2"
parent="1">
            <Chunk from="32256" to="5354657280" />
      </Evidence>
      <Evidence title="/AUTOEXEC.BAT" type="Zero" id="3"
parent="2">
            <Chunk from="1269053736" to="1269053736" />
            <Timestamp type="File's Master Record was modified"
value="2008-03-25 18:19:17.2031250" />
            <Timestamp type="File was created" value="2008-01-27
18:56:10.4900000" />
            <Timestamp type="File was last modified" value="2008-
01-27 18:56:12.0" />
            <Timestamp type="File was last accessed" value="2008-
01-26 23:00:00.0" />
      </Evidence>
      <Evidence title="/boot.ini" type="Zero" id="4" parent="2">
            <Chunk from="1268070688" to="1268070882" />
            <Timestamp type="File's Master Record was modified"
value="2008-03-25 18:19:17.2031250" />
            <Timestamp type="File was created" value="2008-01-27
18:44:29.6500000" />
            <Timestamp type="File was last modified" value="2008-
01-27 18:50:24.0" />
            <Timestamp type="File was last accessed" value="2008-
05-08 20:24:54.5937500" />
      </Evidence>
      <!-- More evidence are located here -->
</EvidenceCollection>
```

Figure 13: Sample XML code

D. The Viewer

The Prototype Viewer has been built using C#. It consists of a main user interface with a timeline and a detail view. The viewer has the ability to load the XML files created by the scanner and will plot these to a timeline. The timeline shows bars for different kinds of evidence. The higher the bar is, the more evidence is at the specific time. The viewer also contains a detail view where all timestamps are listed in a chronological list together with a short text describing each. The viewer also maintains the relationship between the timestamps and its source object so that it can easily be found of the timestamp is found. The viewer also maintains a tree of evidence in memory so that it can be possible for the investigator to find in which object a certain file has been found, and in which object that object has been found and so on.

The investigator is initially presented with a timeline containing all evidence found. The investigator can then mark interesting parts of the timeline and zoom in to get greater detail. The text-based list of evidence below is updated to reflect what is visible on the timeline view. When the investigator finds something interesting, different timestamps in the interesting time span can be browsed in the list view. When a timestamp is selected, the tree of evidence can be viewed. It is possible to see what other timestamps the evidence object contains as well as other properties that has been extracted. If the investigator needs to go into more detail there is a hex view available, showing the whole file in hexadecimal and character format. A screenshot of this can be seen in Figure 14 Hexadecimal View.

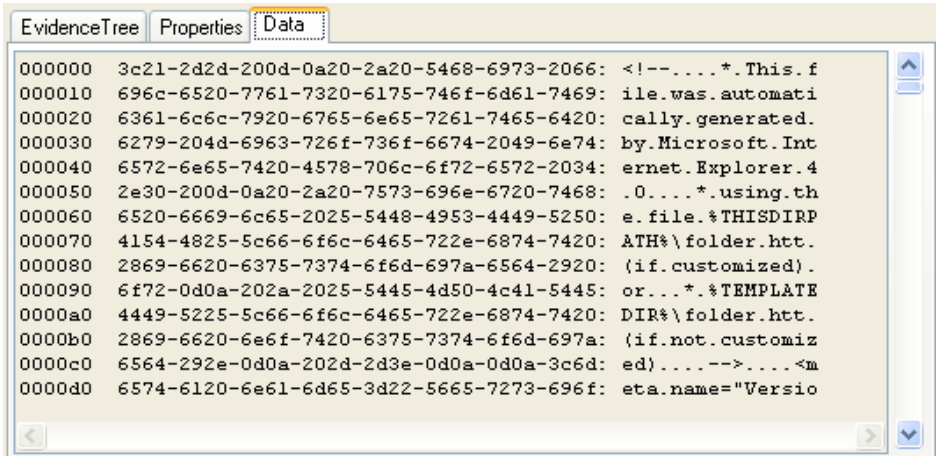

Figure 14 Hexadecimal View

To do this the viewer traverses the evidence tree and taking all data positions into account, when it arrives to the top evidence with not parent, it knows exactly which byte ranges that should be joined and displayed. This will for example work even if the evidence is contained in a NTFS disk which itself resides in an image inside a FAT32 disk image.

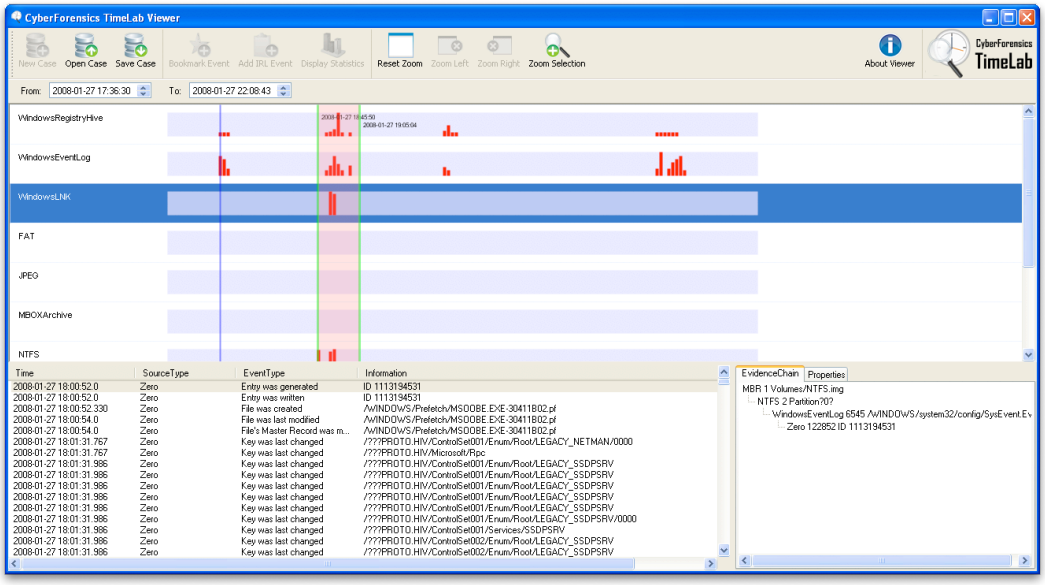

Figure 15 The Viewer

To make it efficient, the viewer maintains an ordered index of all timestamps. Each timestamp is then connected to an evidence object, which resides in anther ordered list with the evidence id as index. Each evidence object has a parent evidence building a tree in the memory. This way of storing the data makes it very efficient to create the timeline visible in Figure 15. To make this even more efficient a cache of the graphs is maintained. The cache is flushed each time the user zooms in or out. The list of evidence resides in a ListView control. To keep the speed and memory consumption to an acceptable level, the list of evidence could not be loaded right into the control. There is a special mode of the ListView control called virtual mode that is enabled. This mode does not keep any data inside the actual control. Instead there is a callback where the control can ask for the total number of items and another callback where the control can ask for an item with a specific id. This made it possible to link the control right into our indexed data structure and made the scrolling very fast.

A similar approach is used to make the hex editor work fast. Instead of loading the whole file the file size is retrieved. The scrollbar for the textbox will then load different fragments of the file and code it into hex on the fly instead of doing it once and let the control handle the scrolling itself. The latter choice not used here would make it impossible for example to display a 10GB file since it would require 10GB of RAM memory and probably reach other limits of what the control can handle.

PROTOTYPE EVALUATION

To evaluate if this way of categorizing and viewing evidence is advantageous comparing to traditional methods a user test have been made. Twelve experienced software developers have been given the task to investigate a fictional computer crime. The developers have received a disk image from the computer that the suspect have used to do their investigation on. They will also be provided with a number of questions that they are asked to find the answers to. The group members will have exactly 20 minutes to scan the image and to find answers to the questions asked.

The developers have been split at random into two groups with 6 test subjects in each; we name the groups The "TimeLab Investigators" and the "Traditional Investigators". The members of the groups know nothing of each other in the same group or in the other group. They have not even been told that they are part of a group.

The TimeLab Investigators will use CyberForensics TimeLab to scan the image for evidence and the Traditional Investigators will be provided the Forensic Toolkit Demo, which can be considered a very stable, reputable and well-used cyber forensics application. FTK is also considered to be state-of-the-art within computer forensic software.

A. Fictional scenario

A well-known cyber criminal, Evilyn Evans was tracked down by law enforcement officials when she was transferring a large amount of money from a stolen credit card at an Internet café in New York. The credit card was previously reported stolen and the credit card number was monitored in major payment services and banks, which made it possible for the law enforcement officials to get immediate notification when it was used. Unfortunately, when they arrived at the crime scene, she was long gone. The personnel at the Internet café recognized her from a photo the investigators had with them and they remembered which of the public Internet stations she had used. The investigators took the Internet station with them as evidence. One of the personnel could also tell the investigators that she used the computer between approximately 06:00 PM (18:00) and 06:30 PM (18:30) 2008-08-13. Initial investigations show that Evilyn Evans has created her own account on the computer system. The bank/payment service where the card had been used reported that she logged on to the bank at 2008-08-13 18:17:05 and suggested that you start looking there.

B. The disk image

The disk image has been prepared with VMWare server in the following way:

- Windows XP has been installed
- Installed Mozilla FireBird (E-Mail client)
- Installed MSN Messenger (IM Client)
- Updates are downloaded for Windows XP
- System is restarted
- System clock is set to 2008-08-13 18:00 AM
- Evilyn logs on
- Evilyn checks her e-mail where she receives a credit card number and instructions from bd@freenet.foo (MBX file generated)
- Evilyn hides the MBX file in a hard-to-find folder
- Evilyn connects to MSN where she receives an account number.
- Evilyn surfs to www.americasbigbank.com and transfers the money.
- Some additional MBX archives were added to the disk image so that there would be some e-mail to scan through; unfortunately MBX archives generated from Apple Mail did not get recognized in Forensic ToolKit, meaning that they only added to the complexity for the TimeLab Investigators.

C. The task

The below task was given to the test subjects together with the fictional scenario previously described:

You have received an image of the Internet station. It is now your job to find out answers to the following questions:

1. Which payment system / bank was used to transfer the money?
2. Which bank account did Evelyn Evans transfer the money to?
3. What was the stolen credit card number?
4. When exactly was the bank withdrawal done?
5. Are there any indications that the job was ordered by someone else, if so, who?
6. Could you identify any additional people involved in the crime, if so, who?

D. User survey

After the task has been completed, the people who have evaluated the prototype and FTK will be asked to answer a number of questions to provide some feedback of what is better in each program.

The test people will also be asked to answer a number of questions to determine their previous knowledge about computer forensics and their competence of computer science in general.

Questions about your previous computer knowledge

1. How many years of computer experience do you have?

( ) 0 years () 1-2 years ( ) 3-5 years ( ) 6-10 years ( ) 10+ years

2. How many programming languages have you used?

( ) None ( ) 1 ( ) 2-3 ( ) 4-6 ( ) 7 or more

3. How often do you view or edit a data file in a hexadecimal editor?

( ) Never ( ) once per month ( ) once per week ( ) 2-3 times / week ( ) every day

4. What is your previous experience to computer forensics?

( ) No experience ( ) Little experience ( ) Some experience ( ) Experienced ( ) Much experience

5. If you have ever used any computer forensic tool please name it or them below:

6. What screen resolution are you using on your computer?

7. If you have attended a computer related university program please specify it/them below:

Questions about the test

1. How difficult was it to solve the task with the software program?

( ) Very difficult ( ) Difficult ( ) Neutral ( ) Easy ( ) Very Easy

2. What did you think about the graphical user interface of the program?

( ) Very intuitive ( ) Moderately intuitive ( ) Neutral ( ) Fairly unintuitive ( ) Unintuitive

3. Was it easy to see coherence between the events that has happened in the target time span?

( ) Easy ( ) Fairly easy ( ) Neutral ( ) Difficult ( ) Very difficult

4. Do you have any suggestions on how this program could be made easier to use or more efficient? (Free-text)

E. Population of subjects

The population has been selected using the strategy of convenience sampling. All subjects were university students or former university students. To identify the population in more detail we have visualized some key properties in the diagrams below. In Figure 16 Years of computer experience and Figure 17 Used programming languages it is clearly visible that the two groups have a very similar computer experience which vouches for a fair comparison between the two software programs.

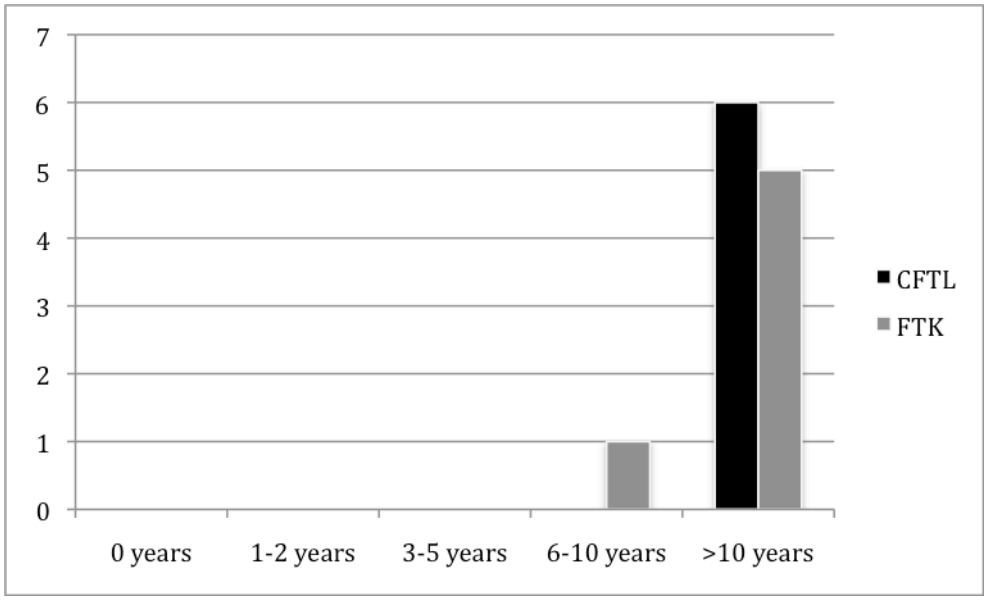

Figure 16 Years of computer experience

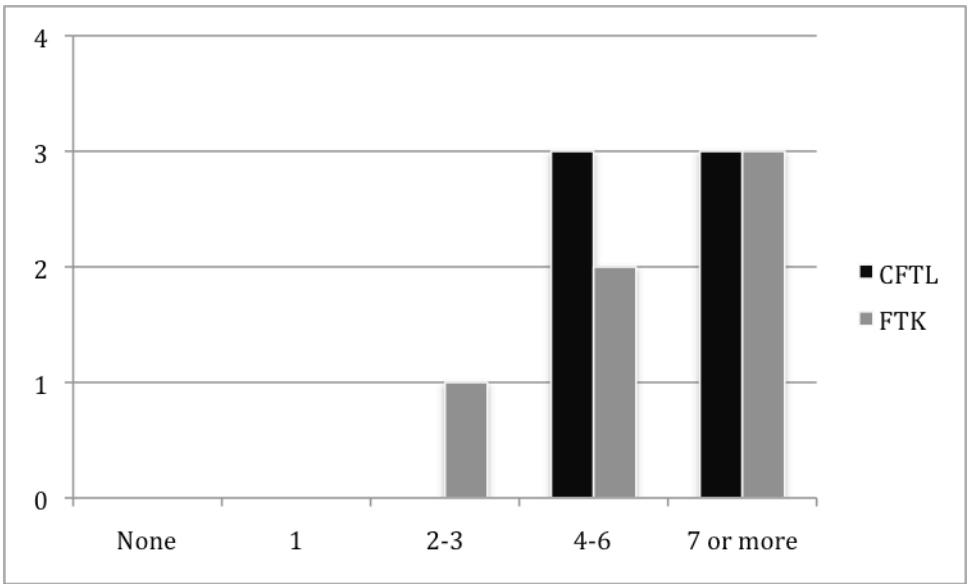

Figure 17 Used programming languages

We have also asked the test subjects about their previous knowledge about computer forensics and their day-to-day usage of hex editors. We think that this is a good indication of how experienced they are in the computer forensics field. In Figure 18 Usage of hexadecimal editors and Figure 19 Amount of experience in Computer Forensics you can see that also here, the groups are quite equal. A slightly larger use of hexadecimal editor in the TimeLab Investigators group can be seen though.

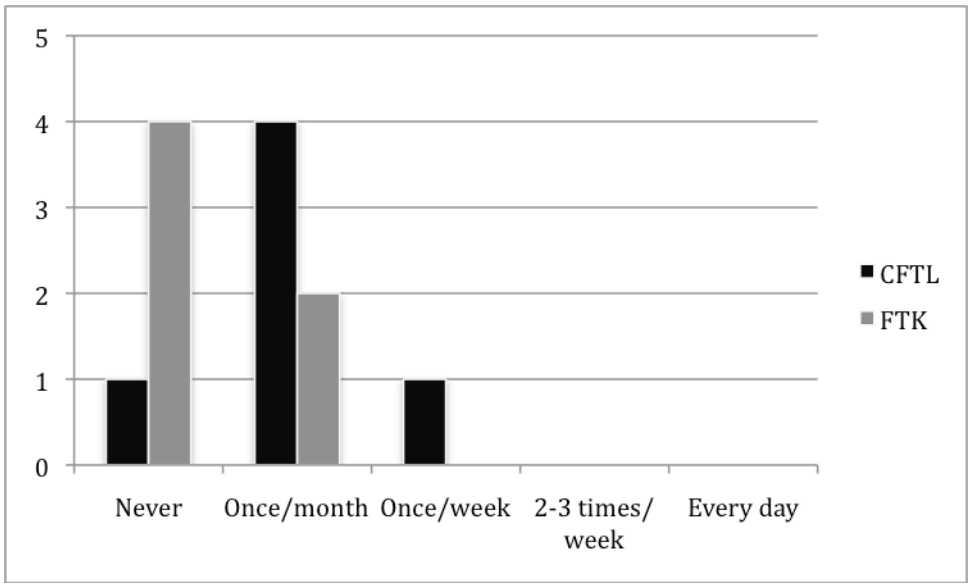

Figure 18 Usage of hexadecimal editors

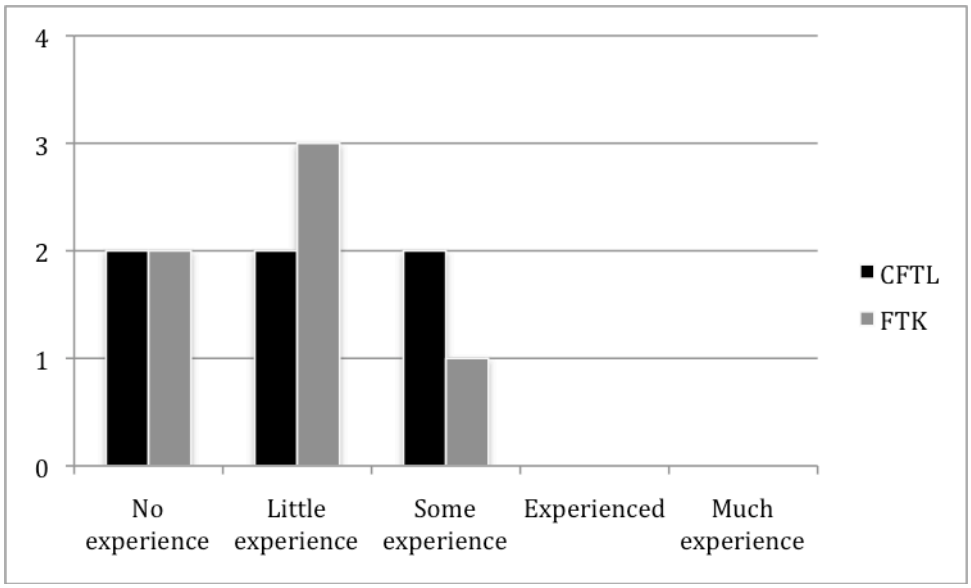

Figure 19 Amount of experience in Computer Forensics

PROTOTYPE EVALUATION RESULTS

The test was successfully completed with twelve test subjects divided equally between each group. The groups was not told that there was another group they were competing with, neither were them told that there was another program tested. The evaluation results

Based on comments from the test subjects the task did not seem to be very hard to solve and the majority answered correctly on all questions. Some test subjects did confuse the buyer and the boss in the case. This has however been treated as correct since it is a mistake in the understanding of the case and not in the software usage. The test subject has still found the relevant information with the tool provided, which is the important part. Two test subjects in both groups did the mistake so it cannot impact the results by giving advantage either tool. Some test subjects also confused the credit card number with the bank account number. This has also been accepted as correct for the same reason. This error was done only once and by the Traditional investigators group. In Figure 20 the amount of correct answers for the two utilities are visualized. The results for the two programs are very equal although a slightly better result can be seen for the TimeLab investigators. If the groups are seen as a whole, the Traditional investigators got a total of 31 correct answers while the TimeLab investigators got 35 correct answers.

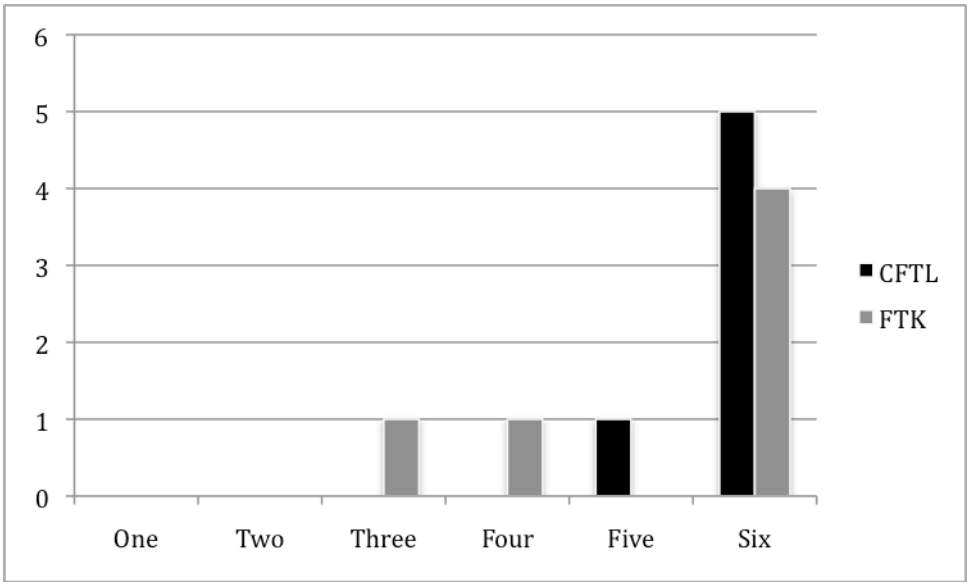

Figure 20 Number of correct answers

The time spent on solving the questions is a very interesting part of the result. Figure 21 shows that no time spent with FTK by any test subject is shorter than time spent by any test subject using CyberForensics TimeLab. This indicates that CyberForensics TimeLab is clearly faster than FTK when working with this type of case. The average time for solving the case in FTK is 45 minutes while the average time in CFTL is just 15 minutes. According to this the case was solved in CFTL in approximately one third of the time comparing with FTK.

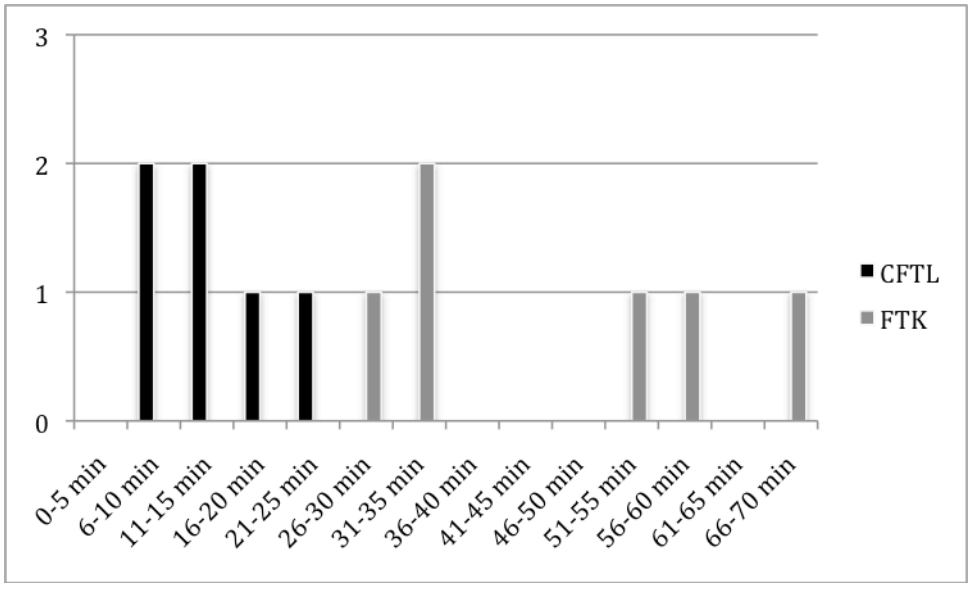

Figure 21 Time spent on solving the task

A. User integration

The test subjects have answered a number of questions on how they experienced the software they have been trying. They have been asked to value how difficult the task was to complete with the utility. The utilities graphical user interface is valued as well as how easy it is to see coherence between events occurring close to each other in time.

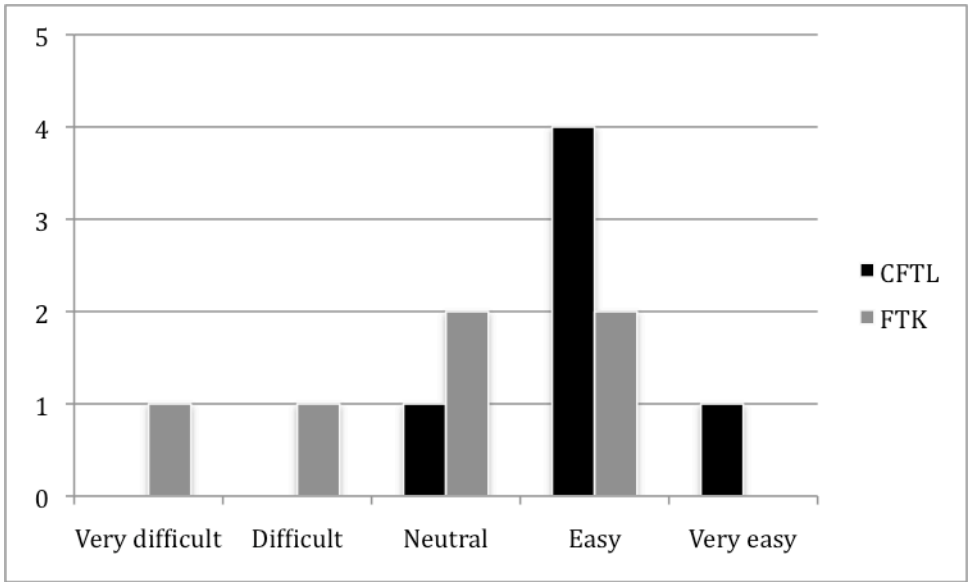

Figure 22 Difficulty of solving the task

In Figure 22 the test subjects have valued how difficult they thought the task was to solve using the utility provided to them. In the diagram it is possible to see that CyberForensics TimeLab is slightly more easy to use than Forensic Toolkit. If "Very difficult" is weighed as -2 and "Very easy" as +2 we get a comparable value for both tools. FTK receives -1 points while CFTL receives +6, which is a significant difference. This can also be connected with the time required to complete the task, which add to this as well. We can also see in Figure 23 that CyberForensics TimeLab is received as a little bit more intuitive. We weight this as well, treating "Very intuitive" as +2 and "Unintuitive" as -2. This gives FTK a value of 0 and CFTL a value of +2. It can also be read in the free text comments about CyberForensics TimeLab that the zoom function (using the left mouse button to mark and the right mouse button to make a selection) is

quite unintuitive and that it would be even more intuitive if this were fixed to use only the left mouse button (click and drag). In Figure 24 we can also see that CyberForensics TimeLab makes it easier to see coherence between events in a given time frame. This is probably because all events are placed on the same timeline view while different views are used for different kinds of evidence in Forensic Toolkit. A weighing here as well with "Easy" as +2 and "Very difficult" as -2 will give FTK -1 and CFTL +6.

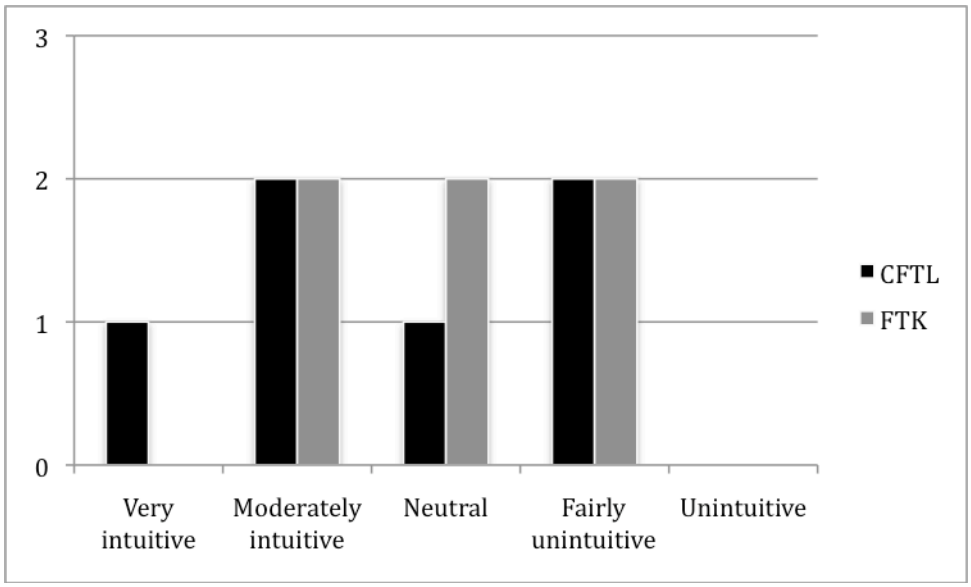

Figure 23 The graphical user interface

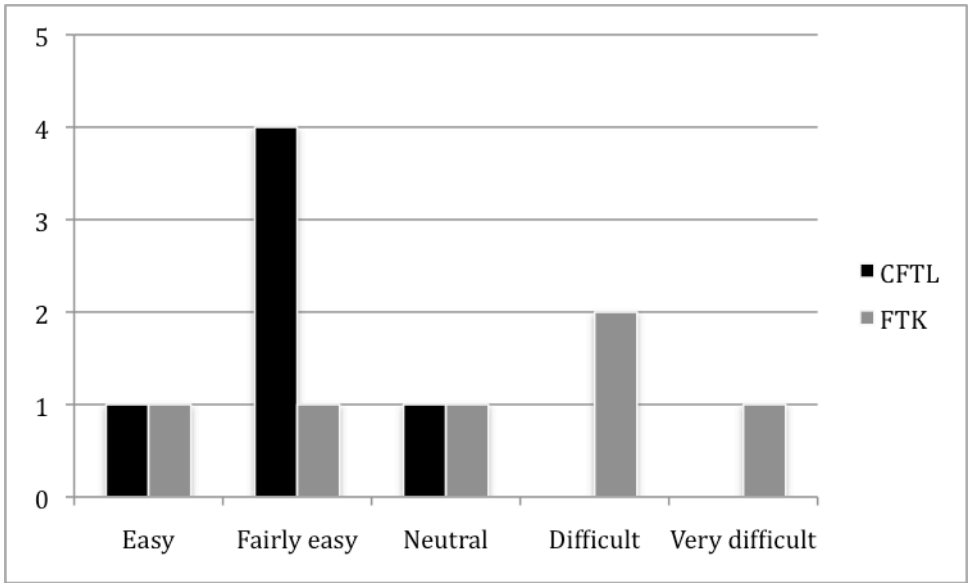

Figure 24 Difficulty of seeing coherence between events in the target time span

B. Improvement suggestions

Many of the test subjects commented on the zoom function, that it was quite unintuitive to use the left mouse button to mark the first selection position and the right mouse button for the selection length. This should probably be changed even though that using both buttons gives more control.

Many test subjects also mentioned that there were some issues with updating the evidence properties when selecting evidence. Currently there is a risk that information from the previously selected evidence is still displayed if the user does not update. This has however been a known issue and already been planned for fixing.

Some test subject also requested a function for panning left and right and to undo a zoom-in, two functions that sounds very reasonable to implement and that would add good value to the program

One test subject requested a search-functionality. Also something that probably would increase the speed of using the software significantly.

DISCUSSION

We have just presented the test result from the user tests of CyberForensics TimeLab compared to Forensic Toolkit. The single most interesting finding is that the test subjects solved the task much quicker using CFTL than using FTK, an improvement of approximately 3 times. If the same results would be achievable in real cases by the law enforcement, this could increase in a much faster time to solve cases. Since most agencies work with a budget it could also make it possible to solve cases that would not fit into the budget otherwise.

We have also seen that more questions have been correctly answered with CFTL than with FTK, 35 for CFTL and 31 for FTK. The difference can seem small, but if instead the number of incorrectly answered questions is counted we have one for CFTL and five for FTK. This means that the Traditional investigators made five times as many incorrect answers than the TimeLab investigators. In other words, it seem like it is easier to find answers to questions about a case with CFTL than with FTK. In reality this could lead to that the law enforcement will find evidence that they could not otherwise have found.

We have also seen that the test subjects considered that it was easier to solve the task with CFTL than with FTK. Since there are people in the test groups both with and without previous experience of computer forensics this would probably in reality result in that it will be easier to find people who are skilled enough to perform computer forensics investigation with CFTL and also that it is easier for trained computer forensics examiners to solve cases.

CFTL also makes it easier for examiners to see coherence of evidence. The user test showed with significance that it was much easier to see coherence between the evidence on the time line in CFTL than it was in FTK. CFTL was weighed +6 compared to FTK that was weighed -1 in this regard. When the law enforcement officials are investigating a case in traditional software they might search for files changed at the target time span, then they might search e-mail and IM messages. Maybe they did not however search the registry for changed keys, which would have showed to them that the suspect connected an USB hard drive and transferred data important to the case. With CFTL all these events are displayed on the same time line so when a specific time is investigated, all relevant evidence are showed in the same place.

We have in total found five different aspects where CFTL is performing better than FTK in the user tests. CFTL made it possible to solve the case three times faster and only 1 question was answered incorrectly. If the tool would work this well on a real case after improving its stability and increasing the supported file formats these 2 aspects would result in decreased costs and increased accuracy. The user tests also showed that it was easier to find answers to the questions, that it was more intuitive to use the tool and that it was easier to see coherence between the events on a given time span. If this were the same with a real case it would be easier for investigators to perform the examination and it would be possible that evidence not found otherwise would be found because interesting evidence are conveniently placed in the same area.

We have to keep in mind though, that CFTL still contains many bugs that make it unreliable. It also lacks support for many file types and file systems that can be found in for example FTK. FTK also have special feature to carve files from free space and in other files. It also have specialized views for browsing special kinds of evidence for example images and e-mail. These are all things that cannot be found in CFTL.

1. How can a computer program scanning a HDD for timestamps be implemented?

In the prototype Scanner we have used Perl as a programming language. Perl has proven to be a very suitable language for the task. Perl is additionally platform independent which makes it possible to connect the scanner to a native viewer for each operating system supported. During development the scanner has been developed and tested on both Mac OSX Leopard and Windows XP. The speed of searching through a hard drive, despite what you would think about a tool written in a script language, is acceptably fast comparing to other products. Since it is hard to compare between other products it is not possible to give any exact figures, since different products scans for different things. But to give some figures, a 5GB disk image (of which 2.3 GB is free space) with NTFS containing roughly 20 000 files and among these the scanner found and registered 152288 timestamps. The test was executed on a 2.4Ghz Intel Core 2 Duo. During the execution one of the cores where executing at a minimum of 90% during the whole execution. This tells us that it was not the hard drive that was the bottleneck and hence makes it uninteresting discussing. The whole execution was completed in approximately 5 minutes and 30 seconds. It would be unfair to make a similar test with FTK and compare the results. FTK handles more file types and scans the free space available on the disk. This will obviously take more time.

The choice of making a plug-in based system seems to have been a good idea whereas we have had some issues with some handlers, for example NTFS; it has been very easy to test the other parts of the system and disabling those not working properly at the moment. It also enforces a good breakdown of the software where each module handles a very isolated task. Other advantages in the future could be to make it possible for the end user to choose which modules to enable and in that way make the scans faster. It would also be possible to make a publicly available interface so that third parties can create add-ons. We have also made some experiments with trying to unit test these Perl modules by themselves and initially this seem to be a very good idea. Unit tests ought to be a very important part of a forensic tool since it is required to prove that the tool is operating in a correct way.

We have proved that it is very possible to make a scanner application that searches though a hard drive and scans it for timestamps. Out prototype implementation handles both NTFS and FAT16/FAT32 and it handles a number of file formats containing timestamps as well.

What needs to be done next to make the scanner part reliable is to write unit tests for the complete set of functionality. This will ensure that the Scanner is working as expected and will serve as evidence if the scanner is to be used in an investigation.

2. How can an application displaying a timeline with evidence be implemented?

Our chosen development language, C# seem like a sound choice. It was very easy to create a customized timeline view component and it was very easy to build the windows GUI. XML also seemed like a good format to use between the scanner and viewer. The format is fully readable and can easily be debugged. It is also completely open what is actually transferred and it can be verified in an investigation. There is also built in support for XML in C# which made it a very easy task importing this data into the viewer.

We managed to overcome the difficulties with making a fast UI with the large amount if information that has to be presented at once. This was successfully done with different caching techniques and by relieving the windows controls from holding the actual data but rather handing this task to a specialized controller.

The user test has also showed that the tool was somewhat intuitive and it was almost possible for all test subjects to solve all questions in the case using the tool. During the user test the users of FTK answered a total of five questions incorrectly while the users of CFTL only answered one question incorrectly. For a prototype where there are still bugs remaining, a large amount of missing functionality and lack of proper testing this can be considered a result much better than expected.

3. In what ways could such a tool be advantageous over modern ways of browsing evidence?

We have performed a user test with 12 test subjects where we have been able to compare CyberForensics TimeLab with leading software on the market today, Forensic Toolkit. The test case provided has been prepared to contain evidences, which are coherent in time, which could have made an advantage for CyberForensics TimeLab. We believe however that this is actually the most common case with evidence and that we have used a very realistic test case. We think that if you put an arbitrary number of evidence on a timeline many of these are actually coherent and will be spotted in the same area of the timeline, and when something interesting appear, you can see the whole chain of events to its left and right.

CONCLUSIONS

We have created a prototype tool that we call CyberForensics TimeLab. The tool is divided into two parts: A scanner that scans hard drives for evidence and generates index files. These index files can be read by the other part, the viewer. The scanner can handle can handle file systems like NTFS, FAT. It can then find e-mail, instant messaging and a number of other file formats, which it will add to the index file. The viewer will read index files and display a timeline where all evidences are displayed. In the viewer the end user can zoom, browse evidence by timestamp and look into evidence properties and hexadecimal data. The tool is evaluated by a set of test subjects that compared the solution with a well-known software forensic tool available in the market. The results show that it could be very advantageous to implement the ideas in CyberForensics TimeLab and that it at least would make it a competitor when the case is appropriate. The user testing shows a number of places where improvements should be made. All the test subjects seemed to be positive about the tool in general.

More information about CFTL will be found at http://cftl.rby.se/ in the future.

FUTURE WORK

First of all we plan on running the user evaluation test with more subjects. The results from this extended test together with the content of this master thesis will then be transformed into an academic paper and submitted to an international conference on computer forensics.

When it comes to the prototype itself we need to develop unit tests for the complete functionality of the prototype. This is essential to guarantee its correctness and to sort out the bugs that are still left in for example the NTFS handler. Regarding the functionality of the prototype we have identified a number of interesting areas, which are listed below.

A. Integrations with existing software

An interesting opportunity would be to integrate the functionality of CFTL into an existing computer forensics utility. In this way it might be possible to reuse evidence extraction from the tool and link the timeline to the evidence files in the tools native interface. The timeline could in this way be a very good complement working together with the current interface of the tool.

B. File Systems

The prototype only covered the NTFS and FAT16/32 file system. Although these are the most common ones in Windows desktop computers and servers, there are other systems that are commonly used. On the desktop side the Macintosh running OSX and the file system HFS+ is very common. Linux with EXT2/3 is also very common. On the server side there is a wide variety of file systems, which would be good to add support for. We also have removable media like CD-ROMs and DVD-ROMs using their own file systems.

Furthermore NTFS and many other file systems have a journal keeping log files for certain activities. These log files might be interesting to take into account as they might reveal events that would otherwise not be found.

C. Mobile phones

Mobile phones and especially smart phones are the new computers. Near everyone have one and there are many manufacturers and models using different operating systems and techniques. To be able to dump the contents of a mobile phone would fit very well into this software. Imagine being able to scan a suspect's mobile phone and get the incoming and outgoing SMS, calls and calendar and possibly even other data into the software. Many new phones have GPS hardware saving where the user have been located. Imagine this information combined with all information on the suspect's computer.

D. Instant Messaging

In the prototype we have built in support for dumping messages from Windows Live Messenger. There are other very popular instant messaging clients that we could scan as well. Some of the most common ones are ICQ, AOL Instant Messenger, MSN Messenger (the client built in Windows XP by default), Miranda, Pidgin, Trillian and mIRC.

E. E-Mail

We have built in support for e-mail clients compatible with the Unix MBX file format in the prototype, there are however other e-mail clients that use proprietary formats. For example Microsoft Outlook uses a special database format. Since Outlook is very commonly used, adding support for this format would be very useful. There are also other e-mail clients that should be added. We also need to consider web-based e-mail, as they are a very common alternative to using an application.

A. Automatic pattern search

All evidence is already stored in a uniform way, making it easy for a computer to handle. To add automatic pattern search to the viewer it would be possible for the investigator to automatically search for certain predefined patterns of suspicious activity, helping the investigator to spot interesting parts of the timeline more efficiently. Evidence could

B. Web Browsers

There are many web browsers freely available to anyone; some of the most common ones are Internet Explorer, Mozilla Firefox, Safari and Mosaic. In the prototype we only have support for scanning the History in Internet Explorer. It would be advantageous to add support for more web browsers.

C. Time Zones and Time Deviance

Today the prototype is considering all timestamps to be in the same time zone. When gathering evidence from many computer, a digital camera, a mobile phone. Each of these have their own system clock and there is usually a deviance between them and to compensate for that we both need to know the time zone of each device and the deviation from a trusted clock. The time then has to be adjusted to the same time zone and correct time.

D. Printers

Printers often contain cache of recent printed documents. The server the printer is connected to often also caches the pages before printing them. For example in Windows this cache is located at C:\Windows\Spool\Printers. To extract the information from the cache on the server, and to extract information directly from the printer would be advantageous. As there are a huge amount of printer manufacturers and models this will be a large job.

E. System and application log files

Although a number of system log files are already scanned by the prototype, there are many more to scan. For example there is a modem log showing when someone connected and disconnected from the Internet. Many file transfer programs keep a log file of the file transfers that could be interesting.

F. Network equipment log files

When users accesses their Internet connection, surfing the web or checking the e-mail they invoke a large amount of Internet equipment on the way from them to the target resource and back. Some examples are routers, switches and firewalls. Many of these have build in logging capabilities which would be very interesting to include in the scanning's. To join time-bound evidence from the suspects own computer together with evidence from the Internet infrastructure around would make it possible to find the same evidence in many places making it very reliable.

G. The thumbnails in Windows

Windows stores thumbnails for all files in a folder in a special database. Decoding this database could give us another source of timestamp data where we even could get data about files that no longer exist.

H. Compressed file archives

Files does not have to be placed directly in a file system but can also be placed in compressed folders. There are many different techniques that can be used for example ZIP and RAR, which are two common formats. These formats contain their own kind of timestamps and Meta data and could be scanned like the other file types.

I. Free-Space scan

It might be possible to find date in the slack space (space not used by any files) on a hard drive. Since timestamps often use a well-defined structure and if the investigator is able to choose a quite small time span to investigate it might be possible to find timestamps located in unusual places.

J. Calendar & planning software

There exist a large variety of calendars software that can be used to keep schedule. Many people keep their whole life planned in their digital calendars on the computer. It might be a good idea to extract the timestamps from the events in these programs and combine it with the rest of the evidence.

REFERENCES

[1] C. Procise and K. Mandia, *Incident Response and Computer Forensics 2nd ed.* New York, McGraw-Hill, 2003.
[2] E. Casey, *Digital evidence and computer crime 2nd ed.*, Amsterdam ; Boston, Elsevier Academic Press, 2004.
[3] A. J. Marcella and R. Greenfield *Cyber Forensics*, Boca Raton, FL, Auerbach Publications, 2001.
[4] G. Mohay et al. *Artech House computer security series, Computer and Intrusion Forensics*, Boston, Mass., Artech House cop., 2003.
[5] B. Carrier *File System Forensic Analysis*, Crawfordsville, Indiana, Addison Wesley Professional, 2005.
[6] "HDD Roadmap Diagram", Hitachi Global Storage Technologies – San Jose Research Center, San Jose, USA, 2003.
[7] (2007) The ILook Investigator product website. [Online]. Available: http://www.ilook-forensics.org/iLookv8.html
[8] D. L. Shinder and E Tittel, *Scene of the Cybercrime Computer Forensics Handbook*, Rockland, MA, Syngress Publishing Inc. 2002
[9] (2007) Wikipedia Forensics page [Online]. Available: http://en.wikipedia.org/wiki/Forensic_science
[10] H. L. Capron and J. A. Johnson *Computers*, *Tools for an Information Age – Seventh edition*, New Yersey, Prentice Hall, 2002.
[11] "Access to the Internet at home by type of connection 2006", Official Statistics of Sweden, Stockholm, Sweden
[12] "Use of a personal e-mail address 2006", Official Statistics of Sweden, Stockholm, Sweden
[13] (2007) Internet World Statistics – Global Village Online. [Online]. Available: http://www.internetworldstats.com/emarketing.htm
[14] "A Study about Cell Phone Usage", Harris Interactive - Public Relations Research
[15] G. Kipper, *Wireless Crime and Forensic Investigation*, Boca Raton, FL, Auerbach Publications, 2007
[16] E. Seagren, *Secure Your Network For Free*, Rockland, MA, Syngress Publishing Inc. 2007
[17] M. A. Caloyannides, *Privacy Protection and Computer Forensics Second Edition*, Norwood, MA, Artech House Inc., 2004
[18] (2007) Technical Note TN1150 - HFS Plus Volume Format. [Online]. Available: http://developer.apple.com/technotes/tn/tn1150.html
[19] Volume and File Structure of CDROM for Information Interchange, Standard ECMA-119, 1998
[20] Volume and File Structure of Read-Only and Write-Once Compact Disk Media for Information Interchange, Standard ECMA-168, 1994
[21] (2007) Wikipedia Windows Registry [Online]. Available: http://en.wikipedia.org/wiki/Windows_registry
[22] V. Mee, T. Tryfonas and I. Sutherland "The Windows Registry as a forensic artefact: Illustrating evidence collection for Internet usage" *Digital Investigation* Vol 3, pp. 166-173, 2006
[23] H. Carvey "The Windows Registry as a forensic resource" *Digital Investigation* Vol. 2, pp. 201-205, 2005
[24] Windows registry information for advanced users (Q256986), Microsoft, 2007
[25] (2007) Log Files – Apache HTTP Server [Online] Available: http://httpd.apache.org/docs/1.3/logs.html
[26] (2007) CPAN – Parse - Win32Registry-0.30 [Online] Available: http://search.cpan.org/~jmacfarla/Parse-Win32Registry-0.30/
[27] Eoghan Casey et al., *Handbook of Computer Crime Investigation*, Great Brittain, Elsevier Academic Press, 2007
[28] (2007) Whitehats.ca - Windows Event Log Format [Online] Available: http://www.whitehats.ca/main/members/Malik/malik_eventlogs/malik_eventlogs.html
[29] (2008) MSDN – Event Logging Structures [Online] Available: http://msdn2.microsoft.com/en-us/library/aa363659(VS.85).aspx
[30] J. Hager (2008) The Windows Shortcut File Format as reverse-engineered by Jesse Hager. [Online]. Available: http://mediasrv.ns.ac.yu/extra/fileformat/windows/lnk/shortcut.pdf
[31] (2008) No Title [Online]. Available: http://home.eunet.no/pnordahl/ntpasswd/WinReg.txt
[32] (2008) Samba – Opening windows to a wider world [Online]. Available: http://www.samba.org/
[33] (2008) ATA Security: Roadblock to Computer Forensics [Online]. Available: http://www.eevidencelabs.com/article/ATA_Security_Roadblock_to_Computer_Forensics.pdf
[34] D. Thomas and B. D. Loader, *Cybercrime - law enforcement, security and surveillance in the information age*, London, Routledge, 2000