UNIVERSITY OF OSLO
Department of Informatics

An Evaluation of Forensic Tools for Linux

Emphasizing EnCase and PyFlag

Master thesis
60 credits

Bård Egil Hermansen

November 8, 2010

Acknowledgements

First and foremost, I would like to thank my advisors for providing insight and expertise, for lending necessary equipment, hardware and literature, for guiding me in the right direction when I needed it, and for reviewing this document. The thesis would not be the same without your input (in alphabetical order):

- Anders Skolseg Bruvik
- Audun Jøsang
- Bente Christine Aasgard
- Margrete Raaum

I would also like to thank Guidance Software for providing a copy of EnCase Forensics, which was indispensable for this paper. Thanks to my friends and family for all the support and encouragement during the writing of this thesis, especially my friends Eivind Bergstøl for voluntarily offering to review this document, even though he was not required to, and Martin Wam for providing technical input during debugging of PyFlag. Finally I want to thank my grandmother, a kind and caring woman, Karen Helene Hermansen, who passed away while I was working on this thesis.

– Bård

Abstract

This thesis presents an in-depth assessment and comparison of several digital forensic tools, with a special emphasis on two particular tool kits. The first one called EnCase Forensics is a commercially available toolkit used by a large number of law enforcement around the world. The second one called PyFlag is an open source alternative used with great success in the winning contribution to the highly acclaimed Digital Forensics Research Workshop (DFRWS) anno 2008. Although tool kits are evaluated as a whole, the main focus is key evidence discovery capabilities.

Considering that most research in this field is based on the Microsoft Windows platform, while little research has been published regarding analysis of a Linux system, we investigate these tool kits primarily in a Linux environment. We conduct real world forensic acquisition and analysis of a Linux file system using the aforementioned tool kits, along with a Linux acquisition tool called dd. During the course of this thesis we have specified testing procedures, problems encountered during testing, and the final results.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer crime is currently one of the fastest growing types of crime. This clearly demonstrates the growing need for more powerful computer forensic capabilities [3, page 110]. To be able to efficiently investigate the increasing amount of computer related crime, or crime where computers are somehow involved, law enforcement agencies need more and more advanced computer forensic software. This kind of software is often referred to as tools or tool kits, and help an investigator perform acquisition and analysis of electronic evidence from digital media.

1.1 Main Contributions

The thesis clarify a prevalent misconception present in most literature on digital forensics, of RAM slack being filled with data from memory, when it is, in fact being padded with zeroes on most modern operating systems.

Furthermore, insight is provide into the relatively young science of digital forensics on Linux systems, showing how this can be accomplished employing current tools. The shortcomings of each tool is stated, along with proposed improvements, e.g. what each tool can gain by looking at the other solution. The authors hope this thesis serve to inspire other academics, practitioners and scientists to continue exploring the often neglected science of digital forensics on Linux systems.

1.2 Scope of Thesis

Digital forensics include a vast field of different techniques and specialized sub categories. This includes disk forensics, memory forensics and network forensics. For each of these three categories there are both live- or dead (post-mortem) analysis. Because forensics in general, but especially disk forensics and memory forensics are vastly different from operating system to operating system, we often differ between forensics on Windows and forensics on Linux.

Considering that that post-mortem disk forensics is the primary focus of both PyFlag and EnCase, this will be the primary focus of this thesis as well. Because Linux forensics is by far the least explored when compared to Windows, this thesis will mostly discuss forensics on Linux. However, since EnCase is run from Windows, and most digital forensic literature discuss forensics on Windows, some content will also apply to a Windows environment.

Due to the vast range of features available in both EnCase and PyFlag, the key features for data discovery are the primary focus during testing. Other features are considered, but not

tested. These features may also be included in the evaluation, but based solely on information from documentation.

Consequently, this thesis will not contain detailed information regarding memory- and network forensics, and live analysis. Additionally, not every available feature of the evaluated tool kits are tested. Moreover, the thesis will not focus on hardware compatibility, since the focus is on evidence searching capabilities, and not hardware support. Lastly, the legal aspects of digital forensics are not discussed.

1.3 The Importance of Digital Forensics

The importance of digital forensics is continuously growing as more users connect to the Internet, and the application of desktop and laptop computers are becoming the norm. Contrary to what some may believe, digital forensics are not only useful for computer related crime, but rather in all fields of criminology. As stated by the textbook on digital forensics used by the Norwegian Police Academy [3, page 43] computer forensics investigations are used in almost 100 percent of civil cases to find evidence that supports or refutes wrongful dismissals, breaches of contract, or discrimination violations. Digital crime investigations are not limited to cybercrimes or cyber terrorist activities. Traditional or physical crimes or misconduct may involve e-evidence and, as a result, leave cybertrails of those activities [3, page 14].

1.4 Introduction to Digital Forensics

Lets start by looking at what is meant by the term digital forensics. Digital forensics and computer forensics are often used interchangeably. As a result these terms are also used throughout the thesis, both referring to the same science. Below are three definitions of digital/computer forensics resulting from a quick "define:Digital forensics" on Google.

- Computer forensics is a branch of forensic science pertaining to legal evidence found in computers and digital storage media. Computer forensics is also known as digital forensics.
- Digital forensics is a field of science of applying digital technologies to legal questions arising from criminal investigations.
- Computer forensics is the use of specialized techniques for recovery, authentication, and analysis of electronic data when a case involves issues relating to reconstruction of computer usage, examination of residual data, authentication of data by technical analysis or explanation of technical features of data and computer usage.

According to [10], the typical definition of digital forensics is a process that involves the preservation, identification, extraction, documentation, and interpretation of computer data. A leading edge researcher and software developer within the digital forensics community, Brian Carrier, has come up with his own definition [10]:

> A process that formulates and tests hypotheses to answer questions about digital events or the state of digital data

As we can see, the science of digital forensics can be pretty comprehensive. To be able to evaluate the tools, or rather tool kits, since they comprise of several tools implemented in a single tool suite, we look at a typical forensics case. One can learn much about the necessary steps in computer forensics by observing how police proceed when confronted with a physical crime scene. When comparing this with how computer forensic specialists work, some parallels can be drawn. When the police arrive at a crime scene, the first priority is to isolate the

crime scene from contamination. In digital forensics, this would include unplugging network cables or disabling wireless access. Next, the police would photograph the crime scene to preserve how it was when they arrived. This would amount to acquiring a digital image, or an exact duplicate of the perpetrators computer, and taking pictures with a camera of the screen. While forensic technicians collect evidence, great care has to be taken, as not to contaminate the crime scene. For a traditional physical crime scene this involves putting on latex gloves, and treading only where it is absolutely needed. The same holds true for digital forensics. While collecting evidence, digital forensic tool kits tend to leave a footprint, i.e. modified data, in effect contaminating the electronic evidence. The smaller the footprint, the better. Like putting on blue plastic socks, measures are taken by the digital forensic investigator to minimize evidenciary contamination, such as mounting a file system with read-only privileges, or utilizing low memory requirement applications when conducting memory forensics. Following evidence acquisition, it is paramount to maintain a strict chain of custody, ensuring that evidence remains untainted. In physical forensics, this is accomplished by maintaining great control over who handles the evidenciary material at all times. If the evidence is not handled by anyone, the evidence would remain locked in a vault to prevent contamination [3, page 51]. This analogy is translated into digital forensics by sealing the evidence with a check sum employing cryptographic hash verification, in effect ensuring that any change to the original evidence, however small, is detected. Only after the crime scene has been isolated and preserved would the search for evidence begin. In a nutshell, traditional physical forensics could be considered the foundation for how computer forensics are conducted.

Digital forensic tool kits are complex, with many functions, and are used during several phases of an investigation. Tool kits used in computer forensics have a feature list spanning across a multitude of phases during an investigation, all the way from first response and acquisition to the analysis of evidence, and documentation. To be able to fairly evaluate all the different features and uses of a toolkit, we are required to understand, at the least, how a regular forensic case is directed. To provide the reader with the necessary knowledge required to understand how tool kits operate, a larger than usual portion of this thesis is dedicated to providing background information as to facilitate an understanding of digital forensics in general.

To be able to conduct digital forensics one should have a basic understanding of forensically relevant legal requirements, and an in-depth understanding of how computers operate. This thesis will focus primarily on the technical aspect of computer forensics. For readers more interested in the legal aspects of computer forensics, the book used by the Norwegian Police Academy [3] would come recommended.

1.5 Categories of Cybercrime

Next, the different categories of cybercrime are discussed, which is relevant for identifying the motive, and thus the intent and probable actions taken by the perpetrator. According to [3, page 62-63] perpetrators utilize different methodologies depending on the type of crime committed, and thus tool kits will be utilized differently according to the type of crime. According to [3, page 61-65], there are basically four kinds of cybercrimes.

- Computer as Target
- Computer as Instrument
- Computer is incidental to traditional crimes
- New Crimes Generated by the Prevalence of Computers

Profit-motivated crimes in which the computer is the target typically involve the search for information [3, page 61]. These offenses might be theft of intellectual property, customer credit card numbers, customer lists, pricing data, or other marketing information [3, page 61]. More visible cybercrimes are blackmail based on information gained from computerized files, such as medical information, personal history, or sexual preference [3, page 61].

Using a computer as an instrument could imply that the criminal obtains control over a third party computer, and uses the system resources for his own needs. This could happen by inserting malicious code to hijack the computer's processes [3, page 64]. These kinds of crimes tend to be profit motivated [3, page 64]. An example would be financial fraud.

There exists a third category where the criminal does not need a computer to commit the crime, but use of a computer facilitates the criminal act [3, page 65]. These are traditional crimes that became easier or more widespread because of telecommunication networks and powerful PCs [3, page 65]. Money laundering, organized crimes, illegal gambling and drugs, child pornography, and other exploitations are the most common of these crimes [3, page 65].

As a result of computer technology getting more and more prevalent in modern society, new crimes are created as a result of this technology. Software piracy, Internet and auction fraud, copyright violation of computer programs, counterfeit equipment, black market computer equipment and programs, and theft of technological equipment fall into this category of computer crime [3, page 66]. Viruses and worms can fall into this category [3, page 66].

1.6 Three Different Scientific Methods

There are currently three major paradigms used in computer science:

1.6.1 Theory

1. characterize objects of study (definition)
2. hypothesize possible relationships among them (theorem)
3. determine whether the relationships are true (proof)
4. interpret results

Theory is the bedrock of the mathematical sciences. Applied mathematicians share the notion that science advances only on a foundation of sound mathematics.

1.6.2 Abstraction

1. form a hypothesis
2. construct a model and make a prediction
3. design an experiment and collect data
4. analyze results

Abstraction (modeling) is the bedrock of the natural sciences. Scientists share the notion that scientific progress is achieved primarily by formulating hypotheses and systematically following the modeling process to verify and validate them.

1.6.3 Design

1. state requirements
2. state specifications
3. design and implement the system
4. test the system

Design is the bedrock of engineering. Engineers share the notion that progress is achieved primarily by posing problems and systematically following the design process to construct systems that solve them.

Closer examination however reveals that in computing the three processes are so intricately intertwined that it is irrational to say that any one is fundamental.

1.7 Discussion and Selection of Scientific Method

Lets start by looking at the first scientific method called Theory. This scientific method is more concerned with predicting causal relationships based on a formed hypothesis. This method is great for investigating cause and effect, but considering the requirements needed for evaluating and testing two tool kits, it does not fit the job description. Therefore we have to conclude that the scientific method Theory is irrelevant for this thesis.

The second scientific method called Abstraction, is concerned with designing an experiment and analyzing the results, which is highly relevant when evaluating two opposing tools. However, this thesis is not about predicting what will occur if some prerequisite is met, and certainly not with constructing a model. The conclusion has to be that the first and second items stated under Abstraction is irrelevant, while the latter two are suited for implementing in this thesis. The method Abstraction should therefore be considered partly relevant.

The third and last method called Design is considered the most suited scientific method for this thesis. All items under Design, except item three, are relevant. Stating the requirements regarding suitable features and performance, and evaluating to what degree these requirements are met, is one of the most important aspects of this thesis. The second item, stating the specifications, are relevant because we want to gain an insight into what the software is capable of and its features. The last item, testing the system, is relevant because we want to test these features, looking for bugs and insufficiencies.

Considering the statements above, the scientific method utilized by this thesis will be an adapted version of Design complemented by Abstraction. Since the main emphasis is on testing and comparing two tool kits, and not designing a new system, some modifications of the approach are required. We list and discuss the requirements and evaluation criteria in detail before explaining the specifications and features of each respective toolkit. The experiment is designed, data is collected and the tool kits are then measured against the selected requirements and evaluation criteria previously selected, and compared to each other. Finally, potential improvements for each toolkit are suggested. The implemented scientific method should be considered a hybrid of Design and Abstraction with the following specification:

1. State requirements
2. State specifications
3. Design an experiment
4. Test the system and collect data
5. Analyze results

Chapter 2

Background

In general, most investigations can be reduced to four basic questions: (a) what happened? (b) when did it happen? (c) how did it happen? and (d) who did it? [12, page 1]. Some also include where and why [3, page 42]. To help answer these questions in a reliable and secure manner forensic experts employ specially designed software tools. In this chapter we will look at how tools and specialized hardware are needed to conduct forensics. To be able to evaluate tools, with their multitude of features and uses during the different phases of an investigation, we need to know what these phases are, and how a forensic investigation is done in practice. For this reason we will look at and discuss several models proposed by different textbooks used for educating police forensic officers and forensic specialists. Most of this chapter is dedicated to looking at the different types of evidence and where tools and forensics personnel search for evidence during an investigation. By having knowledge about where evidence can hide, it is easier to create an image containing evidence in all the hard to reach locations facilitating a thorough and fair evaluation. To be able to conduct a fair and scientifically correct evaluation, we also discuss different analysis environments used for forensics, analysis modes employed and specialized forensics hardware equipment needed to conduct forensics and to use tools to their full effect. Finally this chapter provides a basic understanding of maintaining admissibility of evidence.

2.1 Why Tools are Needed

Why not just perform a *live analysis* of the system, meaning an analysis of the system while it is still running, using built in commands? What if an investigator use 'ps' to list the running processes on the system, or 'ls' to list the contents of a directory?

One reason is because each and every command performed on the suspect system destroys some of its content. When a command is executed, it has to be loaded into memory, which might overwrite memory resident evidence. If memory is full, some part of memory is swapped out to the swap file or swap partition depending on the system configuration, potentially destroying evidenciary data on the disk. Additionally, logs and historical data, like recently executed commands, may be updated causing further contamination to the evidence.

Secondly, file systems tend to flag certain data that should not be displayed to a user under normal circumstances [6]. Since a forensic investigation want to yield all available information to the investigator, even flagged data has to be displayed [6], which is possible using a forensic toolkit.

Third, the suspect system cannot be trusted [6]. When running a command, many parts of the system are intertwined and rely on output from one another. This highly interdependent

functionality means that one compromised component often is enough for the misleading output to propagate undetected through the system and be displayed to the user.

In [18] Dan Farmer and Wietse Venema explains what happens when a command is executed:

1. The shell first parses what is typed in, then forks and executes the command (environment and path variables can have a significant effect on exactly which command gets executed with what arguments, libraries, and so on).

2. The kernel then needs to validate that is has an executable. It talks to the media device driver, which then speaks to the medium itself. The inode is accessed and the kernel reads the file headers to verify that it can execute.

3. Memory is allocated for the program, and the text, data, and stack regions are read.

4. Unless the file is a statically linked executable (that is, fully contains all the necessary code to execute), appropriate dynamic libraries are read in as needed.

5. The command is finally executed.

As one can imagine, there are many different ways a savvy computer criminal can modify the system to alter the output observed by the computer forensic specialist. The shell can be modified to alter how the commands are parsed, the environment and path variables can be changed, or arguments altered to run a modified version of the intended command, or maliciously altered libraries can give thwarted output. The malicious code altering the output seen by the user when running a command is generally referred to as a *root kit*. After the introduction of root kits in the 90's, the investigators could no longer allow themselves to trust system calls or libraries present on the suspect system. Instead they had to bring their own libraries and statically linked executables specifically engineered to ensure the integrity of the evidence. One of the more sophisticated root kits out there is called Blue Pill. By wrapping the operating system inside a *hypervisor*, i.e. a virtual machine monitor, system calls and interrupts are redirected to the Blue Pill interface. In effect, by virtualizing and trapping the computer system, Blue Pill may falsify output to the user, while remaining nearly undetectable.

Fourth, as hard drives have grown larger and larger, with more and more files, more complex operating systems and software, and a wide array of different file formats, a need for a more intuitive and feasible approach has surfaced. Some tools assist the investigator by comparing files on the suspect system to a *hash database*, which is a database consisting of calculated hash values of common files used to identify files while verifying their integrity. This is not an approach unique to digital forensics, considering how scientific methods often involve correlation of new findings against published data. Hash analysis helps eliminating known, unaltered and irrelevant files from the view of the investigator, while flagging other files as suspicious. A file which is renamed to another format, in effect changing the file extension, is an example of a file that would be considered suspicious. If for instance a file called long-live-the-leader.jpg is recognized as an OpenOffice document, it would be flagged, which is done by comparing the first sectors of a file, called a file header, to the file extension.

Fifth, some tools also present the investigator with a graphical user interface (GUI) instead of the traditional command line based tools, making it easier to maintain a clear outline of the case at hand.

Finally, there exists a need for automation and standardization. *Automation* is needed to allow the investigator to do other useful things while the tool is running, which can in some cases take hours or days. The preferred scenario would be if the investigator could start an analysis before going home from work, then coming in the next morning only to find a complete report displayed on the screen. Some tools include an interface for scripting a queue of commands to be executed, allowing a series of commands to be automated. *Standardization* is needed to allow a consistent way of analyzing multiple systems, across multiple platforms, and to produce consistent, accurate and reliable results.

2.2 Common Practice Digital Forensics Procedure

According to the textbook used to educate Norwegian police officers specializing in digital forensics [3, page 42] the overall goal in criminal investigations is to get a better understanding of what happened by finding the five *W*s: who, what, where, when and why. According to [3, page 42], anything helping law enforcement or investigators get a better understanding of what happened or what suspects were doing prior to the committed crime is immensely useful. Next are some suggested models that can provide structure and a framework to the otherwise chaotic process of forensics, making it that much easier to answer these questions. Since tool kits are an integral part of any computer forensic investigation, and since tools often contribute during several phases of this process, these models should be considered important for any forensic tool developer, as they provide the basis for tool functionality.

2.2.1 The National Institute of Justice model

As mentioned in [10] the National Institute of Justice (NIJ) in the United States uses a template for how digital forensic investigations should be conducted:

- **Preparation**
- **Collection**
- **Examination:** Make the electronic evidence "visible" and document contents of the system. Data reduction is performed to identify the evidence.
- **Analysis:** Analyze the evidence from the Examination phase to determine the "significance and probative value."
- **Reporting**

2.2.2 The Model Taught at the Norwegian Police Academy

To complement the approach used by the National Institute of Justice, the textbook used for educating forensic investigators at the Norwegian Police Academy [3, page 44] states that the computer forensic investigation is a process consisting of five stages:

1. **Intelligence:** The process begins with an analysis of the situation to gain a basic understanding of the issues surrounding the incident, crime, or crime scene.
2. **Hypothesis or Theory Formulation:** Based on what is learned during Intelligence, the investigator formulates a hypothesis or theory of the case. The theory is important in interpreting the evidence to come up with answers to the five *W*s.
3. **Evidence Collection:** Evidence is gathered that will be used to test the hypothesis. It is critical that the search for evidence is not limited only to supporting evidence.
4. **Testing:** The e-evidence is examined to identify what could or could not have happened. It may be necessary to collect more evidence or to start over from the Intelligence phase.
5. **Conclusion:** Based on the e-evidence available at the time, a conclusion is reached that the evidence either supports the hypothesis or fails to support the hypothesis.

2.2.3 The Model Proposed by Brian Carrier

Brian Carrier is one of the leading pioneers in computer forensics, having developed The Sleuth Kit, and authored the book File System Forensic Analysis. According to [11, page 5] the process of forensic investigation has three major phases:

- System Preservation Phase
- Evidence Searching Phase
- Event Reconstruction Phase

As stated by [11, page 5] these phases do not need to occur one after another. Since more detailed information is readily available for this model than the first two, an elaboration of the three phases of Carriers model will follow, allowing the reader more insight into the forensic process.

2.2.3.1 System Preservation Phase

The first phase of the investigation process is the System Preservation Phase, where preserving the state of the digital crime scene is main priority [11, page 5]. What actions are taken during this phase varies according to business, legal and operational requirements of the investigation [11, page 6]. The goal of this phase is reducing the amount of evidence that may be overwritten [11, page 6]. This process does not end when data has been acquired, because data needs to be preserved for future analysis, also during subsequent phases of the investigation [11, page 5].

There exists several techniques to preserving the system. Considering that the goal during this phase is reducing the amount of overwritten evidence, it is crucial to limit the number of processes being able to write to the storage devices [11, page 6]. There are different techniques employed depending on whether we are conducting a live analysis or a dead analysis. When conducting a dead analysis [1] all processes are terminated by powering down the system, and duplicate copies of all data are acquired [11, page 6].

Conversely, processes are killed or suspended during a live analysis [11, page 6]. To prevent the perpetrator from connecting from a remote system and erase data, the network connection can be disconnected by plugging the system into an empty hub or switch, preventing log messages regarding a dead link [11, page 6]. Another option involves applying network filters [11, page 6]. To prevent important data from being overwritten while searching for evidence, this data get first priority when copying data from the system [11, page 6].

A cryptographic hash should be calculated right after important data have been saved, regardless of conducting a dead or live analysis. By obtaining hash values from acquired evidence, we are able to prove that the evidence remains unchanged from the original media [11, page 6]. A cryptographic hash such as MD5, SHA-1, and SHA-256, is a mathematical algorithm that generates a block of bits of a fixed size based on input data [11, page 6]. A minuscule change in the input data, even a single bit, results in a dramatic change in the hash value [11, page 6]. This is generally referred to as the *avalanche effect*. The algorithms are designed in a way that it is unfeasible to find two inputs generating the same output [11, page 6]. Consequently, it is safe to assume that a change in the hash value indicates that data has been modified [11, page 6].

2.2.3.2 Evidence Searching Phase

After data has been preserved, the search for evidence can commence [11, page 7]. Typically, the approach involves using a survey, based on the type of incident, to identify common locations where relevant evidence may be found [11, page 7]. During the investigation process we form hypothesis for what may have occurred, and search for supporting or refuting evidence [11,

[1] Dead analysis is the same as post mortem analysis, meaning analysis on acquired evidence after system shutdown.

page 7]. According to [11, page 7], the theory behind the searching process is fairly simple. The general characteristics of the object we are looking for should be defined, and subsequently we start looking for that object in a collection of data [11, page 7]. If for instance looking for an incriminating picture know to be stored in the jpg format, we would first search for all files with the extension jpg, and then start looking for the specific jpg file. There are two key steps in this phase [11, page 7]. One is determining what evidence we are looking for [11, page 7]. The other is where we expect to find the evidence [11, page 7].

According to [11, page 7], most evidence searching is conducted in a file system and inside files. A widely used technique for searching is conducting a *string search*, also known as a *keyword search*, which is to search for files based on their names or patterns in their names [11, page 7]. By employing string searching it is also possible to search the contents of these files, if they are in an ASCII format, for a matching string. Another commonly employed search technique is searching for files based on temporal data, i.e. time related data such as last accessed or last modified [11, page 7].

One way of searching is comparing the MD5 or SHA-1 hash of a file's content to a hash database of known files, such as the National Software Reference Library (NSRL) [2] [11, page 7]. Hash databases are used for finding files known to be good or bad [11, page 7]. This is generally referred to as *hash analysis*. Another approach is by searching for files based on signatures in their content [11, page 7], typically the file header. This type of analysis is called *file signature analysis*. Using file signature analysis enables us to find all files of a given type, even if someone changed the file name [**?**, page 7], or file extension.

When conducting network analysis, we search for packets going to a specific port, or from a specific source address [11, page 7]. Additionally, these packets can be searched for certain keywords [11, page 7].

2.2.3.3 Event Reconstruction Phase

The final phase of the investigation is using evidence found from the previous phases to determine what events may have occurred [11, page 7], which is referred to as *event reconstruction*. After the completion of this phase, the digital evidence should be correlated with physical evidence [11, page 8]. For an investigator to be able to form hypothesis, detailed knowledge is required regarding application- and operating system capabilities for software installed on the system [11, page 8]. This type of analysis can be found in [13].

2.2.4 Comparison of the Three Models

2.2.4.1 Similarities

First, lets start by looking at the similarities between the three approaches. The first phase of the two first models(NIJ discussed in 2.2.1 and NPA discussed in 2.2.2, that being the preparation phase and the intelligence phase, have somewhat of an overlapping meaning. The intelligence phase in the NPA model states that an investigator should get a basic understanding of the case at hand. One can reasonably argue that this has significant overlap with phase 1 - preparing for a case in the NIJ model, since preparing for a case would imply that one has to obtain knowledge of the case prior to conducting the investigation. This could be the nature of the crime, what crime the suspect is charged with, and what crimes, if any, the suspect has been charged with or accused of in earlier life. The preparation phase would also consist of acquiring detailed knowledge of the hardware at the crime scene. The preparation/intelligence phase could consist of talking to the lead investigator, searching through crime records, looking

[2] The NSRL hash database is available at http://www.nsrl.nist.gov

at photographs of the crime scene and the hardware to know what equipment and software to bring etc.

The second phase in the NIJ model, collection, correlates with the third phase in the NPA model, evidence collection. Even though the model proposed by Carrier in 2.2.3 has no phase dedicated to acquisition, it is still, although briefly, mentioned during the first phase called the system preservation phase. In addition the evidence searching phase in the Carrier model is somewhat related to the evidence collection phase in NPA model, since both phases talks about searching for evidence. The evidence searching phase would also relate to phase 3 and 4 of the NIJ model, that being the examination phase and the analysis phase.

The testing during the fourth phase, testing, in the NPA model claims that evidence should be examined, which is also what occurs in the third phase, examination, of the NIJ model. This is also basically what happens in the evidence searching phase in the Carrier model.

2.2.4.2 Differences

Lets proceed to analyzing the differences between the three approaches. Although phase four in the NPA model (2.2.2) has many similarities with the third phase in the NIJ model (2.2.1), the NPA model has the additional implication that the currently chosen hypothesis, the events regarding and surrounding the crime, should be tested. This implies comparing the current evidence with the hypothesis to see if there is any *inculpatory evidence*, supporting the current theory, or *exculpatory evidence*, contradicting the theory. It is always important to have an open and objective mind when conducting a forensic investigation, although not so objective that no theory or hypothesis can be formulated. A main difference between the model proposed by Carrier in 2.2.3 and the two other models is that Carrier argues that a hypothesis should only be formed after looking at the evidence, while the NPA model suggests that a hypothesis should be formed before collecting evidence. This is a major difference. On one hand, if a hypothesis is formed before looking at the evidence, the investigator could feel a form of ownership and pride of his cleverly hatched theory. This could lead to the investigator being somewhat biased in that he will try looking for inculpatory evidence, that being evidence which only supports the theory, which in turn could lead to the investigator overlooking contradicting evidence. On the other hand, it could be wise to have a certain idea of what happened so the investigator has some leads regarding where to look and what to look for. Another main difference between the Carrier model and the two other models is that the other models makes no mention of the importance of preserving evidence. The act of evidence preservation is an important one. Without it, evidence could be refuted in court and deemed inadmissible. More importantly, evidence could be contaminated to the point where it is rendered useless, no longer providing accurate or relevant information.

Comparing the approach listed in the NIJ model(2.2.1) to the approach in the NPA model (2.2.2), the approach by the National Institute of Justice and the Carrier model (2.2.3) could arguably benefit by including a theory formulation phase, the importance of testing and reevaluating the theory, and a conclusion phase, non of which is present in these two approaches. Conversely the approach adopted by the Norwegian Police Academy textbook in 2.2.2 could benefit by more importantly including an analysis and last by not least a reporting stage, seeing the importance of documentation in a digital forensic process to secure the admissibility of evidence.

2.2.4.3 Conclusion

According to [11, page 5] there is no single way to conduct an investigation. The model proposed by Carrier focuses around an event that occurred, like a “hacker” breaking into a government computer. These kinds of criminal investigations would clearly benefit from being able to reconstruct what really happened, both to produce a solid prosecution and to understand how

it was done so it can be prevented in the future. The need for reconstructing what happened might not always be necessary. The search might not be regarding an event, but rather a general search for incriminating evidence, like a hard drive loaded with child pornography. In this case there would not be an incentive for the investigators to reconstruct an event, because there exists no event in the first place. Rather the mere presence of child pornography photos on the hard drive would be more than enough to charge the suspect. The only scenario that comes to mind where a hard drive containing child pornography would be subject to event reconstruction would be if the suspect was a member of an organized child pornography network on Internet, and finding data regarding IP addresses of where the child pornography was downloaded from could expose other members of the network. Another scenario where event reconstruction is not always an issue would be with CERT/CSIRT units in non-police financial or public/communal institutions, focusing more on regaining control than to reconstruct what happened. If the point of entry is discovered and known, institutions may sometimes be forced to stop the investigation, once control is regained, due to inadequate resources or financial limitations. The Carrier model in 2.2.3 would therefore be more suitable to crimes where a specific incident occurred where the primary focus is to find out what happened. This would also somewhat be the case for the NPA model in 2.2.2 since it relies heavily on forming a hypothesis and later testing the hypothesis. On the other hand the National Institute of Justice model would be considered more general in that it has no event reconstruction or an hypothesis, so it would therefore be more applicable to our example with a hard drive full of child pornography, but not in our example of a hacker breaking into a computer. Also the principle of system preservation from the Carrier model in 2.2.3 would be a applicable in all three models, even though it's not even mentioned in the National Institute of Justice model and in 2.2.2. The above statements has therefore led us to conclude that these three approaches should not be looked at from a competitive standpoint, by should rather be considered complementary.

Although a tool cannot be expected to contribute during all these phases in these models, it should ideally be able to contribute during the collection, examination, analysis and reporting stages of the models. The report should as a minimum contain who handled the evidence to maintain chain of custody, what analysis- and acquisition features of the tool the evidence was exposed to, and lastly the result of the analysis.

2.2.5 General Guidelines for Digital Forensics Procedures

Since we concluded in the section above that the models in 2.2.1, 2.2.2 and 2.2.3 should be used for different categories of cybercrime, and that they should be considered complementary, we have included some general guidelines for how an investigation should take place. Not every investigation will use the same procedures, and there could be situations where you need to develop a new procedure [11, page 8]. There are some techniques that have not been implemented, so you may have to improvise to find the evidence [11, page 8]. In [11, page 8] Carrier proposes the PICL guidelines. As stated by [11, page 8] PICL stands for:

- Preservation
- Isolation
- Correlation
- Logging

According to [11, page 8] the motivation behind preservation is that you do not want to modify any data that could have been evidence, and you do not want to be in a courtroom where the other side tries to convince the jury that you may have overwritten exculpatory evidence. According to [11, page 8], during this phase, important data should be copied, the original should be put in a safe place, and analysis should only be performed on the copy so that the original can be restored if the data is modified. Also, it is stated by [11, page 8] that MD5 or SHA hashes of important data should be calculated to help prove that the data has not been modified. Write-blocking devices should be used during procedures that could write to the

suspect data [11, page 9]. The number of files created during a live analysis should be kept to a minimum in case they could overwrite evidence in unallocated space [11, page 9], and one should be careful about opening files on the suspect system during a live analysis because it could modify data, such as last access time [11, page 9].

The second guideline is to isolate the analysis environment from both the suspect data and the outside world [11, page 9]. You want to isolate yourself from the suspect data because you do not know what it might do [11, page 9]. Running an executable from the suspect system could delete all files on your computer, or it could communicate with a remote system [11, page 9]. Opening an HTML file from the suspect system could cause your Web browser to execute scripts and download files from a remote server [11, page 9]. Isolation from the suspect data is implemented by viewing data in applications that have limited functionality or in a virtual environment, such as VMWare, that can be easily rebuilt if it is destroyed [11, page 9]. You should isolate yourself from the outside world so that no tampering can occur and so that you do not transmit anything that you did not want to [11, page 9]. Isolation from the outside world is typically implemented using an analysis network that is not connected to the outside world or that is connected using a firewall that allows only limited connectivity [11, page 9].

The third guideline is to correlate data with other independent sources [11, page 9]. This helps reduce the risk of forged data [11, page 9]. If time is very important in your investigation, you should try to find log entries, network traffic, or other events that can confirm the file activity times [11, page 9].

The final guideline is to log and document your actions [11, page 9]. This helps identify what searches you have not yet conducted and what your results were [11, page 9]. When doing a live analysis or performing techniques that will modify data, it is important to document what you do so that you can later document what changes in the system were because of your actions [11, page 9].

2.2.6 Digital Forensics in Practice at New York State Police

Next, we look at how actual police departments handle a forensic case in practice. According to [3, page 51-52] the New York State Police reacts by first confiscating the computer and relevant technical devices. The officers that seized the equipment then brings it to the NYS Forensic Investigation Center (FIC) where it is locked into a vault. Once the NYS FIC proceeds with the investigation, a backup is created from the original devices using the tools Safeback, Expert Witness (now known as EnCase), and Snapback, before being stored on optical media. This backup process may in some cases take up to two weeks, depending on what sorts of problems are encountered. Typical problems encountered are being unable to get a backup due to venerable systems, double-spaced formatting schemes or compressed hard drives, certain operating systems (especially regarding Macintosh devices), or drive incompatibility, both hardware and software related. After a case file has been created for the case, EnCase is used for raw extraction and organization of the evidence. Using EnCase the NYS FIC investigators search for relevant evidence using regular expression searches. The evidence is then viewed using built-in content viewers in EnCase. After evidence has been extracted, analysis begins. During this process analysts use their experience and training to search the computer evidence for documents, deleted files, images, e-mail, slack space, and unallocated disk space. Following the analysis stage, the lead investigator correlates all computer events, like finding the order of events using timeline- or MAC analysis, looking for related activities and searching for contradictory evidence. Sequentially, an investigator correlates the electronic evidence with non-computer evidence, such as credit card receipts, physical forensic evidence, crime scene reports and eyewitness testimony to obtain a more complete understanding regarding the chain-of-events. The case is then presented in court through standard office software.

The phases employed throughout the investigation is according to [3, page 51-52] the following:

1. Seizing the computer
2. Backup
3. Evidence extraction
4. Case creation
5. Case analysis
6. Correlation of computer events
7. Correlation of non-computer events
8. Case presentation

2.2.7 Why This is Relevant

Tools have to take into account how digital forensics are done in real life. This means facilitating optimal solutions to an ever evolving and quite young research field, while keeping pace with constantly changing technology. It means that tools will have to integrate well with real world digital forensic procedures, and provide users with the right set of features needed to do the job. Some tools are already taking into account the dire need to document every step along the way during a digital forensic investigation, providing a solution to the strict need for documentation, while preserving and contributing to the admissibility of evidence. The models and guidelines outlined previously in this chapter serve as a basis for formulating the toolkit requirements postulated later in the thesis.

2.3 The Process of Digital Forensics

By merging the models, guidelines and general practice in 2.2 on page 9 in conjunction with the outline of the forensic process stated in [3, page 85-92] we have created a composite proposal of how the process of a computer forensic case, from the standpoint of a computer forensic investigator, could be executed:

1. Preparation
    - Talk to people
    - Know what equipment to bring
    - Check records
2. Document the crime
    - Take notes and pictures
    - Identify devices
3. Decision: Live- or post mortem analysis
4. Unplug
5. Create forensic copy
    - Employ write-blocking devices or mount as read-only
6. Copy to a forensically wiped hard drive
7. Compute CRC and hash

8. Evidence searching and analysis
    - Bookmarking data
    - Theory formulation
9. Event Reconstruction
10. Documentation
    - Update backlog
    - Maintain evidence log
    - Create report

2.3.1 Preparation

It is a good idea to prepare for the case at hand by gathering information about the incident, and the circumstances regarding the suspected crime. Talking to the lead investigator could provide the necessary information. As previously stated in 2.2.4.1 on page 11 observing the hardware from pictures of the crime scene can provide a clue to what special equipment might be needed during first response. Also, inspecting the suspects record can possibly provide insight to the nature of the crime. Last, but not least, knowing what the suspect is charged with could be helpful. If the investigation is conducted without the presence of personnel with proper police authority, and at the premises belonging to a financial or public institution, the investigator needs permit from- and access to a person with the authority to make decisions. The investigator should interview the persons who discovered the incident, and if available, the victim. Additionally, the local administrator could prove a valuable resource of information, often possessing detailed knowledge of the computer systems at hand. The obtained information during the preparation phase can prove invaluable when correlating with evidence later for formulating a theory of what really happened. As a last note, the investigator and first responders should always have a collection of useful kits available at all times, containing software with the latest updates.

It is entirely feasible to conceive a situation, where personnel are unwilling to disclose the occurrence of security related issues, due to proprietary systems or bad publicity, which can complicate the investigation.

2.3.2 Document the Crime Scene

The first critical step of an investigation during first response is documenting the crime scalene thoroughly. This can often be a lengthy, albeit important, process. If not everything is documented, evidence might be deemed inadmissible in court. In [3, page 86] it is concluded that before removing or seizing a PC, all connections, devices and cables attached to or in the vicinity of the PC, should be properly documented. The documentation can consist of written or voice-recorded notes, high-resolution photographs, or video. According to [3, page 86], some combination of the aforementioned documentation routines are advisable. This documentation should also identify other devices such as PDAs or digital cameras that could contain evidence [**?**, page 83] If the computer is running, the open screens and running programs should be documented through photography [3, page 86]. When encountering an incident involving several computers spread out over differing timezones, this timezone information has to be noted, and it should be verified that each machine has a correctly configured clock. Crucially, the documentation must contain what has been done to the evidence, and by who. According to [3, page 86], the following is the most essential to include in the documentation:

- Location, date, time, witnesses
- System information, including manufacturer, serial number, model, and components
- Status of the computer, such as whether it was running and what was connected to it
- Physical evidence collected

2.3.3 Decision: Live- or Post Mortem Analysis

When arriving at the scene the investigative team should ask themselves if the system should be unplugged right away, or if a live analysis should be commenced. The investigator should consider if volatile data like memory and open network sockets are vital to the investigation. If the target system is a server providing data for thousands of users, while being hacked into, it may not be feasible to unplug the system right away. Also the investigative team would be wise to obtain crucial information such as the IP address of the perpetrator before powering down. Whether to proceed with the live analysis or unplugging has to be constantly re-evaluated in this case since more damage can be inflicted the longer the system is left on. If however there is a significant risk to loosing data if the machine is left on, a post mortem analysis might be the best option. An example of this is where a disgruntled employee is charged with stealing sensitive data, and is connected to the suspect system remotely. Leaving the system on in this case could give the suspect the opportunity to remove the evidenciary files. Another alternative would be to cut off network traffic at the outer perimeter. This would prevent the perpetrator from doing damage while still being able to monitor intranet traffic. It the responsibility of first responders to make the decision whether to unplug or do a live analysis, and to make that decision quickly.

2.3.4 How to Power Down a Suspect Computer

While a computer is running, and after photographs have been taken for documentation, the decision has to be made as to how the computer should be powered down [3, page 86], assuming a post mortem analysis. As stated by [3, page 86], using the operating system to power down the computer can pose a risk [3, page 86]. The reason for pulling the plug and not powering down the system through the operating system is due to the fact that the operating system or third party hardware or software might alter the contents of the drive, such as deleting temporary files or modify time stamps, as it powers down [3, page 86]. According to [3, page 86], the current best approach is to unplug the computer from the power source, since this preserves the system state as it was when power was cut.

2.3.5 Create Forensic Copy

E-evidence must be protected by putting it into a digital container so that it cannot be tampered with. Possible alterations to the meta data like when a file was last opened, must be prevented. This helps secure *admissibility* of evidence, meaning the acceptance of evidence into court proceedings, while also ensuring fair treatment of the accused. A digital container is an exact duplicate, usually referred to as an image or a forensic copy, of the computer or device [3, page 87]. The investigator then works on the image, while the original hard disk remains securely locked up [3, page 87]. All work must be done to the forensic copy, never the original. Another form of protection is to use write blocking devices or software. By powering on the computer the operating system can make changes to the system during boot time, while third party software or hardware can potentially destroy or alter evidence. Therefore the suspect system should, according to [3, page 86], never be turned on without write-blocking devices. The exception to this is a system being securely booted with a trusted Linux system, in which the suspect system can be mounted manually in read-only mode, instead protected by the operating system, instead of other methods or devices [4, page 154].

Acquiring evidence from a disk larger than the storage medium can pose a challenge. Luckily, some acquisition tools are able to partition evidence files into several smaller files, while also allowing the output to be redirected to another drive during acquisition.

2.3.6 Copy to a Forensically Wiped Hard Drive

Prior to putting the image of the suspect drive on a hard drive, that acquisition hard drive has to be forensically *wiped* clean of all prior data [3, page 90]. Forensically wiping a hard drive means that all areas of the disk are replaced with a single character, usually 0, thus overwriting every file ever stored on the drive [3, page 90]. If a drive has not been forensically wiped before an image is written to it, that image can be tainted by residual data left over on the drive [3, page 90]. After the forensic investigator has ensured that the drive is wiped, the forensic image is copied onto the wiped disk.

2.3.7 Compute CRC and Hash

After the image has been copied to the wiped hard drive, the accuracy of the copying process should be verified. Accuracy is essential, and to ensure accuracy, bit-stream imaging software rely on mathematical cyclic redundancy check (CRC) computations to validate that the duplicate is exactly the same as the original [3, page 91]. According to [3, page 92], the bit-stream of the acquired data is compared to the bit-stream of the original media by CRC validation processes.

It is according to [3, page 82-83] necessary to verify that the acquired media is completely copied. To accomplish this the acquired image is "fingerprinted" and compared to the original media using an encryption technique called hashing [3, page 91], which is also referred to as cryptographic hash verification. *Hashing* ensures the integrity of the acquired evidence since any alteration of the data can be detected [3, page 91]. Hashing generates a unique digital signature for the data, called a message digest (MD). If even one bit of data has been modified, such as a 0 changed to 1, the resulting hash value will differ greatly from the hash value of the original drive [3, page 91].

2.3.8 Evidence Searching and Analysis

Once the forensic copy has been made, the data is searched and analyzed [3, page 91] on the acquired image. The original media is required to be placed in a locker, while the evidence log, refer to 2.3.10 on the facing page, is updated.

Tool kits often provide the user with the feature of bookmarking data, so that important evidence can easily be tagged for later inspection. The key to effective data searches is careful preparation and planning [3, page 92], and to document each search enabling the investigator to easily see which searches have not yet been conducted. To maximize search results [3, page 92] propose that the following tasks be conducted:

- Interview members of the IT staff to learn how and where data has been stored, if applicable.
- Confirm or define the objective of the investigation.
- Identify relevant time periods and the scope of the data to be searched.
- Identify the relevant types of data.

- Identify search terms for data filtering, particularly words, names, or unique phrases to help locate relevant data and filer out what is irrelevant. Meta data can be invaluable to the filtering process.
- Find out user names and passwords for network and e-mail accounts, to the extent possible.
- Check for other computers or devices that might contain relevant evidence.

Filtering out irrelevant data an important process. One of the most effective ways of accomplishing this is using hash databases, refer to 2.1 on page 7 for details regarding hash databases and how they work.

After evidence has been bookmarked, and perhaps even in the very early stages of evidence searching the investigator may form an opinion of what might have happened. This may also lead to more efficient searches if the investigator is right. It is common to do a timeline analysis, sometimes also referred to as a *MAC analysis*, if the case requires event reconstruction. Timeline analysis looks at the order of events by inspecting when files are modified, accessed and when meta data has been altered. A timeline analysis may also contain data from log files, network traffic dumps and other relevant temporal data. The abbreviation MAC, as in MAC analysis, stands for modification time, access time and change/creation time. It is important to interpret MAC times differently on different operating systems, since the letter 'c' can cause some confusion. In Windows 'c' stands for creation time, while in Unix it means change time. Change time describes when the meta data of the file changed, e.g. file permissions or ownership, and is different to modification time since modification time refers to when the contents, not the meta data, was changed. Since the term MAC analysis can be confused with a MAC address assigned to a network interface, it is probably better to use the term timeline analysis instead.

2.3.9 Event Reconstruction

The event reconstruction phase would have to be considered optional, depending on the nature of the perpetrated crime. As mentioned in section 2.2.4.3 on page 12, electronic trespassing, e.g. a "hacker" breaking into a computer remotely, would clearly be an event worth reconstructing, both to obtain inculpatory evidence, and know how to prevent such occurrences in the future. Our example regarding child pornography in the same section, would be an example where event reconstruction, most likely would not be needed. Event reconstruction deals with correlating evidence from many different parts of the system, often incorporating evidence from memory forensics, network forensics and disk forensics into a consistent timeline of events.

2.3.10 Documentation

Proper documentation is considered good scientific practice. Even though this category is referred to last, constantly updating the backlog is very important. As mentioned in 2.3.8 on the preceding page this helps the investigator conduct efficient searches. In addition, documentation is essential if the investigator should be called upon to testify in court. By having a detailed backlog the investigator can easily explain what was done to the system. According to [3, page 83], critical evidence may be subject to reasonable doubt if an investigator is unable to reconstruct accurately what steps have been taken during the investigation. If for instance a change has been done to the original media, the investigator can check the backlog for information regarding how this happened.

In addition, it is required to maintain proper chain of custody procedures during an investigation. According to [3, page 68] every case should maintain an *evidence log*:

> A record or evidence log should be kept to show when all items of evidence, such as server logs, computers, hard drives, and disk, are received or seized and where they are located.

In addition to documenting the crime scene, activities and evidence, findings must be documented [3, page 83]. Most investigations also require the investigative team to produce and present a report at the end of the case, in which case proper documentation would be helpful.

As a last note, any conditions that may serve to tinge the results in any way, e.g. a first responder is the brother-in-law to the perpetrator, or the lead investigator has financial interests in getting a not guilty verdict, should also be included in the final report. This ensures that the results are presented in an ethical and credible manner during the court proceedings.

2.4 Where to Look for Evidence

To be able to forensically evaluate how useful different tools are, we need to know where these tools should search, and what they should look for. There are primarily three main categories of digital forensics. There is the more traditional post-mortem disk forensics, there is network forensics and lastly the relatively new field of memory forensics.

2.4.1 Disk Forensics

Based on sources cited in this section, the following list suggest where evidence can hide. This should not be considered a complete list, since only human ingenuity confines the possibilities.

Residual data

- Unallocated disk space
- Slack space
- Orphaned files

Active data

- User files and programs
- Binary files
- Device files and external media

Protected data

- Host protected area
- Hidden and encrypted data
- Compressed files

Automatically stored data

- Swap space
- Log files
- System configuration files
- OS kernel and start-up files
- Library files
- Cache
- Process directory
- Backup- and system recovery files
- Spool files
- Files copied over a network

Internet activity

- Browser history
- e-mail and instant messaging
- Temporary files
- Cookies
- Monitoring software

At the emergence of digital forensics, disk forensics was the sole practice of an investigation. During an IT-autopsy the computer forensic investigator would pull the plug of the system, remove the hard drive, clone the hard drive, and perform post mortem analysis on the cloned image.

Let's start by looking at what directories should be investigated in a forensic investigation of a system installed with the Linux operating system. When conducting a forensic investigation, the entire hard drive is acquired sector by sector, but some directories should get more attention than others. The following directories is considered mandatory investigation material by the textbook on computer forensics [3, page 257-page 258] employed by the Norwegian Police Academy: /bin, /boot, /dev, /etc, /lib, /sbin, /tmp, /usr and /var. While these directories are considered optional: /home, /lost+found, /mnt, /proc and /root.

2.4.1.1 Unallocated Disk Space

The *unallocated disk* space on a hard drive is the area on the drive which is not currently being used by the file system. Unallocated disk space is analogous to the available-, or free disk space on the media. Since the file system is dynamic, in that it changes in size according to use, the unallocated disk space is also changing correspondingly. If a hard drive is full, there is consequently no unallocated disk space to be examined. Within this unallocated disk space are remnants of earlier use, such as deleted files.

Next, we look at the process concerning file deletion. It is necessary to understand how forensic tool kits can recover and inspect deleted data in unallocated disk space. Contrary to common belief, deleting a file or formatting the disk does not purge the file from the hard drive. In the FAT file system, when a file is deleted or the drive is formatted, the reference to the file is removed from the file allocation table by the operating system, but the contents of the file is still very much intact. This is referred to as *residual data* [3, page 88]. Residual data is data that has been deleted but not erased [3, page 88]. This can be compared to deleting your residential address from the phone registry, but this does not mean that your house no longer exists. When a file is removed from the file allocation table, the area that the file occupies is flagged by the OS as free, so that other files can claim this space. According to [3, page 20] this is done by changing the first character of the directory entry file to a special character, but again - this applies to the FAT file system. Even though the area has been marked as free, the data remains there until another file claims the same space, or the media is wiped. This is why recovery of deleted files is possible. Conversely, the opposite of residual data is called *active data*. Active data is data present on the file system, as opposed to being deleted [3, page 93].

According to [3, page 20], only two steps are required to restore data that has been deleted:

1. The file allocation table (FAT) entry for the file must be linked to the particular location in the data area where the file data is stored.

2. The first character of the directory entry file name has to be changed to a legal character, which restores access to the file.

Note that nothing is done to the data area for the file to be reconstructed. This is partly the reason why digital forensic tool kits can inspect deleted files so easily. Deleted files can be a vital source of information, which is utilized by many forensic tools.

The previous example is regarding file recovery in the FAT file system. File recovery in Ext3 however, is a little more challenging. When a file is deleted in an Ext3 file system, the block pointer, i.e. the location or address of the file contents, i.e. data, is erased from the inode in the inode table [11, page 446]. Since the file name and the actual file data is stored separately, the file name can still be retrieved, but because the block pointer is erased, the data seems gone. To recover the file data, it can be useful to find the block group, in which the file name is stored. Considering that the blocks on which files are stored, tend to be allocated contiguously on the drive, the search can often be restricted to the block group [3]. If the file data is not found within the block group, the entire volume may have to be searched. To recover files in the Ext3 file system, data carving techniques might be required. *Data carving* is the process of identifying and recovering files based on some easily recognizable characteristic in a file's data area, i.e. the file signature. Since file headers tend to consist of the same structures and primitive data types for a given file format, file headers are good candidates for file carving. If the file was recently deleted, the journal of the Ext3 fs may contain a backup of the inode associated with the deleted file, and may still contain a valid block pointer. Consequently, the current inode can be replaced with the inode from the journal, restoring the file, and carving can be avoided.

In the case where a was file deleted, and subsequently partially overwritten by a new file, parts of it may still be recovered, and should not be ignored [3, page 93]. How much, and what portion of the file that can be recovered depends on the size of both files, where on the hard drive the new file is stored, and what techniques are employed by the investigative team. Partially recoverable files, called *partial files*, can help identify motive or intent, passwords, addresses, assets, or other information shedding valuable light on the investigation [3, page 93].

From a forensic perspective, a format is comparable to file deletion. The file allocation table entries are replaced by zeroes, but again, the data area is left untouched [3, page 21]. There is one exception though. If a low level format is conducted, usually done from BIOS, the hard disk is padded with zeroes. This results in the contents of the hard disk being lost, at least through recovery by software.

Even after data has been erased completely, Gutmann [20] claims that data remanence [4] may still be discovered and employed for data recovery using a specialized hardware technique called electron microscopy.

2.4.1.2 Slack Space

A prerequisite for understanding what is meant by slack space, is knowing how data is stored persistently on a storage medium, like a hard drive.

In Linux files are stored by being allocated and written to one or several contiguous groups of sectors on the hard drive called *blocks*. For those more familiar with Windows terminology, blocks are the same as clusters [11, page 409]. How many blocks are needed to store a file depend on both the block size and the file size. Only a single file can be stored on a single block at a given moment, but if the file is sufficiently large it can span across several blocks. A block is the smallest logical unit of storage used by the operating system and consists of several contiguous, physical units on the hard drive called *sectors*.

When a file is stored, there may be leftover sectors in the last block, not occupied by the file. This area between files is commonly referred to as *slack space*. The amount of slack space found between each file varies according to several factors. If either the block size is sufficiently big, or the file size is really small, a single block may contain several "unused" sectors. This "unused" space can contain residual data, like remnants of old files or other artifacts of interest to a forensic investigation.

[3] A block group typically consists of 8 192 blocks, narrowing the search scope considerably.

[4] Remanence means the magnetic induction that remains in a material after removal of the magnetizing field.

According to [3, page 50] slack space can contain confidential data like logon names, passwords, fragments of e-mail messages, documents or desktop faxes, and even legacy data, from prior uses of the computer. Slack space, as in unallocated space, can contain almost anything.

One might be inclined to believe that this tiny slack space only consisting of a few sectors after each file, would in sum amount to very little data. According to [3, page 50] file slack can actually involve several hundred megabytes of data. Taking into consideration that this was written several years back and Moore's law, it reasonable to assume that the amount of data in slack space could amount to several gigabytes in 2010. A hard drive with mostly small files will naturally contain more slack space than a hard drive with mostly large files.

Let's look at what devices one can expect to find slack space. File slack potentially exists on floppy disks, hard disks, zipped disks, and other computer storage devices [3, page 50]. Other examples include mp3 players, cell phones and flash storage devices. Most modern electronic devices used for storing data of an arbitrary size may contain slack space, and thus remnants of earlier use.

So how important is slack space for a digital forensic investigation? According to [2, page 35] slack space is one of the most productive areas of a computer forensics investigation of a computer.

Two Types of Slack According to [3, page 88] there are two types of slack, RAM slack and drive slack, which is also referred to as file slack in [4, page 65]. This thesis will use the term file slack.

RAM slack is the area from the end of file to the end of a sector, except for the last sector [3, page 88]. Since no less than a sector-worth of data can be written from memory to disk, it is the RAM slack portion of the file that is most likely to contain artifacts from memory.

If additional sectors are needed to fill a block, a different type of slack is created called *file slack* [3, page 88]. File slack can be found in the ending sectors needed by the operating system to fill the last cluster assigned to the file [3, page 88], and can consist of data that was created a long time ago [3, page 88]. File slack is created when a file is saved to disk [3, page 88], and can contain multiple partial files. If the file is deleted, it is still available for forensic examination along with its associated slack until the storage space is overwritten by data from another file. Then the overwritten file may then become a part of the new file's file slack.

The difference between RAM slack and file slack, is that file slack consists of partial files already on the disk when the file was stored, while RAM slack according to most literature consists of newly written data taken from memory. Actually, this is not entirely true. Read the paragraph below for more details. RAM slack is more volatile than file slack since RAM slack is always stored prior to file slack. Figure 2.1 illustrates how RAM slack and file slack is stored directly after the file.

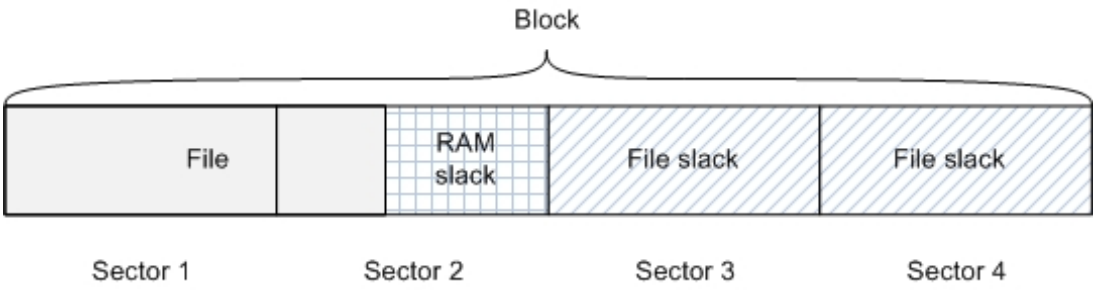

Figure 2.1: RAM slack and file slack

Note that a file may not necessarily contain slack at all. This would be the case if the file size is a multiple of the block size. If the file in figure 2.1 on the previous page was exactly four or eight sectors big, the file would contain no slack. Also, it is possible to have only file slack, and not RAM slack, if the file size is a multiple of the sector size. This would happen if our file in figure 2.1 on the preceding page had a size corresponding to exactly two sectors. Finally it is possible to have only RAM slack without file slack if the file uses portions of the last sector in the last cluster. That would correspond to a file utilizing three and a half sectors in figure 2.1 on the previous page.

A Prevalent Misconception Regarding Slack There are numerous references to RAM slack in literature, most of which are only partly correct, and some downright wrong.

First is a quote from Computer Forensics and Privacy [2, page 34] written in 2001:

> Windows will never write less than one clusterful of data onto a cluster; if it only needs to write half of it, it will mark the end of where that file ends (the "end of file" mark) and fill the rest of the cluster with whatever data happens to be floating about in portions of the computer's electronic memory (random access memory, or RAM).

This was true before the introduction of an upgrade for Windows 95 in 1997 called Windows 95B, which also is known as OSR2. According to [4, page 65] Windows 95B was the first operating system that padded the bytes after EOF with zeroes instead of filling it up with data from memory, effectively rendering RAM slack erased.

Next is a quote from the book used for educating Norwegian Police Academy students in Digital Forensics called Computer Forensics - Principles and Practices from 2006 [3, page 50], published nine years after Windows started padding RAM slack:

> It is important that you understand file slack because it could hold data dumped randomly from the computer's memory.

This quote is more general in that it does not specify Windows explicitly, but rather insinuates operating systems in general. It may hold true that some versions of operating systems still do not pad RAM slack, but we would consider it unlikely. Also, the source states that file slack can contain remnants from memory, when it is in-fact RAM slack that used to be filled with artifacts from memory, and not file slack.

Another quote from Computer Forensics - Principles and Practices [3, page 88] reads:

> When the data file does not fill the last sector in the cluster, the operating system fills the "slack space" with data from the memory buffers.

So we decided to find out for ourselves. Let us begin by explaining the problem. When data in memory is about to be written to disk, the operating system has to write at least a full sector-worth of data, regardless of how little data is about to be stored. If a memory resident file has the size of 1 byte, and the sector size is 512 bytes, the operating system still has to write the remaining 511 bytes. The end of the file is marked with an EOF. The 511 bytes following the EOF is not part of the file system, so they are not observable to the average user. A problem regarding this is that the remaining 511 bytes may contain the password you just used to log in to your Internet bank account, or other confidential information which just happens to be floating around in memory.

We took a test image of a Windows XP partition and inspected sector by sector using EnCase Forensics. Apparently, the area behind many files contained 00 all the way to the next sector. Contrarily, the next sector within the same cluster (same as block in Linux), usually contained data, as would be expected.

What this means is that RAM slack, the area between EOF and EOS (end-of-sector), no longer gets filled with random data from memory. Instead, at least in Windows XP, RAM slack is padded with zeroes when something from memory is written to disk. The area between the first EOS after EOF, and EOC (end-of-cluster), which is file slack, is not padded, meaning old fragments of files can still be found within file slack. In practice, this implies that the password used to log in on your Internet bank account is a little safer than what most literature claim. The password can still get on your hard drive, but according to our knowledge this has to be either because remnants of the swap file are found in file slack, which contain swapped out memory, or if the browser is crashed or killed, and remnants of the dump file containing the password is located in slack space. Even if the browser process is killed, most banks on the Internet run a java applet for login purposes, meaning that the password is probably contained within the java applet, and not the browser. This is something that still needs verification, but we believe that killing the browser in this case would not compromise the password, at least not through the browser dump file.

2.4.1.3 Orphaned Files

Even though some UNIX systems are considered one of the more stable platforms, most systems crash once in a while. This can be due to a software failure, a hardware failure, or quite simply a power outage. Either way, at the next boot a file system check utility called fsck will try to recover any corrupted files it finds. Any orphaned files resulting from the crash will be placed in the /lost+found directory. An investigator could try to look through this directory as a last ditch effort hoping to find useful information.

2.4.1.4 User Files and Programs

When investigating a suspect system, looking for user specific information can yield significant results. According to [3, page 257] the /home directory is a good place to begin looking for user specific files.

In addition to looking for user specific files, programs installed by the user can be a noteworthy source of evidence. Anti forensic software can be installed on a suspect system, making it even more important to search for user installed programs. The directory /usr is the contains all user, i.e., non-system, programs and associated data [3, page 258].

2.4.1.5 Binary Files

The /bin directory contains executable binary files that are needed to boot and/or repair the system. If electronic trespassing is suspected the directories /bin and /boot may provide evidence of tampering [3, page 257]. The /sbin directory contains executable binary files for system/administrative utilities, and is according to [3, page 257] considered mandatory investigative material for an investigator.

2.4.1.6 Device Files and External Media

Files found in the /dev directory, commonly referred to as Linux device description files, can provide information of interest to an investigator. The /dev directory contains several character devices, which are unbuffered files used for exchanging data between serially attached devices, such as a keyboard, serial mouse, or modem, and the CPU [3, page 256]. In addition the directory contains multiple block devices which according to [3, page 256] is defined as buffered files used for exchanging data between random-access block devices, such as hard drives and floppy drives, and the CPU.

According to [3, page 257] the /mnt directory is the mount point for file systems such as hard drives and partitions, and removable media.. However this is only partly true, since it depends upon the Linux version and the system configuration. On Ubuntu version 2.6.31-22 the default mount point for external media is the directory /media. While booting from a flash drive we have also observed that the flash drive is mounted in the directory /cdrom. This mistake was probably due to the fact that the flash drive contained a live-cd image. The most correct approach would be to always check the documentation for each specific system regarding where external media is mounted.

In Linux several useful commands are available for finding out what devices are available, which are mounted, and what file systems they are running. To get a list of currently mounted device files and file system permissions type "mount" from the command shell. For information regarding what device files contain which file systems, and which of these device files contain a bootable partition, type "sudo fdisk -l" from the shell. Finally, to access information regarding both internal drives and external media, like their size, brand, model and which device files represent that disk, execute "sudo lshw -C disk" from the terminal.

2.4.1.7 Host Protected Area

An HPA is an area of the hard disk that vendors use for storing support data [9]. In addition to storing support data, it be used for hiding data from an investigator [9]. It is a protected portion of the disk inaccessible for the normal users unless specific forensic techniques are applied. Albeit, a savvy computer user or forensic specialist might be able to disable this access restriction.

The sectors in the HPA are situated after the sectors that a user can access [9]. As stated in [9], there are three ATA commands that are used to detect and remove the protection mechanism on the HPA. One such command is the IDENTIFY_DEVICE command, which reports the maximum user addressable sectors of the disk [9]. Another command is READ_NATIVE_MAX which returns the total number of sectors on the disk [9]. The exception to this if there is a Device Configuration Overlay (DCO), in which case there are still more hidden sectors after this value [9]. By comparing the output of IDENTIFY_DEVICE and READ_NATIVE_MAX, and observing that these values are different, we can detect if the hard drive contains a host protected area.

When a forensic investigator extracts a normal 'dd' image from the disk, this image will not contain this HPA portion unless the protection mechanism is disabled. To remove the HPA, the command SET_MAX_ADDRESS ATA can be executed, modifying the maximum user addressable sector [9]. By doing this the investigator will get access to all sectors on the drive, including the HPA. The HPA can be disabled either permanently or temporarily. Disabling it temporarily will lead to the protection mechanism being reactivated when the device is rebooted, which is sufficient for the acquired image to include the entire media.

2.4.1.8 Hidden and Encrypted Data

Data can be hidden, encrypted or password protected to thwart anyone trying to retrieve sensitive or incriminating information [3, page 93].

If a file is encrypted using password protection, it can be cracked by utilizing password cracking software. Tools often have different available approaches to cracking passwords. Some tools work by brute force attack; others by dissecting the encryption key [3, page 94]. A *brute force* attack works by trying out vast amounts of possible combinations of words and letters, until the correct password is found. *Dictionaries*, which are files filled with words, are often utilized to make the brute force attack more efficient, often employed along with permutation techniques. *Permutation* works by shuffling letters of a word, or by replacing letters in a word

by other characters. For instance, the word gold could be replaced by g01d, where the letter o was replaced by a zero, and the letter l by the number one. A tool that dissects the encryption key is according to [3, page 94] required with complex encryption, which takes more time than a brute force attack.

If files are hidden by renaming the file extension making them appear as other files, an examination of the file header will reveal what kind of file it really is. Most tools support this feature. The process of hiding something by making it appear as something else, or hiding it within other data is referred to as *steganography*. Steganography in the digital world is most often employed by hiding a message or data within a picture or sound file, by some sort of bit-by-bit manipulation, such as least significant bit insertion. Additionally, if looking for sensitive information leaking out of a secure environment with highly sophisticated access control, like a government computer, data may be transferred through a covert channel. A *covert channel* is a communication channel established between two processes or communicating parties which is not allowed by the computer security policy to communicate. By using non-conventional means of transferring information, e.g. avoiding normal reads and writes, covert channels often remain undetected. An example of a covert channel would be to modify the TCP/IP protocol, so that the sequence number of each packet represent a character, e.g. 1 representing the character a, 2 means the character b, and so on.

The authors would claim that there exists a possibility of hiding a file quite sophistically by encapsulating it within another file, so that the original file header signature is left intact. Doing so would require specially developed software. This would thwart attempts by forensic software to verify the file extension solely by inspecting its file header signature. The manipulation of the original file can be discovered using hash verification if the file used for encapsulation is a well known system file, or more precisely, if a hash value associated with the file is in the hash database. A savvy person would therefore encapsulate the file in file which is not common, and thus not likely to be in the hash database.

2.4.1.9 Compressed Files

Compressed files are encoded making them appear as random characters, rendering the contents indiscernible. If applying a string search matching the contents of a file contained within the compressed file, it will not generate a hit. Most tools, nowadays, at least EnCase and PyFlag, has the ability to automatically detect the compression along with its version, and apply the correct decoding algorithm. This way the contents of the file can be read.

2.4.1.10 Swap Space

The swap space is a portion of the hard disk used for temporary storage of data that would normally be stored in volatile RAM, but is too large to fit there. Any data that could be placed in RAM, could also end up in the swap file. Linux has two forms of swap space, the swap partition and the swap file. The swap partition is an independent section of the hard disk dedicated to swapping, meaning that no other files can reside there. An alternative to the dedicated swap partition is the swap file, which is a single file used for swapping. To find out what kind of swap space is used on a system type the following command from the command shell:

```
swapon -s
```

Since the swap file greatly varies in size consistent with the increase in memory requirements, the swap file may assumably generate file slack, just like any other file.

2.4.1.11 Log Files

Every time a user logs on to the network an electronic trail is left behind [3, page 94]. Network usage, typically containing information regarding who was logged on and when, and how long the user was on the system, is logged by the computer [3, page 94]. Information may also contain who modified a file last and when the modification was made [3, page 94-95]. Additionally, the computer log may indicate when and by whom files were downloaded, and where the downloaded files were stored [3, page 95]. If a file is printed, copied or deleted this may also show up in logs [3, page 95]. What often meets the forensic analyst are deleted log files, but fortunately they are most often not overwritten, and can hence be salvaged.

In a system there are roughly two kinds of log files, namely application created log files and OS created log files.

According to [2] many software applications have the habit of creating a history file of what a user has done with that application. Forensic tool kits should ideally search for such log files, and notify the investigator of their existence. Some applications create their own directory within /var/log/ for logging purposes.

In addition to software created log files, there are also log files created by the operating system. UNIX systems routinely maintain records of logins and logouts, of commands executed on the system, and of network connections made to the system [18]. In Linux we can find most log files in /var/log/. The following is a list of some interesting log files in Linux:

- /var/log/messages: General message and system related stuff
- /var/log/auth.log: Authentication logs
- /var/log/kern.log: Kernel logs
- /var/log/user.log: User level logs
- /var/log/daemon.log: Running services such as squid and ntpd
- /var/log/mysql.log: MySQL database server log file (binary)
- /var/log/wtmp: Login records (binary)
- /var/log/mail.*: Mail server messages

Common for all log files is that they contain timestamp information for all log entries. This can be utilized to partially reconstruct user behavior. Also, instead of putting all the log entries regarding one category in a single file, the entries are instead stored in different files based on their timestamp. For instance new authentication entries are put in auth.log containing the log entries from the current session, while auth.log.1 contain log entries from previous sessions. Even earlier information can be found in auth.log.2.gz, auth.log.3.gz, auth.log.4.etc and so on. This way of splitting up log entries based on their timestamp across different files is a standard that most logs seem to use.

The message files mostly contain kernel related commands, but may also sometimes contain information on previously uninstalled applications, platform information, operating system type, and operating system version.

The auth.log files (auth.log, etc) contain authentication information. Interesting information that can be found here is successful and failed attempts at getting root access. If someone has typed in su or sudo to try to get root access, it will be recorded in this file regardless of whether the attempt was successful or not.

If the investigator or tool cannot find the log files in the /var/log directory, the file rsyslog.conf should contain the location of the logs. The rsyslog.conf is located in the /etc directory.

2.4.1.12 System Configuration Files

There exists numerous interesting files in the /etc/ directory which can be inspected by a forensic analyst or a forensic application to yield useful information about the suspect.

All accounts created will have an entry in the /etc/passwd file. By inspecting /etc/passwd one can find the account ID (login name), user ID (UID), group ID (GID), the location of the users home directory, the login shell, and account information which is typically the full name of the user. Unless the system is configured not to use the /etc/shadow file for storing encrypted passwords, the encrypted password can even be found in this file. However, since /etc/passwd can be read by anyone, most Linux systems today are configured to put the encrypted password in another file, namely the /etc/shadow file, which requires root access to be read. According to [3, page 258] the /etc/shadow file is only found if the installation has been configured to use shadow passwords.

The /etc file also contains other potentially useful information like xinetd configuration, welcome message, and message of the day.

Usually the home directory is in /home/<account ID>/, but this is not always the case. The home directory can be changed to an arbitrary location. If it is changed, inspecting /etc/passwd can yield the real location of the user's home directory.

2.4.1.13 OS Kernel and Start-up Files

By inspecting the /boot directory one can find information regarding the kernel and other files needed for system start-up.

2.4.1.14 Library Files

The executable files in /bin and /sbin use library files in the directory /lib. Library files can be altered to redirect system calls, changing the expected behavior of the system, which is why the analyst brings his own copy of statically linked binaries. Root kits can sometimes make changes to the library files to remain undetected and obtain control over the system. According to [3, page 257] this directory is considered mandatory for an investigator to search through.

2.4.1.15 Cache

The cache contains information regarding current and routinely utilized data, to minimize the number of page faults directed to the hard drive, making the system more efficient. Cache can be utilized in conjunction with other stored information on the system to visually reconstruct a web site recently visited, giving the user a complete or nearly complete visual image of the site. If the relevant data is present in cache and in other parts of the system, this can be accomplished without going online. Since cache contains individual blocks, and not necessarily complete files, the data contained in cache can seem pretty chaotic, but nevertheless this information can be beneficial to an investigation.

2.4.1.16 Process Directory

The /proc directory is a virtual directory, meaning it is not actually stored on the disk, containing special files representing the current kernel state [3, page 257]. This kernel state contain details of the system hardware and any processes that are currently executing [3, page 257]. The /proc/ directory contains a directory for each process with the PID of the process as the directory name. Interesting information can be found here, such as the command line written

in the terminal to start executing the process, which may help identifying the process. This can be found by opening /proc/<PID>/cmdline. If an program, like an editor is started from the terminal to open a document, the cmdline will contain both the name of the file opened and the program used to open it. Since scripts and aliases can be used to start a program, it would be considered safe to check the /proc/<PID>/status file, which also contains the name of the application associated with the current process. The directory also contains a linked directory to the current working directory. By going into the /proc/<PID>/cwd directory the contents of the cwd are displayed. By inspecting the file /proc/<PID>/limits one can observe helpful information regarding the running process. One interesting thing to notice is the priority of the running process, as an abnormally favorable nice priority could indicate the presence of malware. The /proc/<PID>/maps file contains a list of libraries used by the process. If a known bad library is found one can grep the /proc/<PID>/maps file to check if the library is used by the process in question.

2.4.1.17 Backup- and System Recovery Files

According to [3, page 94], some software developers incorporate automatic backup features into their software. The backup features create and periodically stores copies of the file while it is being worked on [3, page 94]. This is especially common with office applications like Microsoft word, and open office.

In addition some operating systems store backup copies used for system recovery if the computer should crash. In addition, some systems store its state when crashing, or stores a reloadable image when the screen saver is activated. These features can possibly yield evidence.

2.4.1.18 Spool Files

As stated in [2], when a user issues the command to print a file, this file is "spooled" to a queue. This queue is stored as a file on the hard disk. After the queue is sent to the printer, and has been printed, the file is removed from this queue. As we have learned previously, a delete is not the same as erase (2.4.1.1 on page 21, meaning that tools can recover this file if it has not been overwritten. Spool files are located in /var/spool.

2.4.1.19 Files Copied over a Network

File copies over the network, are on some system saved to the user's hard drive, rather than to the network file server [3, page 94]. Consequently, if a file has been erased from the file server, this file may still exist as a copy on the user's hard drive [3, page 94].

2.4.1.20 Browser History

Getting access to the browser history can provide useful knowledge regarding the Internet activities from a potential suspect machine. Where the browser history is stored, varies according to the brand and version of browser. Information regarding previously visited web sites can also be obtained by inspecting cookies, as written in 2.4.1.23 on the next page. Documentation should be checked for the particular browser.

2.4.1.21 E-mail and Instant Messaging

E-mail and instant messaging is a valuable source of potential evidence for investigations. Not only do e-mail and IM create a more permanent record than most users realize, the computer

also tracks, in multiple locations, when the messages were sent and opened [3, page 94]. A single e-mail can be saved on numerous servers and computers belonging to *men-in-the-middle*, unknown to the original author, on its path to the intended recipient [3, page 94]. Email and instant messaging are often considered more informal means of communication. Consequently users tend to utter opinions more freely. According to [3, page 94], these aforementioned characteristics make e-mail an excellent source of evidence [3, page 94].

2.4.1.22 Temporary Files

There are basically two kinds of temporary files in a Linux system. The directory /tmp is used for storing temporary files which are not preserved between reboots, which could, according to [3, page 258], be a rich source of evidence if it has not been recently cleaned. The other kind of temporary files are stored in /var/tmp. The difference between these kinds of temporary files, and the temporary files stored in /tmp, is that the files stored in /var/tmp is less volatile, as it is preserved between reboots. When conducting first response, it would be wise to make a copy of these temporary files, as they are almost as volatile as RAM. Note that temporary files can be inspected to yield information regarding both Internet and network activities, as well as locally performed actions.

2.4.1.23 Cookies

Evidence can be hidden almost anywhere. In the DFRWS 2008 challenge [16], contraband information was split up into several chunks and sent over the network in cookies, as a part of the network protocol itself.

If employing Mozilla Firefox in a Linux environment using Ubuntu, one can for instance find information regarding which web-sites have been visited in the /root/.mozilla/firefox/cookies directory. Even this is related to network activity, it can still fall under the category disk forensics since the cookies themselves are stored on disk. If the cookies are captured in-transit on the network, it would be as network forensics. By inspecting the cookies.sqlite-journal and the cookies.sqlite files we were able to find many earlier visited websites on the experimental workstation. The files themselves are in binary format, but in between the binaries are ASCII text of URL addresses.

2.4.1.24 Monitoring Software

In some countries it is getting more and more common to install software software designed to monitor employees' use of company computers [3, page 95]. According to [3, page 95] this kind of software records information such as programs used, files accessed, e-mail sent and received, and Internet sites visited [3, page 95]. If an investigation regarding a business employing monitoring software is initiated, the investigative team should utilize this monitoring software in their investigation. Also, the legality of using monitoring software should be researched if such a system is discovered.

2.4.2 Network Forensics

According to [15, page 1], network forensics refers to the forensic analysis of captured network traffic. Cohen [15, page 1] states that:

> In the simplest case the traffic represents network communication exchanged by a party during the course of some activity.

A prerequisite to conducting network forensic analysis is the availability of a *network dump file*, which typically has the file extension .pcap. This network dump file is obtained and analyzed by a packet analyzer like for instance tcpdump or Wireshark by monitoring the suspect systems network activities. Since the network dump file is just a collection of raw data transferred over the wire, some kind of decoding tool is needed to make reading the raw data more accessible. For this purpose, forensic analysts use a packet analyzer to extract higher level data like the file name and content of the file transferred, or lower level data like which protocol and port was used to transfer it. The packet analyzer tools usually also has some kind of filtering to make searching for specific types of network traffic more readily available. Say, for instance, that one finds a file transfer application using the ftp protocol installed on the suspect system. Then one might want to filter the network traffic on ports 20-22 to find any files transferred by this application. When conducting network forensics, logs from existing IDS/IPS/Firewalls/routers/Netflow collectors should be obtained, or at least frozen.

According to Cohen, 2008 [15, page 2], a network forensic tool should have the following properties:

- Efficiently process very large capture files. The system should ideally be scalable (i.e. more hardware leads to faster processing).
- Extract high level information, for example, files transferred, social networks (who talked to whom) or keyword indexing.
- Be able to substantiate each deduction, clearly demonstrating how the deduction can be cross checked using first principles.

2.4.3 Memory Forensics

In addition to forensically investigate the hard drive and network, a relatively new branch of digital forensics is appearing, namely memory forensics. Just a few year ago, common practice was limited to performing a brute force string search through memory. This approach might yield some evidence, but will also be largely insufficient and pretty random. A better approach would be to navigate the memory resident structures to reconstruct the processes and files. J.M. Urrea suggests an

2.5 Forensics Equipment

Hardware requirements are determined by the type of investigation and data to be analyzed [3, page 106]. All digital forensic labs should have a wide assortment of power supplies, USB 2.0 and FireWire cables, power cables, electrostatic mats and spare expansion slot cards, in addition to workstations and software [3, page 107]. Hard disks are necessary equipment, since e-evidence is generally copied to hard disks [3, page 107]. Additionally, electronic evidence should be archived to DVDs immediately after acquisition, because hard drives can fail [3, page 107]. Disaster recovery plans should be in place, specifying how to rebuild an investigation workstation after it has been severely contaminated by a virus [3, page 107]. Consequently, redundant data storage devices are required. According to [3, page 107] software requirements include all operating systems on the market including all Microsoft operating systems, all OS X versions of Apple Macintosh operating systems and Tiger, and Linux operating systems, including Fedora, Caldera Open Linux, Slackware, and Debian. Although high-end computer forensics tools are able to open or display most data files created with popular software, not all programs are supported [3, page 108]. According to [3, page 108], the software inventory of a digital forensic lab should include Microsoft Office products, Intuit's Quicken, the Corel Office Suite, StarOffice/OpenOffice, Peachtree's accounting software, programming languages such as Visual Basic and Visual C++, and specialized picture viewing software [3, page 108].

2.5.1 Portable Hard Disk Duplicators

2.5.1.1 Disk Jockey

Disk Jockey is a portable hard disk duplicator with write-blocking features essential for computer forensic examinations [3, page 109]. It is available as part of a forensic kit including a serial-ATA (SATA) adapter and longer cables for connecting to hard-to-reach drives inside a computer [3, page 109]. Disk Jockey is considered a Swiss-Army-Knife type of product due to its versatility, and can be used for several operating systems connected via high-speed FireWire or USB 2.0 ports, or as a stand-alone device [3, page 109]. It is used to mirror, span, copy, compare, and test hard drives [3, page 109].

2.5.1.2 Logicube

According to [3, page 109], Logicube is a leader in hard drive duplication, data recovery, backup, and computer forensic systems, and is used in many Fortune 500 companies, and by most government and law enforcement agencies.

2.6 Analysis Environments and Modes

2.6.1 Trusted Environments

Some tools for examining hardware need to be used in a trusted environment (or trusted computing environment), typically in a lab [3, page 95]. Trusted environments have dedicated systems set up for the sole purpose of conducting computer forensics investigations [3, page 95]. In these labs, there is no threat of malicious programs or security risks to the data [3, page 95].

2.6.1.1 Analysis Mode: Postmortem Analysis

A dead analysis refers to an examination of a suspects computer or device performed on a dedicated computer forensics analysis system, or workstation [3, page 96]. This is the most utilized form of analysis mode utilized. The term dead analysis is used because the suspect computer is not running when the analysis is done in the lab [3, page 96]. This type of analysis is also referred to as a postmortem analysis because the crime or incident has already happened [3, page 96]. Only nonvolatile data can be acquired from a dead analysis [3, page 96]. Nonvolatile data is data that still exists after the computer is turned off or the power is removed [3, page 96]. Another term for nonvolatile data is persistent data [3, page 96].

2.6.2 Untrusted Environments

Occasionally, a computer cannot be unplugged. This can be due to several reasons. During electronic trespassing, such as a "hacker" breaking into a system, the investigation requires that sufficient evidence regarding the crime is collected, prior to unplugging the system. Another example would be a service critical cooperate server that has to keep running due to financial implications.

2.6.2.1 Analysis Mode: Live Analysis

According to [3, page 96], if volatile data must be acquired, that being data which is lost when the power is turned off, a live analysis is needed, but this is not necessarily the case. Volatile data can encompass several things, be it memory, logs that are overwritten by a system boot, cache, and so on. If volatile data such as RAM is the primary interest of the investigation, a post mortem memory analysis can be performed as an alternative to live analysis. However, this requires that an image of the memory file is acquired prior to unplugging. A live analysis occurs when the suspect system is analyzed in its own environment while it is running [3, page 96]. In cases such as network intrusion or other crimes in progress, it is vital for a forensic investigator to do a live analysis as quickly as possible [3, page 96]. These on-site locations are untrusted environments [3, page 96]. A live analysis would be needed during a hacker attack or other intrusion to confirm the compromise while it was occurring [3, page 96].

2.7 Admissibility of Evidence

Frequently, investigators have to defend their findings, processing methods, tools, and techniques against challenges raised by the opposing side [3, page 111]. Therefore, e-evidence processing must be done correctly and documented thoroughly or else any resulting court case may be thrown out [3, page 111]. A key principle is that the technologies and methodologies used must be well documented and repeatable [3, page 111].

A judge's acceptance of evidence in a trial is referred to as admission of evidence [3, page 44]. At a minimum, evidence admissibility requires a legal search and seizure - usually with a search warrant or court order - and a chain of custody [3, page 44]. E-evidence must follow the three C's of evidence: care, control, and chain of custody [3, page 67]. These are legal guidelines to ensure that the evidence presented is the same as that which was seized [3, page 67]. It requires documentation of the maintenance of evidence in its original state and preparation for civil or criminal proceedings [3, page 67]. The chain of custody is, in effect, documentation that the evidence was handled and preserved properly and that it was never at risk of being compromised. According to [3, page 44], the documentation must include:

- Where the evidence was stored
- Who had access to the evidence
- What was done to the evidence

Each piece of original evidence that is seized should have a chain of custody log associated with it [3, page 68]. Computer forensics tools such as MD5 (message digest) or MD5sum create a unique value for each file [3, page 66]. If the file is changed in any way, its MD value also changes [3, page 66]. This tool helps ensure that the files have not been altered and can be admitted into evidence [3, page 68].

Without a documented chain of custody, it is impossible to prove after the fact that evidence has not been altered [3, page 44].

It is critical that forensic investigators and tool developers are aware of this, so that electronic evidence will be accepted into court proceedings. Preferably tools used for digital forensic purposes should feature an option to automatically create a report containing this information.

2.8 An Introduction to Tools

Before concluding this chapter, and proceed to evaluation criteria selection, we take a brief look at tools being currently employed on the Linux platform. Recognizing capabilities of other tools serves as a foundation for reporting lacking, but desirable features in EnCase and PyFlag later in this thesis.

2.8.1 Two Categories

Digital forensic tools can be classified into two main categories - acquisition tools and analysis tools. An acquisition tool makes a copy of an observed state, while an analysis tool defines primitive states based on the copy [10], meaning that the acquisition tool creates an image of the device, and that the analysis tool interprets an image. A tool featuring both acquisition- and analysis capabilities, it is often referred to as an analysis tool, even if it is also capable of acquisition.

2.8.1.1 Acquisition Tools

An acquisition tool is used for extracting an image from a device. This is accomplished by copying a predefined number of blocks from the device, thus filling up the buffer, and subsequently appending those blocks to the acquired image, emptying the buffer. This process is repeated until the device is completely acquired, and the result is a file, called an image, containing a bit-stream copy, or an exact duplication of the acquired media. An analogy would be someone scanning a book composed of texts from many different languages. The book is scanned page by page, while the person disregards its contents and meaning. The person doing the scanning does not need to know what languages the book is written in, which is fine, since the results can be interpreted by someone else knowing the required languages. In our case that person would be the analysis toolkit.

Now, imagine that the book being scanned is an open source project stored electronically on the Internet. Imagine that this project has hundreds of thousands of editors, and is very likely being edited while the copying takes place. Some pages will be added, and some deleted during the course of acquisition. The book or device is not imaged or copied as it was during a specific point in time, but instead over the course of several hours. This is actually what happens when the forensic investigator extracts an image from RAM.

An acquisition tool should not only be able to take an image from hard drives, but also acquire memory. Additionally, tools used for *packet sniffing*, meaning the acquisition of network traffic, are also considered acquisition tools.

2.8.1.2 Analysis Tools

Analysis tools range from relatively simple tools restricted to operating on one category of medium, like for instance a disk image, to composite tool kits providing a framework for integrating several tools into one package. The latter has the advantage of being able to correlate evidence from across different components of a computer, like memory, hard drive and peripheral devices.

Since one of the tools to be evaluated will be a hybrid toolkit, meaning it is both able to acquire evidence and analyze it, the evaluation criteria will contain requirements for both acquisition tools and analysis tools.

2.8.2 Acquisition Tools

2.8.2.1 dd

One of the most frequently employed acquisition tools on Linux, is a tool called dd. dd works similar to a copy operation, the difference being that a cp, copies only logical data, while dd copies physical data as well, in effect creating a duplicate of the blocks allocated to the device file, and not just the data which included in the file system. It is command line based, and can be run from an ordinary shell.

There are several advantages of being command line based. Firstly, having no GUI means that it utilizes less memory space. A small *footprint*, meaning how much memory it needs, is especially important when conducting memory forensics. When using software based acquisition tools, the program extracting the memory image will have to be loaded into memory in order to run. This results inevitably in some situations where evidence residing in memory is being overwritten. A smaller footprint keeps damage to a minimum.

Secondly, since 'dd' is command line based, it is possible to utilize the shells other capabilities in conjunction with the tool. 'dd' sends data to Standard Output if the 'of=' flag is not given [7], meaning it can be piped to other applications. Hence, it is possible to send the output of 'dd' directly over the network to an acquisition station using the pipe command combined with for instance netcat or cryptcat.

Another advantage of using 'dd' is that it is fast. The default block size for 'dd' is 512-bytes, meaning it will read 512-bytes from the input file and write them to the output file [7]. Block size used by 'dd' during the acquisition can be changed by setting the 'bs=' flag. Specifying block sizes in the range of 2k to 8k will generate faster acquisitions than using the default size [7].

If 'dd' comes across an error while reading a block from the input file, an error will be generated and the copying process will stop [7]. You can cause 'dd' to keep on going when it encounters an error if you provide the 'conv=noerror' flag [7]. As stated in [7] this will cause all the subsequent data to be written in the wrong location, but is easily corrected by telling 'dd' to pad the areas with an error with zeros, which is done by adding the flag 'sync'. As claimed by [7] this should only be done if there are known errors since this could cause excessive padding if the block size is not a multiple of the original media.

The output of dd is an exact copy of the input file [7]. This format can be used as input to most of the popular analysis tools [7]. This gives dd great flexibility with regards to what tools are being used later in the analysis phase.

Finally, dd is relatively simple to use.

A disadvantage of using dd is that the acquired image cannot be partitioned into several smaller files. This is fine if the storage drive is formatted with EXT3 or EXT4. Conversely, if the storage drive is formatted with FAT32, the size limitation of 4 gigabytes can be a problem. However, this is a minor disadvantage, since the first responders should be aware of this size limitation, and should not use FAT32 when acquiring evidence using 'dd'. The solution is simply to bring an acquisition drive with for instance EXT3.

2.8.2.2 dcfldd

The U.S. Department of Defense Computer Forensics Lab (DCFL) has released an updated version of 'dd' that includes the ability to calculate the MD5 while the data is being copied [7]. According to [7] the dcfldd tool has the option of either calculating a hash for the entire image, or partition the image into several hash windows. Say, for instance that you would want to calculate a hash value for each 2MB chuck of data in the image. Then you could add

'hashwindow=2M' to the command line while executing 'dcfldd'. If you would want to have a hash value for the entire image, using hashwindow=0 is sufficient.

As stated in [7] 'dcfldd' has the option of saving a hash log in a separate file. A hash log is simply the collection of hashes created using 'hashwindow'. This is done by adding the flag 'hashlog=' flag followed by the path which the log should be save to.

The difference of using 'dcfldd' compared with 'dd', is that using 'dd' one would have to execute separate procedures to obtain the hash values. This is not the case with 'dcfldd' which provides a fast method of calculating MD5 hashes of an image.

2.8.2.3 sdd

'sdd' is a variant of 'dd' and has at least one useful feature that 'dd' does not have. Instead of just reporting the number of blocks that were copied in and out and a 1 or 0 value to identify if the last block was not complete, 'sdd' actually gives the number of bytes copied and how much of the last block was copied [8]. Here is an example of an incomplete final block using 'sdd':

```
# sdd if=/dev/sda1 of=dev/bull bs=4k
sdd: Read 205625 records + 3072 bytes
(total of 1052803072 bytes = 1028128.00k).
sdd: Wrote 205625 records + 3072 bytes
(total of 1052803072 bytes = 1028128.00k).
```

The same using 'dd':

```
# dd if=/dev/sda1 of=/dev/null bs=5k
205625+1 records in
205625+1 records out
```

As we see 'sdd' not only tells us that the last block was not copied completely, but how many bytes were actually copied. If we know how much has been copied we can easily adjust the block size, 'bs=', so that the rest can be added to the image. This saves us from extracting the entire image all over again.

2.8.2.4 fmem

With the introduction of the Linux 2.6 kernel in December 2003, a new kind of protection mechanism made its appearance. This new protection mechanism prevents 'dd' from accessing parts of the memory file, even with root access. In response, a new tool called 'fmem', being a kernel module, was developed to allow forensic investigators to create an exact copy of the /dev/mem file in spite of this protection.

2.8.2.5 EnCase Portable

EnCase Portable is a software solution stored on a USB drive specifically designed for non-technical personnel to extract forensic evidence from computers in the field. By inserting it into the computer, the device automatically extracts and stores pictures, Open Office documents, browser history and artifacts, and other digital evidence, even entire hard drives.

2.8.3 Analysis Tools

2.8.3.1 dtSearch

For combining through large amounts of data, dtSearch leads the market with support for over 250 file types [3, page 102]. Huge collections of files can be searched quickly once a document index has been created [3, page 102]. An index is a database that stores the location of every word in a collection of documents, and each word's corresponding location throughout [3, page 102]. Results are listed and highlighted with their exact locations [3, page 103]. Results also can be exported to a format recognized by Microsoft Excel for reporting [3, page 103].

2.8.3.2 AccessData FTK

Forensic Toolkit (FTK) from AccessData is designed for finding and examining computer evidence [3, page 101] and is a tool currently in use by several crime investigation units across the globe, including Norwegian police departments. FTK has full-text indexing, deleted file recovery, and data-carving [3, page 101]. It contains tools for searching, filtering, and analyzing e-mail and zipped files [3, page 101]. FTK includes FTK Imager, the Hash Library-KFF, and Registry Viewer [3, page 101]. FTK can be used to acquire images; to read images acquired using other systems, such as the EnCase system and to view numerous file formats [3, page 101]. It includes the tool dtSearch for full-text indexing and searching [3, page 101].

Notably, FTK can scan a disk for text strings, utilizing these as input to a dictionary attack for cracking encryption passwords.

2.8.3.3 Ultimate Toolkit

Ultimate Toolkit (UTK) is primarily for computer crime investigators [3, page 101]. It contains all AccessData's recovery modules, which are the FTK, Password Recovery Toolkit, Registry Viewer, 100-client Distributed Network Attack, WipeDrive Professional, Microsoft NT Utility, and Novell Utility [3, page 101]. UTK contains components for recovering lost or forgotten passwords, analyzing and decrypting registry data, and wiping hard drives [3, page 101]. A dongle is required to use the product [3, page 101].

2.8.3.4 WinHex

WinHex is a universal hexadecimal editor, used for computer forensics, data recovery, low-level data processing, and IT security [3, page 102]. It can be used in Linux when executed through Wine. It is an advanced tool used to inspect and edit all types of files [3, page 102]. It can recover deleted files or lost data from hard drives with corrupt file systems, or from digital camera cards [3, page 102]. WinHex capabilities include drive imaging, file analysis, and forensic disk cleaning [3, page 102]. WinHex can be used to inspect and edit all kinds of files, recover deleted files, or retrieve lost data from hard drives with corrupt file systems [3, page 102].

2.8.3.5 iDetect

iDetect is a memory acquisition- and analysis tool specializing in extracting information from a Linux memory image. It has features ranging from displaying the contents of memory mapped files, to displaying detailed information about each active process. In addition, iDetect is capable of being used on a live system.

2.8.3.6 Selective File Dumper

Selective File Dumper is an open source forensics tool able to retrieve files of a selected type, either from an image, or directly from a device. It utilizes TSK (see section 2.8.3.11 for more information regarding TSK) for retrieving deleted files. Additionally, the tool features simple keyword searching capabilities.

2.8.3.7 Second Look

Second Look is a commercially available analysis toolkit, also being able to acquire memory from Linux systems, which can be done locally or remotely through direct memory access (DMA), or across a network.

2.8.3.8 The Coroners Toolkit

The Coroners Toolkit, abbreviated TCT, is a collection of command line based security applications, which later founded the basis for TSK, refer to 2.8.3.11. Its purpose was to assist in forensic analysis after a computer break-in.

2.8.3.9 EnCase Forensics

EnCase is a hybrid toolkit, both being able to acquire- and analyze evidence from Windows, Linux and OS X. EnCase Forensics is one of the evaluated tool kits in this thesis, and is discussed in detail in 4.1 on page 55.

2.8.3.10 PyFlag

PyFlag is an open source tool capable of evidence analysis on both Windows and Linux systems, and is one of the evaluated tool kits in this thesis. More on PyFlag in 4.2 on page 64.

2.8.3.11 The Sleuth Kit / Autopsy

The Sleuth Kit, abbreviated TSK, is a both a stand alone digital forensic analysis tool, and integrated into PyFlag for file system analysis. More details on TSK in 4.2.8 on page 75.

The Autopsy Forensic Browser [5] is a graphical interface to the command-line analysis tools in the Sleuth Kit. Autopsy and TSK runs on UNIX platforms, including Linux, and supports FAT, NTFS, Ext2/3, and UFS file systems [11, page 15] [3, page 102]. With these tools, file systems and volumes of a computer can be analyzed, including Windows and UNIX disks [3, page 102]. Autopsy is based on HTML so it can be accessed from any platform using an HTML browser [3, page 102]. Additionally, Autopsy provides a file manager-like interface displaying details for deleted data and file system structures [3, page 102].

[5] www.sleuthkit.org/autopsy

Chapter 3

Test Specifications

3.1 Basis for Selecting Test Specifications

To be able to evaluate digital forensics tools it is important to take into consideration what is meant by digital forensics, and the processes involved in digital forensics. The various definitions of digital/computer forensics in 1.4 on page 2 and The Process of Digital Forensics in 2.3 on page 15 serve as a good foundation. Furthermore, the tool needs to be able to extract, discover and utilize evidence found in various places. The section Where to Look for Evidence in 2.4 on page 20 serves as a suited starting reference. Test specifications will be based partly on the aforementioned requirements, along with some requirements from 3.1.1.

Considering that this thesis evaluate two of the most promising digital forensic tool kits available, it is necessary to define what makes a tool good. A good tool satisfies a significant amount of the criteria selected in 3.2 on the next page, and is versatile, efficient, reliable and easy to use, with support for most modern operating systems and file systems. It is also able to discover and analyze most types of evidence.

Conversely, a poor tool, is a tool that is limited in its feature list, slow, difficult to use and with only support for a single operating system and a few file systems. It often has problems extracting and discovering various types of evidence.

3.1.1 Additional Basis

In addition to the aforementioned requirements, Brian Carrier proposed several criteria as a requirement to digital forensic analysis tools; Usability, Comprehensive, Accuracy, Deterministic and Verifiable [5]. *Usability* is a requirement imposed to solve the complexity problem, which is the fact that data at its lowest format is too difficult to analyze. The complexity problem is solved by providing data to the user from a layer of abstraction, formatted in such a way that it is easier for the investigator to comprehend. He continues by claiming that a tool must be *comprehensive*, which means the investigator should have access to all output data from the different layers of abstraction. An example would be having access to view a selected file in hex dump while traversing the file hierarchy. Furthermore, *accuracy* is a criteria imposed to solve the error problem, which introduce errors into the final product because of the abstraction layers. The error problem is solved by ensuring that output data is accurate, and that a margin of error is calculated, facilitating the correct interpretation of the results. A tool has to be *deterministic* to ensure that the same output is produced when provided with the same translation rule set and input. Finally, an analysis tool has to be *verifiable*, meaning that other analysis tools should produce the same results. In addition to these requirements, Carrier propose a recommended feature, which is that a tool should be *read-only* [5].

According to the book used by the Norwegian Police Academy in educating its students in digital forensics, there is a set of objectives which must be achieved by an investigation. These objectives can be summarized as protect, discover, recover, analyze, report, and provide an opinion of the results [3, page 48-49]. First, the computer system, devices, files and logs must be protected during the forensic examination from alteration, damage, data corruption and malware. Second, all files must be discovered, including all active-, deleted-, hidden-, password-protected- and encrypted files. Third, the contents of discovered files should be recovered to the extent possible and made accessible. This includes deleted files, hidden- or temporary files, swap files, password protected files and encrypted files. Fourth, steganalysis methods should be employed to reveal the existence of, and potential locations of steganography. Fifth, analysis of data found in hard-to-reach areas of the disk has to be conducted, including unallocated space and slack space. Sixth, an overall analysis of the system is printed as a report, containing information regarding discovered and relevant data. Finally, the investigator should provide an opinion of the system layout and evidence found, including attempts to hide, delete, protect or encrypt potential incriminating evidence, and be accessible for expert consultation.

Additionally, [3, page 97] states that tools should support the investigator perform several tasks during the investigation. The tool should be able to recreate specific chain of events, or sequence of perpetrator activities, including file deletions, e-mail communication and Internet activity. The tool should also be able to find key words, key dates, and help determine which data is relevant. In addition, tools must help the investigator search for previously stored copies of documents, or document drafts, privileged information and installed programs. Finally, tools should help authenticate file- and temporal [1] data.

3.2 Selection of Test Specifications

Next, we present the proposed criteria for evaluating the two tool kits, PyFlag and EnCase. In this section, each stated criteria will include one or more arguments for why it should be included as an evaluation criteria. If there are doubts to whether it should be included as an evaluation criteria, arguments against including it will also be stated. If a criteria is not selected for evaluation and testing, either due to time limitations, scope or irrelevance, it is proposed how the criteria could be tested.

3.2.1 Acquisition

3.2.1.1 Wiping

As stated in 2.3.6 on page 18, wiping a hard drive means replacing the contents of an entire hard drive with a single character, usually 0. Wiping of the storage disk should always be executed prior to acquisition. Wiping tools exist as open source and proprietary software, but the tools should be verified forensically before employing such tools. Wiping a hard drive is an important task, since evidence can be tainted by leftover data on the acquisition hard drive.

One the one hand, it would be beneficial for the user to be able to forensically wipe storage media using a familiar environment, i.e. the toolkit. On the other hand, there are several third party tools available that specialize in the wiping of electronic media. It is by no means a necessity for a digital forensic toolkit to have this feature, which is why wiping will not be included in the final evaluation of PyFlag and EnCase.

[1] Temporal data means time related data, i.e. MAC times.

3.2.1.2 Network Acquisition Capability

The toolkit should have the capability of sending the image, as it is acquired, across a network, either directly by being integrated into the toolkit, or indirectly through piping the output from a shell, if the toolkit is command line based.

By being able to send the acquired evidence directly over the network, there is no need to mount external hard drives or USB devices on the system. Mounting additional hardware causes changes in the state of the system, like the update of logs, loading of resources into memory and so on. Modifying the file system only slightly, may overwrite existing evidence, changing the state of the system, and thus may delete important evidence. By bringing a safe lab machine to the site, a network cable can be connected to both machines, and the evidence can be acquired. This may assumably lead to minor changes in the system state as well, since logs may be updated, and buffers in memory will have to be filled before sending the packets. It is, however, much quicker to connect a lab machine to the target machine, than it is to open the cabinet and plug in an acquisition hard drive. Also, it may be necessary to acquire over the network when confronted with a laptop. The laptop may have proprietary security schemes marrying the hard drive to the motherboard, making traditional acquisition more difficult [4, page 123]. By interconnecting the two systems using a network cable in this scenario, the target hard drive inside the laptop may be acquired.

Considering that there already exist tools capable of network acquisition, e.g. piping the output of dd to netcat or cryptcat, there is no crucial need for for network acquisition in Linux. Additionally, due to the scope of this thesis, and the limitations in hardware resources, having only a single computer available, this will not be tested or included in the evaluation.

3.2.1.3 Peripheral Devices

The tool kits should be able to reliably and persistently acquire all individual files and data stored on all external devices connected to the system. All residual data, including deleted files and partial files should be contained within the acquired image. The acquired image of the peripheral device should be an exact duplicate.

Even though acquiring peripheral devices is an important feature, this can be accomplished through third party acquisition tools, and is therefore not included in the evaluation.

3.2.1.4 Content Acquisition

The tool kits should be able to extract and contain all available content, from all sectors on the acquired device, and the number of sectors acquired should be verified.

Acquiring all sectors from a drive is imperative, and is why this criteria is both tested and included in the final evaluation. The number of sectors acquired are compared to the sectors reported to be in the acquired partition. Since PyFlag does not feature acquisition capabilities, dd is used for PyFlag, while LinEn is used for EnCase.

3.2.1.5 Swap Acquisition

The toolkit should be able to acquire swap. As noted in 2.4.1.10 on page 27, swap can contain artifacts swapped out from memory due to memory saturation. Thus, acquiring swap can give the forensic analyst an insight into the state of the system, like running processes, open files and such, at the time prior to acquisition.

Swap acquisition is closely related to memory forensics, which is outside the scope of this thesis. Therefore, swap will not be tested during this thesis. On the other hand, string searching could be performed on swap without using memory forensics, but due to time limitations, this test will have to be omitted.

Swap acquisition could be tested by creating a small C application running a for-loop malloc-ing [2] an easy recognizable string. The iterator in the for loop should run so that about four to five gigabytes of memory is allocated, depending on the amount of memory installed. This would force some memory to be swapped out. The acquired swap image could subsequently be string searched looking for a match.

3.2.1.6 Preservation and Verification of Original Data

In the court of law, one of the most fundamental principles regarding evidence handling is that the evidence has not been tampered with. This is why maintaining a strict chain of custody during an investigation is critical.

The tool should be able to forensically preserve the original media while evidence is acquired. Ideally, the tool should ensure that the media is mounted in a read-only environment. Since this is usually handled by blocking devices or the operating system, this can pose a challenge.

At a minimum, the toolkit should be able to verify that no changes to the original media has occurred. This is done by image "fingerprinting", using cryptographic hash verification techniques. A single changed bit on the original media, will result in an *avalanche effect*, meaning that the hash value produced will contain a significantly different hash value (e.g., half of the bits flipped), making it easy to identify if a device has been tampered with.

In addition to preserving the original media, the acquired evidence should also be preserved while it is being investigated. It should be possible to verify that no changes has occurred to the acquired media during the investigation. To ensure that no tampering has occurred, the forensic investigators are required to provide the court with two hash values for each device. One cryptographic hash verification from before the evidence was acquired, and another acquired after acquisition. By comparing the hash values from the original media to the duplicate media, the validity and admissibility of the evidence is ensured.

Preserving the original data is an important characteristic, and will therefore be tested and included as an evaluation criteria as part of this thesis.

3.2.1.7 Acquisition Format

If the tools are used for acquisition as well as analysis, the tools should be able to store meta data in the evidence file. This helps organize the evidence in a logical and protected manner, making sure that the correct piece of evidence is archived with the right case.

This is an important feature of a forensic toolkit and will be included in the evaluation. However, no testing on this criteria will be conducted, since the documentation provides information regarding acquisition capabilities and supported formats.

3.2.1.8 Host Protected Area

As described in 2.4.1.7 on page 26, some hard drives contain a protection mechanism called HPA, hiding several sectors on the disk from view. A tool should ideally be able to detect an HPA residing on the hard drive, and extract the data it contains. At minimum, it should

[2] malloc stands for memory allocate, and is a command in C used for reserving and allocating memory to data.

be able to discover and display the amount of sectors found on the hard drive, so that the existence of the HPA can be verified by looking up the hard drive specs on the manufacturers website, and comparing them with the number of sectors reported. If there is a difference in the number of sectors, the analyst can assume that an HPA is present.

The authors found results indicating the presence of a Device Configuration (DCO) on both the acquisition media and the storage media. A DCO is a protection mechanism hiding the presence of an HPA, and making the disk look smaller than it actually is [11, page 53]. Each hard drive was inspected using the Sleuth Kit tool disk_stat and dmesg from the command shell, but no HPA was found. Also, the maximum user sector was reported significantly larger than the maximum disk sector indicating the presence of a DCO. Due to the fact that there are few tools able to detect and remove DCO [11, page 53], the authors decided to abstain from HPA testing in this thesis. Also, since the storage and target systems are both hard drive partitions instead of entire hard drives, HPA and DCO protection will be insignificant in this test. However, the capability to acquire HPA data is important, which is why this criteria will be included in the evaluation. However, information regarding HPA testing capability is solely based on documentation, and not testing.

3.2.1.9 Residual Data

It should be determined if the tool can recover deleted files and partial files. Finding residual data is arguably one of the most important features of a forensic analysis toolkit. It enables the forensic analyst to excavate previously deleted files. The longevity of deleted files alternate, from files that was deleted just prior to the suspects computer being confiscated, to files deleted long ago. This source of evidence is considered invaluable to an investigation.

Considering that evidence found in unallocated space can be of critical importance, this criteria will both be tested and included as an evaluation criteria in this thesis. However, partial files in slack space is harder to produce, and as such, only unallocated files are tested.

3.2.2 Automation

3.2.2.1 Scripts

As previously stated in 2.1 on page 7, a tool needs to be able to automate tasks. Acquisition and analysis takes time. It is not uncommon for an analysis or acquisition to take several hours to complete, depending on the size of the data. For this reason, there is a need for automating several tasks pre-programmed to run in succession. Scripting is a form of automation, allowing the user to queue and configure tasks prior to execution. More importantly, scripts allows tasks being performed over night, leading to more efficient utilization of time. Scripting is therefore a desirable, if not required asset in a forensic toolkit.

It would be considered beneficial if the toolkit provides pre-programmed scripts for the most common tasks. It also, however, should allow the user to program customized scripts, using a scripting language. The ability to create customized scripts makes the scripting significantly more powerful.

Scripts are important in providing an automated solution to many tedious forensic processes. However, considering scope and time limitations, scripting will not be tested in this thesis.

3.2.2.2 Automated Report Generation

Digital forensic tool kits should be able to automatically generate reports. This should include information regarding acquired media and the findings of the investigation. It should be presented in an easy to understand format, since the report have to be presented to non-technical personnel like jury members or customers. The Report should be stored in an electronic format, allowing the reviewer to click files, and see their content. Moreover, content should preferably be available for inspection in their native format. Including a picture or office file would be little use if the reviewers has to inspect binary code. It cannot be expected of the tool kits to support the native viewing of every possible format, but the most common ones should be supported.

However, manually written reports can be customized to a much greater extent, but will take considerable time completing. On the other hand, automated report generation saves time, and allows the investigator to append notes and comments to bookmarked files as evidence is found. Additionally, reports are often in a standardized format, at least within one toolkit, allowing reviewers to quickly get an outline of the findings. Considering the statements above, automated report generation is to be both tested and included as an evaluation criteria in this thesis.

3.2.3 Evidence Searching

3.2.3.1 Hash Analysis

By filtering out irrelevant information, the toolkit can greatly reduce the amount of data needed to be manually inspected. One way of accomplishing this is by utilizing a hash database (see 2.1 on page 7).

A toolkit should have the ability to connect to a database with precomputed hash values of common and known files. By calculating the hash values of every file on the computer, and comparing them to the hash values stored in the database, the tool can classify and filter files. Files known to contain malicious code, or child pornography can be flagged as suspicious, while common files, like ordinary kernel files, can be excluded from the list of files displayed to the forensic analyst. This form of data reduction facilitates a more efficient and targeted approach.

Conversely, a disadvantage of employing a hash database, is that a single, wrongfully classified file can compromise and invalidate the entire investigation. The hash database needs to be proven, widely employed, extensive and trusted.

As derived from the statements above, the benefit of employing hash databases for hash analysis far outweigh the potential downside. Consequently, hash analysis will both be tested and included as an evaluation criteria of the two tool kits during this thesis.

3.2.3.2 Tagging or Bookmarking Functionality

Being able to tag or bookmark files and folders for later inspection is important in digital forensics. The toolkit should be able to organize the bookmarks or tagged evidence in a lucid manner. This can be accomplished by organizing evidence into folders, or by attaching a category label next to each item.

The authors will not perform any testing regarding this feature. However, bookmarking is essential to creating automated reports, so it will be examined and utilized during testing, and will also be included as an evaluation criteria.

3.2.3.3 File Signature Analysis

The tool should be able to recognize misnamed files by inspecting the file header information, instead of file extension. It should be able to find files that have been misnamed with a misleading extension.

Tools that excavate, sort and identify files based solely on their file extensions are by nature insufficient and fallible. File extensions are by no means a sure way of identifying a file. It requires only modest knowledge of computers to change the file name or file extension, which is why forensic tool kits should consistently identify files based on file headers. However, Linux does not utilize the file extension for deciding which program should open the file. Instead, Linux use the file header for this purpose.

Considering that anyone can rename a file, file signature analysis has to be considered an important feature of any forensic toolkit. Consequently, file signature analysis will both be tested and included as an evaluation criteria.

3.2.3.4 Timeline Analysis

To be able to reconstruct the string of events on a suspect system, timeline analysis is critical. As previously stated in 2.3.8 on page 18, timeline analysis is used for deciding when changes to the file system occurred, e.g. when a file was modified, or when it was last accessed. It would be beneficial if the MAC times are presented in an easily graspable format, preferably through a graphical illustration. By making MAC times readily available to the user, the toolkit would facilitate a better insight into the activities performed on the computer.

Considering the importance of timeline analysis, this feature will be tested and included as an evaluation criteria.

3.2.3.5 Graphics File Formats

The tool should be able to detect, and preferably display a compilation of pictures or graphics acquired. Graphics should be presented in an organized manner.

This functionality will be explored in sections 4.2 on page 64 and 4.1 on page 55, but will not be explicitly tested in either toolkit. However, pictures are downloaded and viewed as part of testing automatic report generation.

3.2.3.6 E-mail and IM

It should be determined if the tool kits are able to find exchanged e-mail message information, and instant messaging communication. Because e-mail and instant messaging often contain informal communication, acquiring this type of evidence can be especially productive. Tool kits should contain support for the most commonly employed formats of e-mail and instant messaging.

Considering that e-mail communication is stored automatically to disk, provided the e-mail client is running locally, e-mail will be extensively tested during the course of this thesis. On the other hand, IM sessions are not automatically stored to disk. Considering that IM clients will only store chat sessions if logging is enabled, and network forensics is outside the scope of this thesis, IM will not be tested or included as an evaluation criteria.

3.2.3.7 Browser History

Tool kits should be able to extract the browser history and visited web sites from suspect systems, which can be accomplished by extracting data from the browser cache and history files. In addition to visited web sites, search expressions typed in Google or other search engines are possible to extract, since these search expressions are stored as an URL. In [4, page 92], it is stated a real life scenario, involving a suspect having been found with an open safe inside his residence. The following URL was found on his computer:

> http://www.google.com/search?sourceid=navclient&ie=UTF-8&oe=UTF-8&q=how+to+crack+a+safe

Considering the aforementioned importance of acquiring and analyzing browser history, this feature will both be tested and included as an evaluation criteria.

3.2.3.8 String Searching

String searching, also known as keyword searching, works by processing the contents looking for a matching string, like a word. String searching is the most commonly employed searching technique for memory resident data, but is also widely utilized in disk forensics. This form of analysis can yield significant results, and are often utilized in conjunction with indexing, which is discussed in 3.2.3.9.

String searching is a key data discovery feature, and will consequently be included as an evaluation criteria and as well as thoroughly tested for the evaluated tool kits.

3.2.3.9 Indexing

Tool kits should provide an indexing feature, enabling faster keyword/string searching of evidenciary files. Creating an index can take a considerable amount of time, but once done, searching will be much quicker. To understand indexing, consider the following analogy; Just as a person reading a book is able to look up in the table of contents, or index at the back of the book to find a certain topic, and jump directly to the correct page, indexing creates an list of offsets for each indexed word. Thus an indexed search is able to get direct access to the indexed word inside the file without reading the entire file. This is why indexed string searching is so much quicker than for instance grep. Grep will have to read the entire file to find the string being looked for.

Considering the aforementioned reasons, indexing will be tested and included as an evaluation criteria in this thesis.

3.2.3.10 PIM Applications

The tool should ideally be able to extract *personal information management data (PIM)* from calendars, contacts, task lists etc. This should also apply to deleted PIM data. Extracting this information could prove invaluable to an investigation.

To test PIM applications means installing additional software and populating these with data. Due to the broad range of topics included in this thesis, and time limitations, the scope of testing must be restricted. Consequently, PIM applications will unfortunately be excluded from testing, and will not be considered in the final evaluation.

This could have been tested by installing PIM applications, putting some personal information into the system, utilizing calendars, contact lists in e-mail applications and such, and see if the tool kits are able to extract this information.

3.2.3.11 Compressed File Archive Formats

It should be determined whether the toolkit can find text, images, and other information located within compressed-archive formatted files (e.g., .zip or .tar) residing on the suspect system. The tool should be able to detect, display, decompress and look inside multiple layers of compressed files. If there is a compressed file containing a picture inside another compressed file, the tool should be able to display the picture and the name of the file, just as it would, had the picture not been compressed.

The process of *carving*, meaning searching for files or objects based on content instead of meta data. The user should be able to mount compressed files on a global level, utilizing carving if necessary. If each and every encountered compressed file has to be located and uncompressed manually using the toolkit, the result is extensive overhead, i.e. "manual labor", for the examiner.

Being able to decompress data is an important aspect of digital forensics. Consequently, this criteria will both be tested and included in the evaluation for each toolkit.

3.2.3.12 Decryption

A decryption feature would be desirable in a digital forensic toolkit. The tool cannot be expected to decrypt most encryptions, since many modern cipher texts will take too long to decrypt.

It could, however, be feasible to find the key used for encryption if the encryption was performed locally. When utilizing the key for encryption, the key will necessarily be buffered in memory while encrypting, thus making memory a reasonable place to search for the key. In addition, buffering the key in memory can lead to the key being written to RAM slack on older operating systems. Finally, it is a slight possibility that the key actually exists somewhere on the file system itself, or in cache. All these techniques are based on the assumption that the cipher text is encrypted locally.

If the hard drive is encrypted using FDE, or full disk encryption, it is considerably more difficult to decrypt. The reason for this is because software based FDE encrypts the entire hard drive, including the memory file, making acquisition difficult. If it is a hardware or hybrid based FDE solution, it may even encrypt the boot sector. Consequently, the decryption features of a toolkit cannot be expected to decrypt a drive using FDE, unless the key is provided by either the suspect, or is found as a written note in the vicinity of the crime scene.

A more challenging situation arises if files are found on the suspect system encrypted by a third party. The only option may then be to brute force the cipher text for the key. This may prove a difficult and time consuming process, but could work when trying to decrypt some older encryption algorithms using shorter key lengths. It would possibly be more feasible to subscribe to a third party decryption cluster service for such computationally requiring decryptions.

Unfortunately, this criteria will not be tested and included in the final evaluation. Decompressing files take a considerable amount of time, and due to the wide ranged of features covered and tested in this thesis, some features will necessarily have to be omitted from testing.

3.2.3.13 Password Cracking Capability

The toolkit may be able to acquire the users password to obtain the contents of the suspect system. A multitude of stand-alone applications that performs password cracking are available, one of the most prevalent in Linux being John the Ripper. These tools come equipped with several cracking modes, like brute force, permutation and dictionary attacks, and they work

pretty well. One might ask if there is a need for integrating this password cracking capability into existing tool kits.

One one hand it would be practical to collect all desired features needed for a forensic investigation into a single big toolkit. Why? First of all, only a single application needs to be updated, as opposed to updating several smaller tools. This wastes time which could otherwise have been used more productively. Secondly, if the computer forensic investigator is already familiar with the application, it would be easier for the forensic investigator to learn newly introduced features of a known environment, compared to learning an unfamiliar, new tool from scratch.

On the other hand, password crackers work excellent as they are. If password cracking is needed during a live investigation or first response to retrieve a user password, it is critical that the tools used have a small footprint, so that when they are loaded into memory the small size of the tool keeps the damage to potential evidence residing in memory to a minimum. In other words, less is more.

This would lead the authors to conclude that even though a password cracking capability is a somewhat desired feature in a toolkit, the need for small, independent password cracking tools are evident. This holds especially true for live analysis, needing tools with small footprints. The above statements has led the authors to conclude that small, and stand-alone password and cracking applications are still needed in their current form, and that forensic tool kits does not need to feature password cracking. Consequently, password cracking capability will neither be tested or included in the evaluation process.

3.2.3.14 Reconstructing Web Pages from Cache

As stated in 2.4.1.15 on page 29, cache can be combined with other system resources to present the forensic analyst with a complete, or nearly complete picture of previously visited web sites, without actually going online. More importantly, if the web site has been changed since acquisition, or has been taken offline, the investigator can see what it looked like when it was visited by the suspect.

The reconstruction of web pages from cache and network traffic involves network forensics, which is outside the scope of this thesis. Consequently, this feature will not be tested in this thesis or included as an evaluation criteria.

This feature could have been tested by starting a packet sniffer [3] like Wireshark or tcpdump, visiting a couple of web sites, acquire an image of the drive, provide the tcpdump file, and see if the tool kits are able to reconstruct and display the web site without going online.

3.2.3.15 Live System Restoration

A forensic tool should have the ability to restore the acquired image to a hard drive, and run it in a live environment This is useful if the forensic analyst wants to see the system as the suspect did. A requirement for live system restoration is that the hard drive is of greater size than the image or evidence files.

Considering the considerable amount of time it takes to restore a live system to another drive, this criteria will not be tested or included in the evaluation.

[3] A packet sniffer is a network traffic acquisition tool, able to capture data being transmitted over a network.

3.2.3.16 Log Analysis

The toolkit should have support for log analysis. The evaluated tool kits should be able to display log files to the user in a comprehensible and parsed format. To be able to parse logs the tools will need to either feature different parsing schemes customized for each type of log, or be able to import a log used as a template for analyzing logs of that specific type.

Considering the need for limiting the scope of the testing, this criteria will not be tested. However, log analysis is considered important in digital forensics, which is why this criteria will be included in the evaluation. The evaluation will consequently be based on available information in the documentation.

This functionality could have been tested by trying to parse a supplied log, and see if it was displayed in a human readable format.

3.2.4 Expandability

3.2.4.1 Plug-in Support

The tool could benefit from facilitating third party development of plug-ins. It is a way for the community and users to expand and improve the features of a toolkit themselves, instead of waiting for an update that may take a considerable amount of time, or never be realized. However, this can be compensated for if the developers behind the toolkit frequently release updates with added capabilities, and spends time on forums noticing and acting on requests made by the users.

Information regarding plug-in support is readily available from the documentation, which is why this will not be tested. However, facilitating third party plug-ins is an important issue, and will be included in the final evaluation.

3.2.4.2 Open Source

A forensic toolkit should arguably be available for code level inspection. By being able to empirically inspect the source code, the analyst can justify odd results or bugs resulting from use. Additionally, being exposed to third party scrutiny, as open source software code often is, can lead to a more robust, tested and proven toolkit.

A disadvantage, however, of having open source digital forensic tool kits is that suspects may take advantage of the available source code, developing covert techniques for hiding evidence.

The aforementioned facts has led the authors to conclude that the benefits of software being open source, outweigh the disadvantages. Given time, open source software could have a greater potential than closed source projects.

With readily available information regarding which tools are open source and which are proprietary, there is no need for conducting any testing regarding this criteria. However, this criteria will be included in the evaluation.

3.2.5 Performance

3.2.5.1 Computational Efficiency

Computational efficiency, in this context, is referred to the time taken by the toolkit to perform a specific task. If a tool is faster, it is considered more computationally efficient. Since digital

forensic tool kits often work on large amounts of data, some tasks tend to take several hours, or even days. The more efficient a tool is, the more time can be spent on other more pressing matters, making this criteria an important one.

This could be measured while executing various tasks and searches on one tool, and subsequently comparing it to the time taken by the other tool on the same tasks.

On EnCase this is pretty simple to measure, since EnCase reports the time used on task completion, including acquisition, searching and indexing. When measuring computational efficiency on PyFlag, this has to be measured manually, by starting a clock when the task is started, and stopping it when it finishers. When for instance a scanner is run in PyFlag, the window reporting how many tasks are left will simply disappear. This will require constant monitoring to be able to tell when a task is finished.

Since EnCase runs exclusively in a Windows environment, and PyFlag is run on Linux, the operating system may impact the measured times. Additionally, the features in PyFlag and EnCase is implemented very differently. An example would be how images are processed and displayed to the user. In PyFlag, a scanner will have to be executed to make all pictures available, looking explicitly for images. In EnCase, images are shown immediately to the user, but are identified by their file extension instead of file signature. For EnCase to be able to find all images, a file signature analysis for the entire case will have to be conducted. Consequently, PyFlag and EnCase has not completed the same task, which is why this is difficult to measure objectively. Considering that computational efficiency may be different according to the tasks completed, this will not be included as a stand alone criteria. Instead it will be included in the test results where there is a significant difference between the two tool kits.

3.2.5.2 Task Logging

Having a centralized system for logging, integrated into the toolkit, logging all the various tasks performed by the user, would have to be considered a huge benefit for any forensic department. This would relieve the forensic investigator of the mundane task of logging each and every small step during the investigation, leading to extra time for more important tasks. Also it would consistently produce a detailed backlog to see what investigative actions were performed on the evidence.

This criteria will not be extensively tested. However if either or both tool kits perform logging of completed tasks, it will noted in the evaluation.

3.2.5.3 Multi-user Environment

Ideally, several investigators should be able to work on the same case simultaneously. This would facilitate more efficient investigations, but would also introduce problems concerning distributed solutions. To prevent *deadlocks*, that being the case where two or more competing actions are waiting for each other to finish, thus creating an eternal loop, the system would have to be developed according to requirements present in modern distributed systems. Alternatively, a more simple solution would be to use existing technology, such as a database, to ensure ACID [4] properties and prevent deadlocks.

Considering that information regarding multi-user capability is readily available from documentation, neither toolkit is tested for this criteria. However, having a multi-user environment is considered beneficial, and will therefore be included in the evaluation.

[4] ACID are the most significant requirements for databases and stands for atomicity, consistency, isolation and durability.

3.2.6 Extensiveness

3.2.6.1 Comprehensive File System Support

If a toolkit only works with one specific operating system, a typical department dealing with digital forensics has to have multiple forensic tool kits to be able to handle all the different operating systems available. A single, versatile and multi-platform toolkit with support for the most common file systems is desirable.

This information is readily available from the documentation, making testing redundant. However, having support for multiple file systems is important, which is the why this criteria is included in the evaluation.

3.2.6.2 Case Management

A digital forensic toolkit should have a system for creating, organizing, maintaining and retrieving numerous forensic cases resident on a single examination system. For forensic tool kits to be successfully integrated and used throughout law enforcement and corporations dealing with forensics, the toolkit has to have the ability to manage multiple cases at once.

The case management abilities of both PyFlag and EnCase is stated in the documentation. Thus, any testing is needless. Conversingly, because proper case management is an important feature for administrating and maintaining multiple cases, this criteria is included in the evaluation.

3.2.7 Memory Extraction

3.2.7.1 Process Listing

By utilizing the structures in memory, it is possible not only to perform string searching, but in fact, list all the running processes of the system. This can shed valuable light on the current state of the suspect system. If the tool is able to reconstruct the state of processes, this can be used for discovering malware, or other malicious processes.

However, since the scope of the thesis is restricted to disk forensics, this criteria will neither be tested or evaluated during this thesis.

The process listing capabilities of a tool kit can be tested by first starting up a few processes on the system being investigated, noting process ID (PID), time and processnames using the `ps -A` command. Subsequently the suspect systems memory (/dev/mem) is acquired using e.g. fmem and dd. If the tool kit features process listing, the output should be similar to the output of `ps -A`.

3.2.7.2 File Restoration from Memory

By traversing the proper structures in memory, it is possible to reconstruct files mapped in memory, even if they are not saved to disk. This can be useful to inspect recent changes to files not yet stored.

Since memory forensics is outside the scope of the thesis, this criteria will not be tested or included in the evaluation.

3.2.8 Usability

A digital forensic toolkit should ideally be relatively easy to use. Due to the complexity of current tools, this is an absolute necessity. Preferably, a graphical user interface (GUI) should be available. A GUI helps the investigator more easily see the big picture of a case, and can more easily review bookmarked or tagged evidence.

This will not be tested because the authors have no objective way of testing this criteria. However, our subjective experience from working with both tool kits will be stated in the evaluation.

3.2.9 Deterministic

For the acquired evidence to be reliable and considered admissible, the tool has to consistently produce the same results.

The tools should also conform to the same standard regarding hash verification. When comparing the hash-values produced by one tool, e.g. using md5sum, to another tool using the same hash, the values should match. According to [3, page 100], even if both tools use md5 hash, depending on how bad sectors are handled, the different tools can produce different md5 values. The authors assume this is due to different error granularity, meaning how many sectors are skipped if a bad sector is detected. This will have to be tested for verification.

Considering that the only way to test if the tool kits are consistently producing the same results, is by utilizing a toolkit over a significant amount of time, this criteria will not be tested or included in the evaluation, due to the scope and time restrictions of this thesis.

3.2.10 Accuracy

3.2.10.1 Adjustable Timezones

Another important characteristic the tool kits should conform to, is being able to set the correct timezone. When creating a new case it is imperative to choose the correct timezones. This is crucial if wanting to look at events which happened in different timezones, and be able to create a common timeline in which these events took place. This is especially important when working on international investigations, where multiple parties are situated in different timezones, or in domestic investigations in countries spanning across several timezones. A tool should have the ability to convert MAC times from differing timezones into a single, common timezone, as to facilitate more easy correlation of events, and in building a consistent timeline.

This is testable by setting up two suspect systems, each configured with different timezones, establish a network connection between the two parties, and sending a file across the network. By acquiring the logs from each system, which would be in local time, the tool kits should be able to align the differing timezones to a common setting, creating a valid timeline for the communication.

Due to the restricted hardware availability during the writing of this thesis, this criteria is not tested or included in the final evaluation.

Chapter 4

Toolkit Functionality

4.1 EnCase

EnCase Forensics is a proprietary toolkit issued by Guidance Software, and is the de facto standard for digital crime investigation units worldwide. It is a toolkit for gathering and evaluating electronic information, and one of the Department of Defense (DoD) approved tools [3, page 98]. According to [3, page 98], EnCase software is very easy to use due to a well developed GUI. EnCase Forensics has to be run on a Windows platform, but is able to forensically examine evidence from a wide variety of file systems including all Windows file systems, Linux EXT2 and EXT3, Macintosh HFS, HFS+, Sun Solaris UFS and many others.

4.1.1 EnCase Evidence File Format

One feature separating EnCase from most other forensic tool kits is the fact that meta data and information regarding a case, is stored in a tailored format, also containing the acquired image. This format is developed specifically for EnCase. However, because EnCase has increased in popularity, other tool kits, e.g. PyFlag, are also supporting this format.

According to [4, page 180] an EnCase evidence file, called the *EWF* file, consists of three major components: the header, the data blocks, and the file integrity component (CRC and MD5). The header is located on the front end of the EWF file, while the data blocks follow the header [4, page 180]. The file integrity component exists throughout the file, providing redundant levels of file integrity [4, page 180]. According to [4, page 180] each individual data block, including the header, is verified and sealed with its own CRC. The entire section of data blocks is subjected to an MD5 hash function, referred to as an *acquisition hash* [4, page 180]. This acquisition hash is appended after the data blocks [4, page 180]. See figure 4.1 for how the various components in the EWF format are put together.

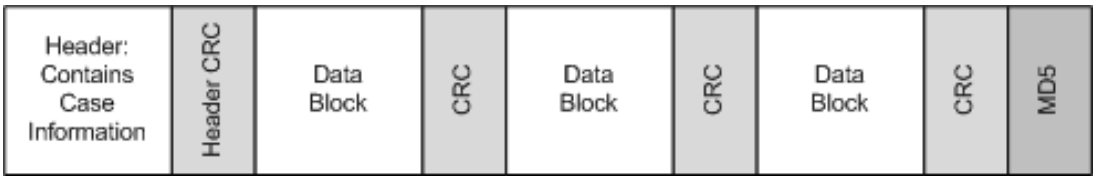

Figure 4.1: The EWF format

According to [4, page 180] the header and all CRCs are not included in the acquisition hash, which is only calculated on the data. This is an important design decision since the acquired

image is to be compared with the original media after the investigation has concluded, as to verify that no changes to either media has occurred.

As previously mentioned in this section, the EWF format facilitates storage of meta data along with the evidence itself. The meta data regarding the case is stored in the EWF header, and includes the evidence name and number, notes that may be of interest, date and time of acquisition, which version of EnCase was used for acquisition, and the OS under which acquisition took place [4, page 180]. By storing this information with the evidence, the EWF format ensures that information regarding the case is always stored with the correct piece of evidence.

4.1.2 Layout and Usability

Since it is concluded in 3.2.8 on page 54 that a good layout is an important evaluation criteria, the most pertinent layout features will be explained below.

The environment available through the GUI in EnCase is context sensitive. This implies that the different options made available through the System menu bar, the Toolbar and by right-clicking, varies according to the context, e.g. differing file types.

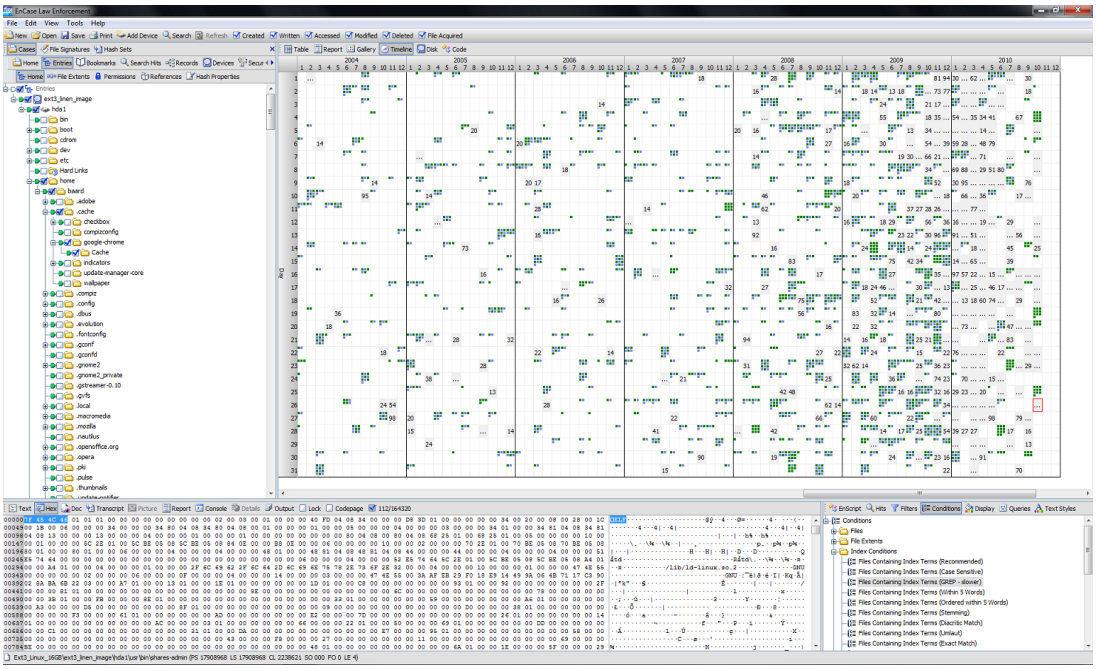

Figure 4.2: A screenshot of the layout in EnCase

4.1.2.1 Panes

EnCase Forensics has divided its screen real estate into four windows named by their primary examination function: the Tree Pane, Table Pane, View Pane and Filter Pane [4, page 210]. See figure 4.2 for the layout in EnCase and the location of the different panes. The Tree pane is in the upper left corner, Table pane in the upper right corner, View pane in the lower left corner, and Filter pane in the lower right corner. The *Tree Pane* is used primarily for navigating through the file hierarchy of the acquired image, and is also used for selecting the scope of a given search. The *Table Pane* is used for inspecting individual files inside a directory selected from the Tree Pane. When a specific file has been selected from the Table Pane, the *View Pane* can be employed to display details about the specified file. As such, the level of granularity

steadily increases when navigating through the primary panes from the Tree Pane, to the Table Pane and finally to the View pane [4, page 210]. As the name implies, the last pane called the *Filter Pane*, is used for filtering data, by utilizing conditions, filters and EnScripts.

4.1.2.2 Views

Besides panes, EnCase features additional ways of inspecting the collected data, simply referred to as *Views*. These provide the user with a specific way of observing data, with the most important being Gallery View, Disk View, Timeline View, Table View, Report View, Hex View, Picture View and Doc View.

The *Gallery View* is accessible from the Table view and presents the user with a collection of all pictures found in the currently selected folder. By right clicking the root folder in the Tree Pane, thus selecting all sub folders and sub files, every single graphics file on the medium is displayed. This is significantly useful when conducting an investigation based around graphical evidence. At the default setting, EnCase identifies pictures by file extension. Consequently, a file signature analysis should be performed prior to inspection, especially if the investigation revolves around illegal pictures. This makes sure that all factual pictures are displayed, even if misnamed.

The *Disk View* is also accessible from the Table view, and is able to display every single sector and block on the hard drive. The user can choose whether to display sectors or blocks. The sectors or blocks are displayed as small consecutive rectangles with varying color according to their allocation status, organized as a two dimensional matrix. E.g., a light blue color signifies that the sector or block is allocated, dark blue that it is unallocated, while a yellow-greenish color that it is part of volume slack. In addition, by right-clicking, it is possible to specify a selected sector or block by number, and more detailed information regarding this sector or block, like file name, its contents and other meta data, will be displayed in the View Pane.

As with the two previous views, the *Timeline View* is accessible from the Table view. By utilizing this view, the user is able to get a calender-style picture of file activity. See the Table pane in figure 4.2 on the preceding page for how it looks. When first entering this view, the MAC times for the last previous months are displayed. By selecting files or folders from the Tree Pane, the user is able to specify which files are to be displayed. The user is able to filter for one specific form of file activity, e.g. file modifications or access times. By scrolling the mouse wheel the user is also able to magnify or zoom in on specific dates and times of day, allowing more detailed and accurate MAC time information.

The default view in the Table Pane is the *Table View*. This view displays a list, organized as a spreadsheet, of all files in the folders selected from the Tree Pane. The files are displayed as rows, while the attributes and properties of each file are displayed as columns. Each file is listed with a vast array of meta data, e.g. file name, physical size, permissions, hash value, full path, symbolic links, and 32 other attributes. Considering the huge amount of information in the table pane, it is extremely helpful to utilize the sorting functionality, along with the ability to hide, move and lock columns. This makes it easier for the investigator to arrange the information in the Table view, so that relevant columns are placed next to each other.

The *Report view* displays very detailed meta data about a single, selected file or object (e.g. file permissions, size, MAC times), and is available in the View Pane [4, page 240]. By right-clicking, the Report view can be exported to a document or web page enabling tailored reports.

The *Hex View* is a hexadecimal content viewer available from the View Pane. It allows the user to inspect the binary contents of a block or sector, displaying individual bytes in a hexadecimal notation on the left side, while presenting the same information as ASCII on the right side. It is used for inspecting raw data.

If EnCase identifies a selected file, sector or block as a picture, the picture is displayed to the user in the *Picture View*, found in the View Pane. Even if a single sector is selected that represents a smaller part of a complete picture, the entire picture will be displayed to the user.

The *Doc View* is useful for displaying files in their native environment, as they were intended. E.g. Open Office documents are displayed as though they were opened in Open Office, .doc or .docx documents as they would be displayed in Microsoft Word and .html files as web pages. The Doc view is accessible in the View pane. According to the official web site for EnCase, EnCase support the native viewing of around 400 different file formats.

There are several other views such as Code view, Text view, Console view, Transcript view, Details view and Output view, but these are not considered significant. This thesis is not concerned with providing documentation for EnCase og PyFlag, only provide an outline of the most significant and relevant features complying with the stated evaluation criteria in 3.2 on page 42. Thus, these views are omitted from this discussion.

4.1.3 File System Support

EnCase is able to analyze an extensive variety of different file system formats, including Windows FAT 12/16/32, NTFS, Macintosh HFS/HFS+, Sun Solaris UFS, ZFS, Linux Ext2/3, Reiser, BSD FFS, FFS2, UFS2, NSS, NWFS, AIX jfs, JFS, TiVo Series One and Two, Joilet, UDF, ISO 9660, Palm and also file systems specific for CD-ROMs (CDFS) and DVDs [11, page 15] [1].

4.1.4 Acquisition

There are primarily four main ways of obtaining evidence files in EnCase. One way of obtaining evidence is available from the system menu in EnCase Forensics, and is primarily, but not exclusively, used for obtaining evidence from a system running Windows. Another way of obtaining evidence in EnCase is by utilizing its *LinEn* application, an acquisition tool specifically designed to obtain evidence from a Linux platform. In addition, EnCase facilitates adding a raw image, not conforming to the EWF format, such as an image taken with dd. Lastly, it is possible to perform a network acquisition.

When acquiring an image in either EnCase or LinEn, the user has the option to choose what level of compression, if any, that is to be utilized. E.g., higher compression results in slower acquisition, while a lower compression ratio yields faster acquisition. The benefit of compression is evidently that the evidenciary files occupy less space on the hard drive, which could be important if available disk space is sparse. It is also possible to select the *error granularity*, meaning the number of sectors that get zeroed when an error is found. In addition the user can define the block size used for acquisition, facilitating more efficient acquisitions. With a larger block size the acquisition software talks less to the device, resulting in faster acquisitions. It is, however, important to use a block size which is able to add up to the exact size of the suspect media. This is important to ensure that no excess sectors are written at the end of the duplicate, which is not present in the original media. This would lead to differing hash values, possibly compromising the admissibility of the evidence. In addition, EnCase is able to restart acquisitions if an interruption occurs. EnCase and LinEn are both capable of calculating CRC and hash values to verify the integrity of both the acquired evidence, and the original media.

If any sectors are not acquired, EnCase will report it. Information regarding missing sectors or read errors can be found under Devices at the case-level views tab in the Tree pane. If any errors are listed here, the exact sectors on the acquisition volume are listed, so that these can be inspected.

By interconnecting the suspect system to a second machine running EnCase, with a network cable, a network acquisition can be performed. The main reasons for acquiring over a network cable using EnCase is according to [4, page 123-124] if you want to acquire invisible HPA or DCO data, data from a laptop drive, if wanting to acquire the data quickly, or preview the data before acquisition.

4.1.4.1 LinEn

The developer of EnCase, Guidance Software, has developed LinEn specifically for conducting acquisition on Linux platforms. EnCase is unable to acquire evidence from a Linux system, which is why LinEn is needed. LinEn is included in EnCase Forensics, and can be utilized by creating a live-cd, boot disk, or bootable USB with LinEn included. This is accomplished by selecting Create Boot Disk from the system menu, and merging a downloaded live-cd image with LinEn. An even better solution, would be to have LinEn stored on a separate flash drive. Since USB flash drives are re-writable, it facilitates a more convenient storage medium for an application that regularly has to be updated. LinEn includes the the same acquisition options, e.g. selecting block size, compression and number of worker threads, as an acquisition performed from EnCase. See figure 4.3 for a screenshot of the GUI using Linen.

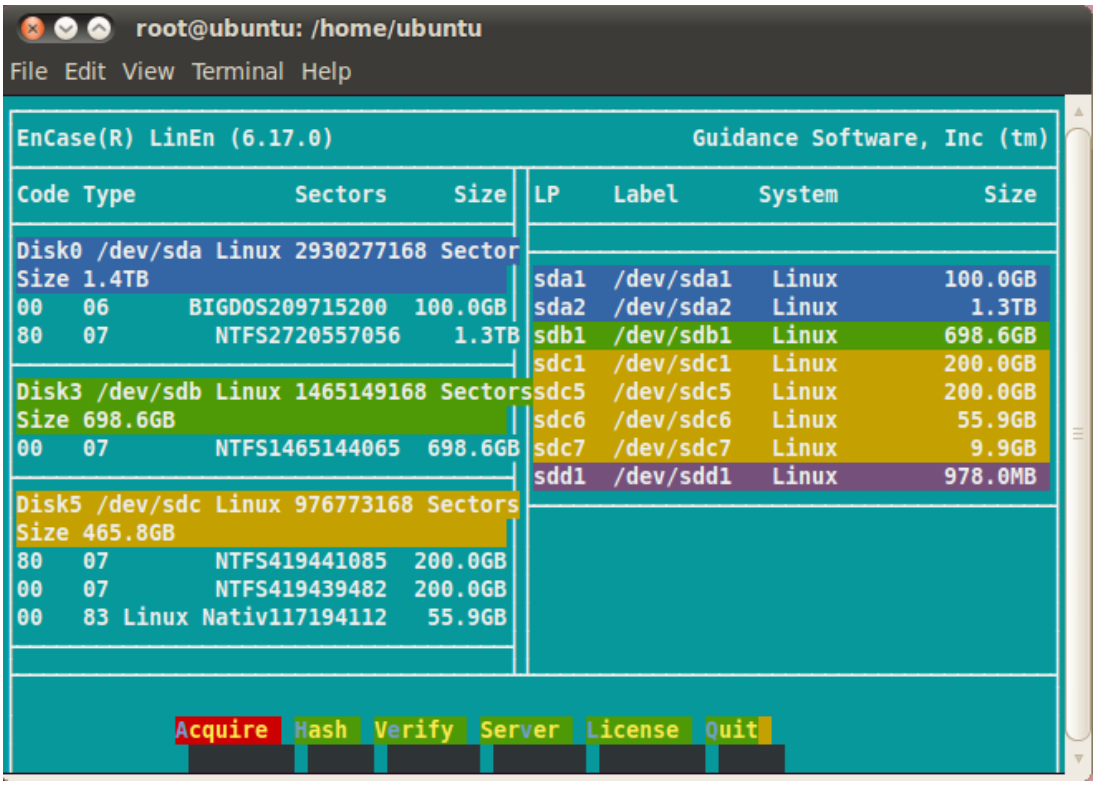

Figure 4.3: The LinEn GUI

4.1.5 Searching for Evidence in EnCase

There are primarily three ways of searching for specific data in EnCase Forensics. The most straightforward way is by browsing through the file hierarchy in the Tree pane while looking at the contents of each directory in the Table pane. This approach compares to browsing through a file hierarchy on a retail Microsoft Windows or Macintosh OS X operating system with a browsable window-based point-and-click graphical user interface. The difference from normal

file hierarchy browsing is that files which are normally hidden from view, like deleted files, can be inspected. This way of searching for evidence is tedious, and the investigator has to know exactly where to look.

4.1.5.1 The Search Tool

Another approach, which is more targeted, is by utilizing the search tool. This utility is used for conducting searches, but is in addition used for conducting various types of analysis, the most significant being the hash analysis discussed in 4.1.8.2 on page 62, and the file signature analysis discussed in 4.1.8.1 on page 62. These analysis are available as options in the search dialog window. The search tool can be utilized by searching for specific keywords, or it can be used for searching more generally by finding specific types of files, like e-mail, or searching through the Internet history.

4.1.5.2 Filters, Conditions and Queries

Finally, the third way of looking for evidence is by applying *filters*. Filters are according to [4, page 513], specialized EnScripts that allows the filtering of a case based on a set of parameters. By employing filters it is for instance possible to exclusively display all the picture files on the evidenciary media.

One important difference from the search tool is that filters can utilize *indexing*, while searching does not. Indexing is a very tedious process, often taking hours or even days, depending on the size of the acquired media. Once the index is created, the search for evidence using filters is significantly faster than using the search tool. Another important difference is that filters can, to a much greater extent, be customized, allowing even more targeted searching than employing the search tool. Having mentioned that the search tool does not utilize the index, even after it is created, means that it will always take a considerable amount of time, unless a small portion of the evidenciary media is selected.

According to [4, page 513], *conditions* are used for creating custom filters using preset parameters accessible through dialog box choices. In a way, conditions can be viewed as the building blocks of filters.

The conditions tab in the Filter pane has many conditions that can be applied to an indexed case. It works by showing all files matching the current condition in the Table pane. There are numerous types and categories of different conditions. By choosing Files and the File Groups under Conditions in the Filters pane, it is possible to organize the viewing of files according to their file type. Some important file types include all pictures, compressed files, executable files, e-mail, movies and programming code files, along with many more.

Filters work in an additive manner. If one has previously filtered for all pictures, and then subsequently filtered for e-mail, all files matching the picture filter are displayed, along with all files matching either the e-mail filter, or both. Under the Filter tab in the Table view of the Table pane, it is shown what type of filter(s) the files match.

As a last note, filters and conditions can be combined to create complex queries using boolean logic. This facilitates the combination of several filters into an even more powerful tool.

4.1.5.3 Indexing

EnCase Forensics features an indexing function used for facilitating more efficient and faster searches. By performing a search employing filters and conditions against the index, and not the data, the results are according to [4, page 333], near instantaneous search results. The

process of creating an index can take a considerable amount of time, but once finished, the time savings are tremendous [4, page 333]. Several options are available for configuring and optimizing the indexing process. It is possible to choose how much memory should be allocated to indexing, if files that match a hash database of known files and operating system files should be included in the index or not, and the minimum number of characters a word should contain to be added to the index. In addition, the user can decide whether to convert all words to a lower case format, and whether to utilize a *noise* file, eliminating common words such as "the", "and", "a" and so forth. Noise files for most languages exist, but unfortunately it is only possible to utilize a single noise file.

4.1.6 Automation Tools

4.1.6.1 EnScript

EnCase features a scripting language, called EnScript, which can be used for automating various forensic processing tasks [4, page 509]. These scripts are called EnScripts, and are small pieces of code [4, page 509]. New scripts can be programmed, or the pre-built scripts can be employed.

According to [4, page 509], EnScript is a proprietary programming language and application programming interface (API) existing within the EnCase program environment. Consequently, a slight disadvantage of EnScript is that EnCase needs to run while running EnScripts [4, page 509]. Therefore, EnScripts cannot be utilized while acquiring an image in Linux using LinEn. According to [4, page 509], EnScript adheres to the ANSI C++ and Java standards for expression evaluation and operator meanings, resulting in a comfortable transition for users accustomed to programming in those languages [4, page 509]. As a last note, filters, conditions and queries are also considered forms of EnScripts, see 4.1.5.2 on the facing page.

4.1.6.2 Automatically Generated Reports

EnCase is capable of automatically creating paperless reports. The report is created based on the bookmarks. By organizing the folders containing the bookmarks, a report presenting the findings can be automatically created. The folders should be organized in a specific way though. In the root directory a single folder should be created. This will become the heading of the report. Within that folder, the author of [4] suggests creating two folders, namely "Acquisition and Device Information" and "Examination Findings".

By exporting the bookmarks to a web page report, the evidence is linked into the document, enabling readers to click the evidence link in the report, inspecting the evidence for themselves.

According to [4, page 510], by running the Windows Initialize Case module of the Case Processor EnScript with each case, an attractive and informative front end is automatically added to the report, including a Linux version as well.

4.1.7 Recovery

4.1.7.1 Undeleting files

When mounting media, or loading evidenciary files into EnCase, deleted files, i.e. files found in unallocated space, are automatically recovered without the users intervention [4, page 60], by the combined effort of two features called the Link File Parser, and the File Finder. The first time evidence is loaded into EnCase, a slight wait is imposed, before the acquired evidence is made evident. During this pause, waiting for evidence to load, deleted files are actually

restored. This is not something the user has to worry about, since this is done automatically. From the users perspective, the deleted files are made available for inspection just by loading the evidenciary files into EnCase.

4.1.7.2 Recover Folders

EnCase Forensics features a *Recover Folders* option, enabling folders on the acquired volume to be recovered. EnCE - The Official EnCase Certified Examiner Study Guide, second edition, recommends scanning the volume for directory structure meta data very early in the case [4, page 477]. This is done by right-clicking the volume in the Tree Pane, and selecting Recover Folders. Even if the folders contain no files, the mere name of the folder can be helpful when searching for evidence.

4.1.7.3 Partition Recovery

Partition recovery is a feature in EnCase, which automatically [1] rebuilds the structure of formatted NTFS and FAT volumes. Unfortunately, this feature does not exist for Linux systems typically using the Ext2 or Ext3 file system.

4.1.8 Additional Analysis Features

4.1.8.1 File Signature Analysis

EnCase is able to conduct file signature analysis on selected files, or across the entire range of evidence in the case. Until a file signature analysis is performed, EnCase identifies files based on their file extension, which may not be correct if the file is misnamed. File signature analysis is available by running the search utility discussed in 4.1.5.1 on page 60, and selecting the verify file signatures option.

4.1.8.2 Hash Analysis

EnCase has the ability to calculate hash values, using MD5, for files on the acquired evidence, and compare those hash values to a database consisting of precomputed hash values for known files. This process is known as a hash analysis.

If the hash values are identical, the evidenciary file can be assumed to be an exact duplicate of the file in the hash database. The odds of two dissimilar files having the same MD5 value, is according to [4, page 360] one in 2^{128}, amounting to about 340 billion billion billion billion. Consequently, when two hash values match, it is practically safe to assume that the two files are exactly the same. Hash analysis is useful for discovering system- or program files that have for some reason been tampered with, either by the suspect, a virtual trespasser, or by malware. A hash analysis can be executed in EnCase Forensics using the search tool discussed in 4.1.5.1 on page 60.

4.1.8.3 E-mail Support

EnCase has support for several e-mail formats, and has the ability to both parse and mount them [4, page 514], which is useful to display e-mail in a native format. This feature is available using a sub tab in the Tree Pane, i.e. the window in the top left part of the screen, which is called the Records tab. By utilizing this feature, e-mail messages can be observed in a typical e-mail format with headers such as From, To, Subject, Created Date, Sent Date, Received,

Header, and Attachments [4, page 514]. Currently, EnCase 6, supports the following e-mail systems:

- Outlook PSTs/OSTs
- Outlook Express DBX/MBX
- AOL 6.0, 7.0, 8.0 and 9.0 PFCs
- Hotmail
- Yahoo!
- Netscape web mail
- MBOX archives (UNIX)
- Lotus Notes v6.0.3, v6.5.4 and v7
- Microsoft Exchange EDB Parser

E-mail clients not mentioned in this list are not supported, and EnCase will therefore not be able to retrieve e-mail information regarding those data formats.

4.1.8.4 Restoration

Another form of analysis is by observing the suspect drive as it is running. EnCase Forensics is able to restore EWF files to another drive, in effect enabling the user to boot the suspect drive and run applications [4, page 542]. Restoring a drive and booting it enables the user to see the suspect system much like the suspect did. A drive can be restored either physically or logically. If restoring it logically, the drive will not match the original drive when conducting hash verification, since the data outside of the file system will not be present, but it will operate in the same way as a physically restored drive in every way. A drive can be restored by right-clicking the drive from the Tree Pane, and choosing restore. Subsequently, the user will be faced with selecting which drive the suspect drive is to be restored to. When restoration is complete, a hash verification is conducted, with the resulting hash value displayed to the user.

4.1.8.5 Adjustable Timezones

EnCase Forensics features adjustable timezones, which is essential for conducting correct timeline analysis. This is done by right-clicking a case or device and selecting "Modify time zone setting" [4, page 389]. This is unfortunately only supported on evidence acquired from Windows. To make matters worse, it is seemingly not possible to adjust the timezone data for any single file - it has to be done globally. This is suboptimal since a file which was sent across a network from a different timezone should be configured to a correlating timezone. Hence, the MAC times of this file will not be consistent with other relating timezone data.

4.1.8.6 Compressed and/or Encrypted Files

A *compound file*, is a file containing data which may be compressed, encrypted, hierarchical, or any of the above [4, page 480]. EnCase Forensics is able to mount and decode these files, displaying them in a logical or hierarchical structure [4, page 480]. This is done by right-clicking the file from the Tree Pane, followed by selecting View File Structure. Once mounted the compound file can be traversed in the Tree Pane, just as any other file.

Compound files can be identified from the Table View by first running a file signature analysis, and subsequently sorting for files with "* Compound Document File" in the File Signature column [4, page 480]. Filters, conditions and queries are other techniques suitable at compound file discovery.

However, as expected, not all compound files are supported by EnCase. The sheer quantity of differing formats, and the difficulty of decrypting encrypted files is challenging for any digital

forensic toolkit. An add-on module called EDS adds decryption features to EnCase Forensics. Temporary files are often compound files, and are according to [4, page 480] good candidates for mounting.

EnCase provides a purchasable, add-on module called *EnCase Decryption Suite*, abbreviated EDS, used for decryption [4, page 532]. According to [4, page 531] EDS provides the EnCase examiner with the ability to decrypt files and folders encrypted with the Microsoft Encrypting File System (EFS). EDS can gather information locally, needed for decryption, if the encrypted file was encrypted on the system being investigated. It is stated in [4, page 531] that if the file was encrypted remotely, i.e. by a third party, or if the encrypted file is protected by Syskey, EDS can employ dictionary cracking, export local user accounts, or utilize NT password cracking software. In addition, EDS provides support for obtaining passwords from password-protected Outlook PST files, and can also be used for decrypting PC Guardian-encrypted hard drives [4, page 531].

4.2 PyFlag

PyFlag is a general purpose, open source forensic toolkit merging all aspects of forensics, i.e. disk forensics, memory forensics and network forensics, into a single solution [15, page 1-2].

The name PyFlag comes from being a toolkit developed in python, while the latter part of the name - Flag, is an abbreviation for Forensics and Log Analysis GUI. Being an open source project enables its users to empirically inspect the code, and develop new features if needed. Unlike EnCase, PyFlag is required to run on Linux, but by having a web based user interface, it can be accessed and operated remotely from clients running other operating systems. More details on the Web GUI in section 4.2.7.7 on page 74.

PyFlag employs a *virtual file system*, or VFS to serve as a common container or structure for all evidence. The VFS is organized as a hierarchy, consisting of both folders and files, and is browsable, much like an ordinary file system. Once the VFS is populated, PyFlag does not differentiate whether evidence came from a network dump, memory or from disk. This allows disk forensic techniques, e.g. keyword indexing or hashing, to be applied to all the data, regardless of whether it is derived from memory, disk or network traffic [15, page 2]. The VFS also enables evidence from different sources to be utilized in a complementary fashion. An example of this, is utilizing evidence from the Internet cache in conjunction with data obtained from network traffic to produce more accurate rendering of web pages [15, page 2].

PyFlag was originally designed by the Australian Department of Defense (Cohen and Collet, 2005), and was later released under the GPL (Free Software Foundation, 2007) license [15, page 2].

4.2.1 Layout and Usability

The layout of PyFlag is based around context sensitive screen content, with an ever-present, blue system menu on the top of the screen, called the Menu Bar. Alongside the Menu Bar are two lesser elements, specifically a button in the top left corner of the screen called the Home Button, and a drop-down menu in the top right corner, enabling the user to choose which case to work on. All content, except from the Menu Bar, the Home button, and the drop-down menu, changes dynamically according to context.

Considering the significance of the Menu Bar with regard to user interaction, i.e. implications for usability, we need to look at its contents. Each Menu Bar element has a drop down menu with various related tasks. Consult figure 4.4 on the facing page for a visual impression of the layout, and the functionality embedded in the Menu Bar:

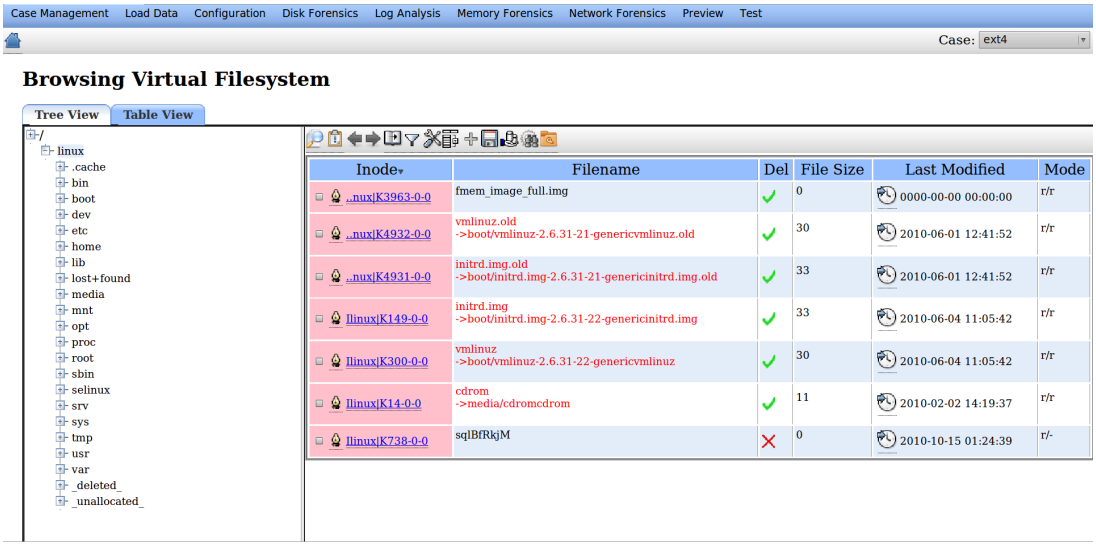

Figure 4.4: The PyFlag GUI

The layout concept of PyFlag is both similar in some aspects, and different in others, to the layout of EnCase Forensics.

On one hand, while browsing the VFS, the layout of PyFlag is based around the Tree View and Table View, as can be seen from figure 4.4. In the figure, the Tree view is selected. The *Tree View* in PyFlag serves roughly the same function as the Tree Pane in EnCase, while the *Table view* corresponds to the Table view in EnCase. To get to the equivalent of the View Pane in EnCase, an inode, representing a file, is clicked, creating another browser instance window containing detailed information about the selected file.

Even though PyFlag does not have re-sizable windows, as with EnCase, it is entirely feasible to open multiple instances of the web browser, bringing up different content in each one. This way, it is possible to look at the evidence from several aspects at the same time, much like EnCase views evidence in different granularity using different Panes. When clicking a file to get more information, while browsing the VFS, a new browser instance will be created, containing detailed information about that file. If another file is selected, the contents of the newly formed browser window will automatically update to show details regarding the newly selected file. This is quite similar to how EnCase updates the View Pane, when selecting a file in one of the other panes.

On the other hand, the layout in PyFlag is web based, which eliminates the possibility to right-click for context sensitive options. Instead, PyFlag utilize a toolbar, called the *Widget Toolbar*, consisting of several *Widgets*, or small icons, each with a tool tip. The Widget Toolbar can be observed in figure 4.4.

4.2.2 File System Support

Since PyFlag utilize The Sleuth Kit (TSK) for file system analysis, PyFlag is able to conduct forensic analysis on the same file systems as those supported by TSK. The file systems supported by TSK, and in effect PyFlag, is FAT, NTFS, Ext2/3, HFS+, and UFS1/2 file systems [21]. More on TSK in 4.2.8 on page 75.

4.2.3 Acquisition

Since PyFlag does not feature acquisition capabilities, but instead relies on acquired evidence using third party tools, like dd, PyFlag does not feature a specific kind of evidence file storage format. Instead, PyFlag supports many different third party formats, including the EnCase EWF format. Amongst other notable evidenciary formats supported by PyFlag is the sgzip format, which is a seekable gzip, raw data images, network dump files, and memory captures.

4.2.4 Searching for Evidence in PyFlag

After loading an *IO source*, e.g. an image, into PyFlag, the virtual file system may contain several files, but most likely not all. Before searching for evidence, it is necessary to populate the VFS with files that was not initially loaded. This is done using Scanners, which are discussed in more detail in 4.2.7.5 on page 74. Scanners ensure that all evidence and files are made accessible in the different Viewers in PyFlag.

4.2.4.1 Keyword Searching

It is possible to conduct keyword searching, i.e. string searching, using the Keyword Search widget from the Widget Toolbar, or by clicking the Home button and selecting Search Indexed Keywords. However, there are two prerequisites to running a keyword search. A dictionary will have to be loaded, and indices has to be created.

A dictionary file is a text file containing search phrases. The dictionary can also be stored permanently in the PyFlag database. The dictionary file has to be updated with the search phrase(s) that is going to serve as input for the keyword search, which can be accomplished using a common text editor. After the dictionary has been updated with the search phrase, the dictionary is loaded by running the script load_dictionary.py from the pyflag/utilities directory. The script is run using the python command from a command shell, and is followed by an argument, which is the dictionary file. An example would be:

```
python /pyflag/utilities/load_dictionary.py dictionary.txt
```

Before the keyword search can commence, indices has to be created. More on indexing and how they are created in 4.2.4.2. Furthermore, keyword searching can also be applied to files not normally containing ASCII text, like compressed files [21]. Scanners are able to decompress files, so the contents are made accessible. More on scanners in section 4.2.7.5 on page 74.

4.2.4.2 Indexing

PyFlag is able to employ a specific type of scanner called “Keyword Index files” for creating searchable indices, enabling keyword searches. Keyword searching was discussed in 4.2.4.1. This scanner is available in the Widget Toolkit by clicking the Scan button and selecting Configure GeneralForensics, and then enable the “Keyword index file” scanner. An *index* in PyFlag is a binary file storing a list of offsets for each word in the dictionary [21]. By running the Keyword index file scanner, an index for each individual file is created, containing the location, i.e. offset within that file, for each of the search phrases in the dictionary [21].

A prerequisite for creating an index is that a dictionary has to be loaded, which was discussed in 4.2.4.1. The indexing process uses this dictionary to create an index entry, like a shortcut, to the search phrase inside the file. Note that only search phrases present in the dictionary will be indexed.

The obvious advantage of using indexing rather than for instance `grep` to search for a single keyword, is faster search times once the indices are created. A grep will have to read the entire file before a result is displayed to the user, while the same search using indices and keyword searching will skip directly to the correct offset of the file, in effect reading only the keyword [21]. Another advantage of allowing the user to specify the contents of the dictionary is obvious. By using a small dictionary file containing few search phrases, the indexing process will be more targeted, resulting in faster index creation times.

A disadvantage using indexing, is the time it takes to create it, which will depend on the size of the VFS, and the number of search phrases in the dictionary.

4.2.4.3 Grep and find

It is possible to install a program called Fuse to work with PyFlag, allowing access to standard Linux grep and find. Fuse works by allowing users to write file systems in user space, giving the users access to higher level libraries and languages, like python.

4.2.4.4 Filters

PyFlag features filtering capability available from the Widget Toolbar. Filtering of data is accomplished writing simplified SQL-like statements into a Search Query box, assisted by two drop-down boxes containing available syntax. Complex queries can be built using boolean logic like "AND" and "OR" between statements. Once the query is typed into the Query box and submitted, the query is translated into real SQL syntax. The resulting SQL query can be inspected by clicking "SQL used" from the Widget Toolbar, but unfortunately the user cannot modify the underlying SQL. Another useful feature in the Widget Toolbar is the "Count rows matching filter" button. This will count the number of inodes in the directory matching the currently applied filter.

Filtering in PyFlag is very different from filtering in EnCase. In PyFlag, a filter is applied to the currently chosen directory, not across the whole evidenciary media. Instead of utilizing filters as a way of searching for evidence, filters in PyFlag serve to narrow the scope of displayed files within a directory. For instance, by clicking the Filter Table widget, and entering `‘‘Last modified’’ after 2010-06-04’’ AND ‘‘Filename’’ contains ‘‘.img’’`, all files which have been altered during or after this date, and contains the string ".img" will be displayed, while all other files are hidden.

The filter can be deactivated by clicking the Filter widget once more, or by selecting a new directory. Consequently, if the user wishes to apply the same filter to the newly selected directory, the filter has to be manually reactivated, and the search query must be updated. To increase efficiency, the user can copy the MySQL statement prior to selecting a new folder, and subsequently paste it into the Search Query of the next filter, in effect reapplying the filter to the next directory.

4.2.5 Automation Tools

4.2.5.1 Scripting

PyFlag employs a scripting language called PyFlash allowing automation of common tasks. The most prominent areas of application for PyFlash is automating tasks requiring constant interaction over a long period of time. PyFlash can be started by typing pyflash into a command shell after installing PyFlag. Farell [17, page 55] demonstrated in 2009 that PyFlag can be utilized to conduct a fully automated analysis employing PyFlash.

PyFlash can be used for investigating the system manually, or it can be supplied with a list of commands, i.e. a script, automatically executing these commands sequentially. A script can be used as input for PyFlash by executing `pyflag < commands.txt` from the command shell, where evidently commands.txt is the script. The scripting language is based much on standard Linux commands, e.g. cd, ls, stat, less, cat, and several others. PyFlash can be utilized in conjunction with python to create powerful scripts containing if - else statements, for-loops etc. Finally we look at a simple demonstration of creating and running a script. This script does two things; First it loads and scans the case using a Gmail scanner. Secondly, it prints out information about an arbitrarily pre-selected file. The script looks like this:

```
load mycase
scan * gmailscanner
cd /linux/_unallocated_/
stat o00000005
```

The script was started by typing `pyflash < myscript.txt` from the command shell. It produced the following output to stdout [1]:

```
ubuntu@ubuntu-desktop:~$ pyflash < myscript.txt
Welcome to the Flag shell. Type help for help
Checking schema for compliance
Flag Shell: />Loaded case 'mycase'
Flag Shell: />Waking workers [20711, 20712]
Scanning complete
Flag Shell: />current working directory /linux/_unallocated_/
Flag Shell: /linux/_unallocated_/>status : alloc
uid : None
links : 4
dtime : 0000-00-00 00:00:00
size : 0
filename : /linux/_unallocated_/o00000005
name : o00000005
gid : None
link : None
mode : r/r
mtime : 0000-00-00 00:00:00
inode_id : 14675
path : /linux/_unallocated_/
atime : 0000-00-00 00:00:00
inode : Ilinux|o791719936:0
ctime : 0000-00-00 00:00:00
-------------
```

4.2.5.2 Automatically Generated Reports

PyFlag is able to assist the user in creating detailed reports. Farell [17, page 55] showed in 2009 that PyFlag has the ability to automate report generation, using the provided scripting language PyFlash. PyFlag is able to prepare reports with support for interlinked static HTML pages.

Before a report can be created, important evidence files must be bookmarked. Consequently, these files are available for inspection by selecting the View Case Report, under the Case Management tab in the Menu Bar. By utilizing the Export Table widget from the Widget Toolbar, the user is able to export tables or files to the report.

[1] When data is sent to stdout, it is printed on the screen.

4.2.6 Additional Analysis Features

4.2.6.1 Hash Analysis

According to the PyFlag documentation located on the official PyFlag website, PyFlag is able to load the largest public hash database, the NSRL database maintained by the National Institute for Standards and Technology (NIST). This database is loaded running a script called nsrl_load.py from the /utilities directory, and contains over 25 million entries. For more information on hash analysis in general, refer to 4.1.8.2 on page 62.

4.2.6.2 Offline whois queries

Anyone with a Linux system can perform a whois query, but these are typically slow, and only allow a limited number of queries. To allow for faster queries, in an unlimited amount, PyFlag allows the user to download a whois database from APNIC or RIPE, containing the entire range of assigned IP addresses, and execute these queries offline. According to [21], the script can be invoked by running `./utilities/whois_load.sh`.

4.2.6.3 Log Analysis

PyFlag has extensive support for log analysis. By utilizing special modules called *Log Drivers*, which are templates or blueprints of specific logs, PyFlag is able to handle any kind of log file. By providing PyFlag with a log file, it is able to use that log as a template, i.e. a Log Driver for parsing other logs of the same type.

Since there exists differently formatted logs of the same type due to different software versions, a Log Driver can be made more flexible by creating a *Log Preset*, which is an instance of the log driver configured by the user. This enables a single Log Driver to handle subtly different log formats despite versioning.

4.2.6.4 Network Forensics

PyFlag has a fair amount of features regarding network forensics. More specifically, PyFlag is able to look at individual packets being transferred over the network, inspect FTP downloads, find previously visited web sites using DNS and HTTP requests, and intercept IM and e-mail communication. All network forensic features are available from the Menu Bar under the Network Forensics tab.

PyFlag currently supports Hotmail, Gmail, Yahoo mail, and supports the retrieval of IM sessions, like Yahoo messenger, IRC and MSN Chat [15, page 2] [21]. Additionally, it is capable of extracting messages from HTML and Javascript pages [15, page 2].

4.2.6.5 Memory Forensics

The Volatility Framework is integrated into PyFlag facilitating several memory forensics functions, with the most prominent being able to view running processes, open sockets and open files on an acquired memory image. These features are available in PyFlag under the Memory Forensics tab in the Menu Bar, see 4.4 on page 65.

For memory forensics to be conducted, an image of memory will have to be loaded into PyFlag. This is done selecting Load IO Data Source under the Load Data tab in the Menu Bar. When prompted for the IO subsystem on the following screen, it is imperative to choose Standard from the drop-down menu.

4.2.6.6 Correlating Timezones

PyFlag makes a distinction between the case timezone and the evidence timezone. The *case timezone* is according to [16, page 3] the timezone in which PyFlag will display all timestamps to the user. If this is set to the local timezone at which the suspected crime occurred, the timestamps will be translated to this timezone, enabling a view of all events in local time, as they occurred at the crime scene. Contrary to the case timezone, the *evidence timezone* reflects the local timezone in which the evidenciary images were acquired. The evidence timezone can be the same as the case timezone, but not necessarily. It is especially important to choose the correct evidence timezone if the evidence contains zip files, since they store only local time and not UTC [16, page 3]. If the wrong timezone was chosen in such a scenario, it would not be possible to compare zip timestamps with timestamps from the network capture or memory image [16, page 3].

4.2.7 Architecture

With PyFlag being an open source project, unlike EnCase Forensics, it is possible to examine how the different components of PyFlag interact. M. I. Cohen has published an overview of the different architectural components of PyFlag, which can be inspected from figure 4.5.

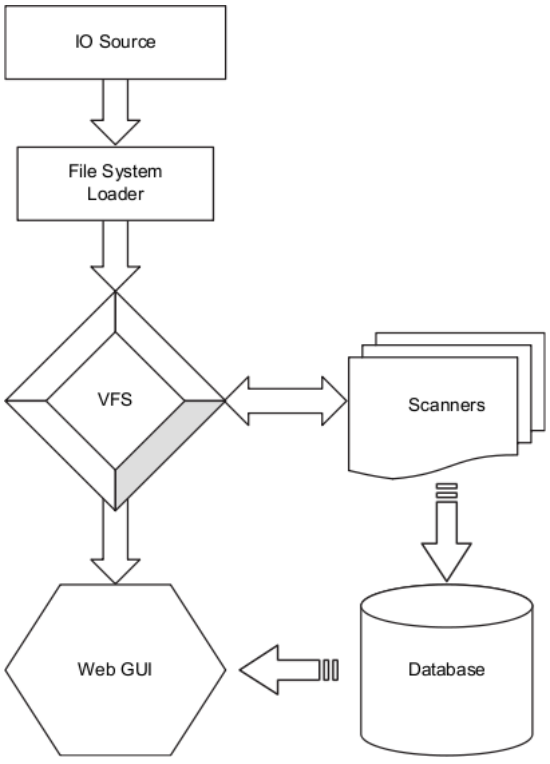

Figure 4.5: The underlying architecture of PyFlag [15, page 113]

The *IO Source* utilize IO Source Drivers to make sure different imaging- or evidenciary file formats are correctly interpreted and imported into PyFlag. Subsequently, the IO Source provides its output to the *File System Loader*, which populates the virtual file system, or *VFS*, with data. The VFS is based on a real file system, and contains *inodes*, which can be thought of as data containers from a users standpoint. In fact, they are not containers, but a string pointing to where a specific data object, like a file, a partial file, or some other data artifact is stored. More on inodes in section 4.2.7.4 on page 73. Unlike the inodes in a Linux file system, the inodes in PyFlag serve as "containers" for all sorts of data, like memory resident processes

or network traffic. Some files, especially those within compound files, may not be discovered by the File System Loader, in effect not being displayed to the user. For these files to be visible in the VFS, *Scanners* are manually employed by the user, searching for inconveniently accessible data, e.g. by decompressing compressed files, performing carving on files with no logical connection to the file system, and searching for e-mail. PyFlag operates on top of a *Database* which is filled with data after the VFS has been populated, and after a Scanner has finished executing, providing it uncovered new data. Finally, the database and the VFS provide output to the *Web GUI*, enabling the user to get a visual representation of the data.

Next, we proceed by looking at each architectural component of PyFlag in greater detail. Some components are associated with the process of loading an image file into PyFlag, so to obtain an improved understanding, this process is also briefly mentioned in the following sections.

4.2.7.1 IO Source

According to the online PyFlag manual [21], an IO Source is simply a collection of information used for getting the raw image data. Images are often stored in formats other than raw data dumps, sometimes referred to as dd format, which is done in order to provide compression, or store case related meta data together with the original image [21].

PyFlag supports many different image formats through the use of IO Source Drivers [21]. The IO Source Drivers are selected in the Select IO Subsystem drop-down box in the GUI prior to loading the IO source file. This enables PyFlag to know how the format of the image file should be interpreted. The option, Standard is used for most image files like mem, pcap, hard drive images and zip files, while EWF is used for EnCase files. Other options for selecting the IO Subsystem include RAID5 and Mounted.

After selecting the IO Source Driver, the user is required to input the partition offset. In most cases this will be set to zero, but in some cases the partition table needs to be consulted to know where the file system actually begins. This is usually required when providing PyFlag with an image file consisting of an entire hard drive containing multiple partitions, as opposed to providing PyFlag with an image of a solitary partition. The partition table is located at the beginning of each hard drive, usually just after the MBR, and is implemented differently according to the operating system used for creating the partitions [11, page 71]. By clicking the button named Survey the Partition Table, right next to the offset field, PyFlag utilize The Sleuth kit for automatically extracting information from the partition table. If there are no partitions present in the image file, a pop-up window will appear telling the user that no partitions are found. Conversely, if TSK discovers a partition table, the GUI will tell the user know which partitions are available. By simply clicking the selected partition, PyFlag automatically inputs the correct offset in the offset field.

Before submitting, the user gives the IO source a Unique Data Load ID, which is an arbitrary string representing the name of the IO source data. This is merely a name for the image file, serving to provide the user with an easy way of recognizing the image. The meta data relating to this IO source is then stored internally with the IO source under this ID. If the user wish to process the same IO source at a later time, the same ID is used, saving the user from inputting the same meta data once more.

4.2.7.2 File System Loader

When a new case is first created, the VFS is empty. Consequently, the VFS must be populated with some initial data, which is the task of the File System Loader [21]. To read more about the VFS, refer to section 4.2.7.3 on the next page.

The authors of the PyFlag manual claim the File System Loader simply examines the file system being located within the output provided by the IO Source, and discovers all the files

within it [21]. However, this statement is only true to a certain extent. If files within the IO Source are composite in nature, containing other files, e.g. compressed archive formats, these may not be extracted by the File System Loader. To decode and populate the VFS with such files, Scanners must be employed, which you can read more about in in section 4.2.7.5 on page 74.

When output from the IO Source is loaded as input into the File System Loader, an application called Magic identifies the input file, displaying its findings to the user. Based on the results provided by Magic, The File System Loader will try to automatically select the correct file system type. If Magic is unable to identify the file, the user will have to select the file system type manually. E.g. Magic can identify a pcap file, and will select the PCAP Virtual Filesystem without the need for user interaction. Conversely, if the file is a compressed ZIP file, the user has to manually select the right file system type by selecting Raw from a drop-down box in the GUI. Other options for file system choices include Linux Memory, Windows Memory and Sleuth kit.

Furthermore the user types in the selected VFS Mount Point, which is what folder in the VFS the contents of the image file will be loaded into. This can be useful for organizing the case. A proposal for organizing the VFS would be to mount network captures in /net/ and memory images in /mem/, in effect keeping these sources of evidence in separate folders, away from the disk image.

If the file is a memory file, the user also needs to provide PyFlag with a Profile, and a Symbol Map. The profile is the kernel version of the OS the memory was extracted from, and is automatically detected and selected by PyFlag.

The final step of loading an image- or evidence file into PyFlag is by pressing the Submit button. The loader then calls the VFS to create the corresponding files within the PyFlag VFS [21].

4.2.7.3 The PyFlag Virtual File System

PyFlag employs a virtual file system, or VFS, to facilitate the integration of several IO sources into a single, browsable pseudo file system. A virtual file system is an abstraction layer running on top of the IO sources. This makes it possible to import a network capture, a memory image, and an image of the hard drive into the same VFS, enabling the user to view all this data in a single browse-and-click web interface.

After IO sources have been imported, the VFS is populated with regular, non-compound [2], active data [3] by the File System Loader. To ensure that residual data and composite files are also displayed in the VFS, we need to employ scanners, which are discussed in further detail in section 4.2.7.5 on page 74.

Data items found within IO sources are organized into inodes, much like the inodes found in an Ext3 Linux file system. These inodes can be inspected by clicking on them in the web interface, giving the user in-depth information regarding the object within the inode. An inode can be a simple file within a file system, a stream transferred over the network, or a process found within memory. Once the data is imported into the VFS, PyFlag makes no distinction between inodes with regard to their source. An inode containing data captured from the network is treated the same as an inode with data obtained from a file residing on a hard drive image. The fact that all data objects are stored and treated in the same way, results in considerable leverage. As a consequence, PyFlag is able to run the same set of features on all data objects whether the data originated from network traffic, memory or disk.

[2] Non-compound data is data that is neither compressed or encrypted.
[3] As mentioned in 2.4.1.1 on page 21 active data is data that has not been deleted.

As stated by [21], the VFS provides three central facilities. First and foremost it can open files, returning the file data. Secondly, it can create new files within the VFS. Finally, it can browse the file structure within the VFS, listing both file names and directories.

4.2.7.4 inodes

As previously stated in section 4.2.7 on page 70, inodes populate the VFS, and points to where the actual files are located. An *inode* in PyFlag is a unique address within the VFS, facilitating access to a specific file [21]. It can be thought of as an offset within the VFS, pointing to where the file can be found. A file in the PyFlag VFS is a discrete piece of data. Consequently every file has a start address, an end address marked by an EOF, and a finite size. As such, an inode is simply a medium containing meta data, telling PyFlag where the data can be found.

In addition to telling the VFS where the file is, the inode can also contain other meta data telling PyFlag how to process it. This is made possible by representing inodes as strings instead of integers [21]. The inode can be split into several components denoted by the pipe character '|'. Each of these pipe characters signify that a different driver is to be used in processing the component of the inode. The first character in the inode name, and the first character following a pipe character tells PyFlag which driver is to be utilized for that specific component. Using the correct driver, the VFS opens the component and obtains the file [21]. The VFS now moves to the next component, determines its driver and provides it with the previous file opened [21]. Once all components have been processed, the resulting file is provided to the user of the VFS [21]. See figure 4.6 for an example.

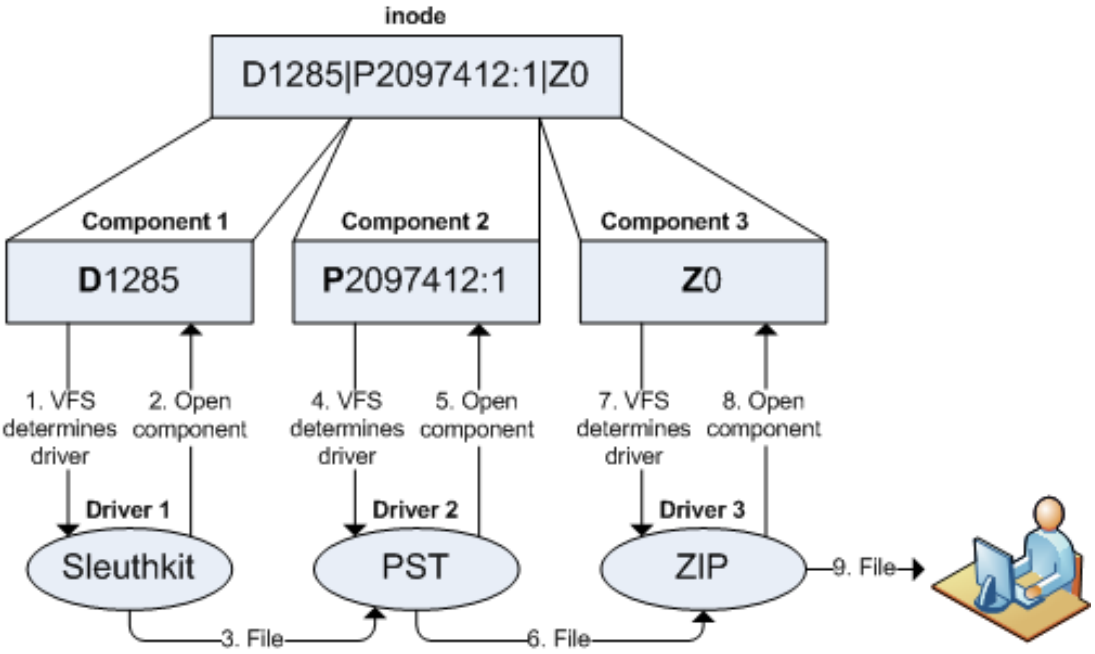

Figure 4.6: How a PyFlag inode is processed, to display the file to the user

The PyFlag VFS can according to [21] contain an indefinite number of inodes. Consequently, PyFlag can contain an infinite number of files, unless restricted by hardware limitations. It is reasonable to infer that this is due to the fact that strings in python can have an arbitrary length.

An inode is unique and will always provide the same file within the VFS, but note that several inodes can refer to the same file. This can happen if a file is captured as it is sent across the network, but is also found stored within the image of the hard drive.

4.2.7.5 Scanners

Simply put, scanners are employed to populate the VFS with data that the File System Loader was unable to. After an IO source has been imported into PyFlag, not all available files may necessarily be accessible right away. What is available depends on the format and content of the IO source file. PyFlag provides the user with the opportunity to use something called scanners to extract data of interest within files or file systems.

Scanners are employed by clicking the Scan widget from the Widget Toolbar. This opens a new window containing five types of primary scanners, or scanner categories. Each primary scanner is able to run independently in its default configuration, or more specific scanners inside the primary scanner can be enabled or disabled. The primary scanners are called:

- Network Scanners
- File Type Related Scanners
- Compressed File Support
- General Forensics
- File Carvers

It is possible to customize the scan for each primary scanner in great detail by pressing the configure button next to the name of each scanner. This displays all specific scanners that the primary scanner consists of.

For instance, one can use a compression scanner to automatically decompress all zipped files within the VFS. This creates virtual inodes within the VFS, being populated with the newly extracted content. Lets say the user wants to import a zip file containing a memory dump file (.mem), and a network capture (.pcap). When the zip file is loaded initially, PyFlag only displays an inode with the file name raw_filesystem. Because this inode represents a reference to a compressed file, it needs to be extracted before the contents can be accessed. By pressing the Scan button in the GUI and enabling "Compressed file support", the scanner is able to process the file, decompressing its contents. When the scanner has completed, the .pcap and .mem files can be accessed through the GUI. Scanning is done recursively, so if the compressed file contains other compressed files, they will be extracted and integrated into the VFS also.

There are many other uses for scanners besides decompressing zip files, like extracting e-mail and IM data from network captures, perform JPG carving [4], or scan the browser history, just to name a few. In addition to employing scanners for evidence extraction, it can also be used for creating keyword indices.

4.2.7.6 Database

PyFlag operates on top of a MySQL database. This gives PyFlag multi-user capability and ensures the atomicity, integrity, consistency and durability of the underlying data.

4.2.7.7 Web GUI

Instead of being an application restricted to run locally on a forensic workstation, like EnCase, PyFlag utilizes a web based GUI facilitating concurrent, multi-user capability on the same case. The Web GUI receives input from both the database and the VFS, thus displaying this to the user.

[4] Carving is the process of locating, identifying and extracting files which are not part of the file system, e.g. by looking at the file header.

By default, PyFlag is configured so it can be accessed through a web browser on IP address 127.0.0.1, i.e. localhost, through port 8000, but this can be changed in the GUI. The combination of a web interface with an underlying database makes PyFlag a very powerful multi-user tool. Once the GUI has been accessed, the user can select PyFlag Configuration under Configuration from the system menu, and configure the GUI to be accessible from other ports or IP addresses. This enables users running other platforms, like a Macintosh or Windows, to connect to the server backend running PyFlag, accessing and utilizing PyFlag through the provided web-based GUI. Additionally, it is possible to run the database server on the same machine used for accessing the GUI, in effect providing a single computer solution.

4.2.8 The Sleuth Kit

PyFlag employs The Sleuth Kit (TSK) for conducting filesystem-level analysis [21], which is a C library and a collection of command line tools. Since TSK forms the foundation for file system analysis in PyFlag, it is inadequate to consider PyFlag exclusively - TSK will also have to be explained. A module written in C called sk integrates TSK into PyFlag. The Sleuth Kit was developed by Brian Carrier, and runs on UNIX platforms including Linux [3, page 102]. Additionally, TSK is open source [3, page 102]. According to [6], TSK is based on The Coroners Toolkit. To understand the basics of TSK, TCT must be explained.

TCT was developed to collect data from live systems using a collection of command line tools. Included in the toolkit was grave-robber and pcat, used to copy data from live systems, ils and icat for analyzing inode structures, MAC time to make timelines of file activity, unrm to extract unallocated data blocks from the file system, and finally lazarus, which was used to analyze chunks of data and guessing content type while recovering deleted files.

TCT supports the FFS and Ext2 file systems, and is both free and open source. As pointed out by [6] TCT had some limitations. First and foremost it had no support for file and directory names. This was because TCT was restricted to the block and inode layers. Secondly, it had the disadvantage of being extremely platform dependent. The analysis system actually had to be the same as the system being analyzed. Lastly, TASK had no support for common file systems like FAT and NTFS.

In an effort to adjust for these drawbacks, Brian Carrier developed TCTUTILs, which was an add-on to TCT, adding the capability of recognizing file and directory names.

Even though TCT was improved by the integration of TCTUTILs, it still had some limitations. As a result, TASK, the predecessor to TSK was developed by Brian Carrier in conjunction with @stake, a computer security company later acquired by Symantech. TASK merged TCTUTILs and TCT into one package. According to [6], platform dependence was removed, which meant that users were able to for instance analyze a Linux image on a Solaris analysis station. Additionally, support for FAT and NTFS file systems were introduced, along with new tools for using hash databases and sorting files.

Considering that a suspect system should never be trusted (see 2.1 on page 7), TASK did not rely on the OS to read a file system. As a result, TASK was able to display files which was flagged and hidden by the OS.

4.2.8.1 Design

TASK and TSK is designed around a layer based model, enabling access to all data. As stated in [6], TASK, and thus TSK and PyFlag, operate on four layers. These are the Data Unit Layer, Meta Data Layer, File Name Layer and File System Layer. The *Data Unit Layer* contains the content of the files organized into discrete [5] units of data. The *Meta Data Layer* contains

[5]The term discrete is used for describing something that has a specified start and end, and a finite size.

descriptive data about the data, while the *File Name Layer* holds the file- and directory names. Finally, the *File System Layer* has information specific to the file system being investigated. There existed eleven tools in 2003, each with its distinct purpose [6]. The prefix of the tool, which usually is the first letter, signifies which layer it operates on, while the subsequent characters describes its function [6]. For instance, any tool starting with the character d, is a tool operating on the Data Unit Layer. For the Meta Data Layer the prefix is i, File Name Layer the prefix f, and for the File System Layer the prefix is fs. According to [6] there are five different functions employed by the tools on different layers. These functions are ls, which lists information, cat, which displays the content, stat, showing details about an object, find, which maps other layers to its layer, and finally calc, used for calculating something in the layer. According to [6] the eleven tools operating in different layers are the following:

- Data Unit: dls, dcat, dstat, dcalc
- Meta Data: ils, icat, istat, ifind
- File Name: fls, ffind
- File System: fsstat

These eleven tools can- and have been used to build new tools. An example of this is MAC time, using fls and ils to get timeline information regarding files, e.g. when it was last modified and accessed, or when the file permissions were changed.

4.2.8.2 Current Use For Command Line Based Tools

Without its graphical interface, i.e. Autopsy, or the PyFlag framework, The Sleuth kit and other command line based tools can still have valid use in their native form. This holds especially true regarding live forensics, where a small footprint is crucial. Command line based tools have no GUI, and can be run individually, without the need to install a big forensics suite, thus they need less resources, resulting in a smaller footprint. Another advantage, is that these tools can easily be scripted to run in succession providing input from a regular bash script.

Chapter 5

Testing

5.1 Test Preparations

Before testing can commence, the system to be analyzed is populated with testable evidence. The newly installed system was booted, and testable evidence planted. In addition to testable evidence, the drive contained files required for running Ubuntu. For convenient evidence retrieval and testing, files are placed in the ~/evidence directory. Additionally, this section contains tests for selected evaluation criteria regarding acquisition and hash verification.

5.1.1 Send and Receive E-mail

For testing the tool kits ability to retrieve and analyze e-mail, the target system will have to contain various formats of e-mail. The prerequisites include installing e-mail clients and creating e-mail accounts. See figure 5.1 on the next page for the complete test preparation.

When downloading e-mail, these are stored in the ~/evidence directory. However, it was soon discovered that Gmail does not have any feature enabling the user to download or archive e-mail, so tests ten, eleven and twelve were skipped.

5.1.2 Populate Browser History

Before the browser history analysis capabilities of EnCase and PyFlag are tested, the browser history on the target system will have to be populated, which was done by visiting various web sites.

Since several web browsers are tested, a prerequisite is installing Mozilla Firefox, Opera and Google Chrome. The test preparation plan is as follows:

1. Firefox: visit www.cnn.com, www.msn.com, and www.forskning.no
2. Opera: visit www.cnn.com, www.msn.com, and www.forskning.no
3. Chrome: visit www.cnn.com, www.msn.com, and www.forskning.no
4. Firefox: visit www.facebook.com and log in
5. Opera: visit www.facebook.com and log in
6. Chrome: visit www.facebook.com and log in
7. Firefox: visit www.google.com and search for “how to crack a safe”
8. Opera: visit www.google.com and search for “how to crack a safe”
9. Chrome: visit www.google.com and search for “how to crack a safe”

Test number	From Client	To Client	Subject	Body	Attachment	Compressed attachment	Archived/ Downloaded
1	evolution mail	gmail	emailtest1	from evolution mail to gmail	no	no	no
2	evolution mail	gmail	emailtest2	from evolution mail to gmail with attachment	yes	no	no
3	evolution mail	gmail	emailtest3	from evolution mail to gmail with compressed attachment	yes	yes	no
4	evolution mail	gmail	emailtest4	from evolution mail to gmail archived	no	no	Archived
5	evolution mail	gmail	emailtest5	from evolution mail to gmail archived with attachment	yes	no	Archived
6	evolution mail	gmail	emailtest6	from evolution mail to gmail archived with compressed attachment	yes	yes	Archived
7	gmail	hotmail	emailtest7	from gmail to hotmail	no	no	no
8	gmail	hotmail	emailtest8	from gmail to hotmail with attachment	yes	no	no
9	gmail	hotmail	emailtest9	from gmail to hotmail with compressed attachment	yes	yes	no
13	hotmail	evolution mail	emailtest13	from hotmail to evolution mail	no	no	no
14	hotmail	evolution mail	emailtest14	from hotmail to evolution mail with attachment	yes	no	no
15	hotmail	evolution mail	emailtest15	from hotmail to evolution mail with compressed attachment	yes	yes	no
16	hotmail	evolution mail	emailtest16	from hotmail to evolution mail archived	no	no	Downloaded
17	hotmail	evolution mail	emailtest17	from hotmail to evolution mail archived with attachment	yes	no	Downloaded
18	hotmail	evolution mail	emailtest18	from hotmail to evolution mail archived with compressed attachment	yes	yes	Downloaded

Figure 5.1: Test preparation plan for e-mail

5.1.3 Insert Various File Types for Report Generation

A prerequisite for conducting this test is downloading HTML pages, Open Office documents, picture files, word files and others, to see if these can be included in the report. Files should preferably be viewable in their native format. These are the files planted on the suspect system.

- Downloaded HTML pages
- Pictures
- Archived E-mail
- Various Office documents (Open Office, Microsoft Word)
- Text files (ASCII)
- Binaries

5.1.4 Create Compressed File Archives

To test the tool kits ability to look inside compressed files, several compressed files were created, including compressed active files and compressed residual data in unallocated space. Compressed residual data was created in section 5.1.6, so only active compressed data is be considered in this section. Additionally, the tool kits ability to handle multiple layers of compression is tested. The test preparation plan is as follows:

- Create a tar file called tar-test1.tar containing two files. These files are named inside_tar-test1_1.txt and inside_tar-test1_2.txt.
- Create a tar file called tar-test2.tar containing the compressed file inside_tar-test2.tar, containing innermost_tar-test2.txt.
- Create a zip file called zip-test1.zip containing two files. These files are named inside_zip-test1_1.txt and inside_zip-test1_2.txt.
- Create a zip file called zip-test2.zip containing the compressed file inside_zip-test2.zip, containing innermost_zip-test2.txt.

5.1.5 Rename Files

To be able to test file signature analysis, several files on the target machine is renamed so that file extensions are changed. The test preparation plan is as follows:

- Rename a gif file to misnamed1.txt
- Rename a png file to misnamed2.sh
- Rename a sh file to misnamed3.jpg
- Rename a text file to misnamed4.jpg
- Rename a binary (ELF) file to misnamed5.doc

5.1.6 Create Deleted Files

The ability of the tool kits to retrieve deleted files from unallocated space is tested.

When creating a file, the user has no control over which blocks and sectors are allocated to it. If creating and deleting files one after the other, there is a slight possibility that one of the newly created files will erase a previously deleted file. Conversely, one can safely assume that if all files are first created, and subsequently deleted, those files will have been allocated to different blocks in the drive, and can be found in unallocated space.

In order to test slack space, a previously deleted file will have to be partially erased by a newly created file. Since the operating system, and not the user, maintains control over which blocks are allocated to which files, it is significantly more challenging to test the recovery of files in slack space. Even if creating and deleting files sequentially, in an attempt to overwrite deleted files, there is no guarantee that any of the deleted files are partially overwritten, since we have no control over block allocation. Consequently, only unallocated space will be tested during this test.

The following files were created in the ~/evidence/ directory:

1. Three text files called del1.txt, del2.txt and del3.txt
2. An archived tar file called del7.tar with the del7.txt file inside
3. A compressed file called del8.zip with the del8.txt file inside

All files were subsequently deleted using the command `rm` from the command shell.

5.2 Acquisition

Throughout this section there is a couple of terms that the reader should be familiar with. The term storage partition is used for the hard drive partition where the evidenciary files using LinEn, and image from dd will be stored. The term target partition is the hard drive partition on the suspect system that is to be analyzed. Finally, the term examination system refers to the hard drive used for running analysis.

5.2.1 Preparing to run LinEn and dd

Some preparations had to be made before acquisition could begin. This included installing a couple of small programs, formatting the target and storage partitions, wiping the storage partition, making a bootable USB drive, configuring BIOS to boot from the USB drive, getting superuser access from the live-cd, mounting the storage partition, and calculate MD5 hashes, before finally running LinEn and dd.

5.2.1.1 File System Selection

Before the storage drive could be physically connected to the suspect system, it had to be formatted. The real estate of the storage drive was chosen to be split up into two equally sized partitions, one used for acquisitions using dd with regard to PyFlag, and one for acquisitions using LinEn with EnCase. At first, NTFS was chosen as the appropriate file system for the entire disk, but due to drive incompatibility issues, NTFS was rejected. Refer to 5.2.9.1 on page 84 for more details.

Because of the incompatibility issues with regards to forensically inaccurate NTFS writes in Linux, FAT32 was chosen as the best solution for acquiring evidence using Linen. The FAT32 file system was a reasonable choice since it needed to be compatible with both Linux and Windows. As opposed to NTFS, FAT32-writes in Linux are considered forensically accurate. Additionally, FAT32 can be both read and written to by Linux if mounted. Even though EnCase can analyze Linux files, and acquire them by running LinEn in Linux, EnCase itself cannot run in Linux. Choosing FAT32 facilitates acquisition in Linux to the FAT32 partition on the storage drive, and analysis in Windows using EnCase.

For storing the acquired image using dd, the file system system Ext3 was chosen because of its compatibility with PyFlag, and the fact that Ext3 is the file system installed on the target partition.

5.2.1.2 Application Installation

The application gparted is used for formatting a drive, while unetbootin is used for creating a bootable USB drive. These applications were installed using `sudo apt-get install` from the command line in Linux. A reasonable question is why use a Linux application to format the drive with a native Windows file system. This will be explained in section 5.2.9.2 on page 85.

5.2.1.3 Wiping the Storage drive

Wiping the acquisition- or storage drive clean is an important step in securing that the evidenciary files are not tainted with residual data which used to be stored on the drive. Refer to 2.3.6 on page 18 for an explanation of wiping. The hard drive was wiped using the Wipe Drive option from the system menu in EnCase. The process took a few hours.

5.2.1.4 Bootable USB Creation with LinEn

The USB drive was formatted with a FAT16 file system by gparted in Linux. When the format was complete, the system was booted into Windows 7 where EnCase is installed. An Ubuntu live-cd image was downloaded and merged with LinEn, the tool used by EnCase to acquire evidence in Linux. The Ubuntu-image was merged with LinEn in EnCase by doing the following:

1. Selected Create Boot Disk from Tools in the program menu
2. Found the downloaded ISO image of Ubuntu and clicked next
3. Activated Alter Boot Table
4. Set Source as the path where the Ubuntu ISO image was located
5. Set Destination as the path where the merged image of Ubuntu and LinEn were to be stored, and clicked Next
6. Chose the path to LinEn and pressed Finish

After the image was verified, the image was placed on the USB drive, and the system booted back into Linux. The merged image was then relocated to a local disk on the examination system and unetbootin was executed using sudo. By clicking ISO and choosing the appropriate merged image, unetbootin was able to create a bootable USB drive.

5.2.1.5 Configure BIOS

Before the target partition could be mounted on the examination system, the examination system had to be booted in a trusted operating system, containing nothing but LinEn and the OS itself. BIOS was configured by pressing the Delete button on the keyboard during boot. The USB drive was set as bootable, and the BIOS was saved and terminated.

5.2.1.6 Boot using the USB Live-CD

During the next boot, F11 was pressed to enter the boot menu. The USB flash drive was selected, which booted the system into the live-CD environment provided by the USB drive.

5.2.1.7 Acquiring Superuser Access on the Live-CD

To be able to do forensics and acquire evidence using LinEn, the user has to have root access. This was accomplished by typing `sudo passwd root` in the command shell, followed by executing `su`.

5.2.1.8 Mounting Storage Drive

Before the storage partitions could be mounted, it was necessary to find out what device files were associated with those partitions. First, the command `mount | grep [sh]d.` was executed from the command shell, making sure the target system had not been automounted [1]. Next, the command `lshw -C disk` and `fdisk -l | grep [sh]d.` was invoked. These two commands give the user information regarding what devices or hard drives are available, and what partitions these contain. The most important information derived from running the lshw command is the size of the disk, the logical device file in /dev assigned to the device, the brand and model of the disk, and the type of interface standard used by the disk. This information can be utilized in identifying the target- and storage drives. Once the target- and storage drives have been identified, the fdisk command can be used to display information regarding partitions on each drive, and additionally the file system and device file assigned to that partition.

Before mounting the storage partitions, directories had to be created on the running OS stored in the bootable USB drive. The directories /mnt/EXT3 and /mnt/FAT32/ were created by using the mkdir command from the shell.

Now that directories were created, and the partitions had been identified, the storage partitions could be mounted. By executing `mount /dev/sdb5 /mnt/EXT3/` and `mount /dev/sdb6 /mnt/FAT32`, the storage partitions were mounted with r/w access, ensuring that the data acquired by dd and LinEn were properly stored.

5.2.2 Pre-acquisition Hash Calculation

Prior to acquisition, md5sum was utilized to verify the initial state of the target drive. The command used was:

```
md5sum /dev/sdb7 > /mnt/storage/pre-acq-dd-md5sum
```

The resulting MD5 hash value was 39f3dc59a446cc83f83982784c5da42a, and was stored in the file pre-acq-dd-md5sum.

5.2.3 Running LinEn

Now that the necessary preparations had been made, LinEn could finally be executed. Even though LinEn was merged along with the Ubuntu image on the USB drive, the contents of the USB drive was in the /cdrom/ directory. To start LinEn, `/cdrom/linen` was executed from the command shell. Note that LinEn has to be run in superuser mode for LinEn to be able to successfully identify and acquire partitions or hard drives. If the user does not have root access, linen will start normally, but it will display no available devices. The acquisition, hash calculation, and hash verification took about 24 minutes.

[1] Mount will report all mounted S-ATA, DVI and USB devices, as well as access permissions.

5.2.4 Post-acquisition Hash Verification for Linen

After having acquired the evidence, LinEn calculated and verified the MD5 hash value. The produced hash value was 39F3DC59A446CC83F83982784C5DA42A, which matches the hash value produced in 5.2.2 on the facing page, indicating that the target drive had not been contaminated by the acquisition process.

5.2.5 Running dd

The target was acquired by running `dd if=/dev/sdb7 of=/mnt/storage/ext3_dd.img` from the command shell. dd reported `31535532+0 records in, 31535532+0 records out, 16146192384 bytes (16GB) copied`, indicating that all data was successfully acquired. These numbers were noted for later comparison with the size of the target partition to verify that all data was actually duplicated. The acquisition and hash calculation took about 20 minutes.

5.2.6 Verify Number of Blocks Acquired

The number of bytes on the target drive is matched against the reported number of bytes acquired by the tool kits. Because HPA and DCO protection is able to hide some sectors on the disk, cryptographic hash verification is not enough to verify that all evidence has been acquired.

The most reliable way to find out how much data a drive contains is by checking the manufacturers web site. However, since the target drive is actually a partition, we executed `sudo fdisk -l -u`, calculated the number of sectors on the partition, and compared reported sectors to output from dd/LinEn. Note that HPA and DCO does not apply to a single partition, so consequently this step is included to emphasize that hashing the disk is not neccesarily enough to prove that all evidence has been acquired.

According to fdisk, the target drive started on sector 1 433 608 533 and ended on sector 1 465 144 064. By subtracting the start sector from the end sector, and adding one, it was discovered that the partition contains exactly 31 535 532 sectors. By multiplying this number with 512, the sector size in bytes reported by fdisk, it is confirmed that the target partition contains exactly 16 658 192 384 bytes [2], which matches the number of bytes reported to be acquired by dd and linen. Consequently, the image acquired is an exact duplicate of the original partition.

5.2.7 Post-acquisition Hash Verification for dd

Subsequent to running dd, another MD5 hash value was calculated to verify that the original media had not been altered by the acquisition process. The resulting hash value was 39f3dc59a446cc83f83982784c5da42a, which matched the hash values obtained earlier. This is conclusive evidence that the original media was not compromised by the dd acquisition.

5.2.8 Case creation

After the evidence had been acquired and verified, the examination system used for analyzing evidence was booted.

[2] By calculating $(32\ 535\ 532 \cdot 512)/1024^3$ we found that the size amounts to 15.51 gigabytes, which is feasible considering the partition size was set to about 15 GB.

5.2.8.1 PyFlag

To be able to find evidence in PyFlag, the storage partition will have to be mounted, and the upload directory in PyFlag has to be configured. The storage partition was mounted with `mount -o ‘‘ro,noload’’ /dev/sdc5 /mnt/EXT3/`, ensuring read-only access. By including the noload option, it is ensured that the Ext3 file system journal is not loaded upon mounting. If instead the target partition was mounted using the read only option -r, this would cause alterations in the journal, and would show up as a mismatch of the pre-mounting MD5 hash and the post-mounting MD5 hash.

Once the upload directory was configured, and the storage partition mounted, a case could be created in PyFlag. The acquired image was loaded as an IO source, and the VFS populated. At this point, the file hierarchy of the suspect system was displayed in the Tree view, and could be browsed.

5.2.8.2 EnCase

After booting into the examination system, which for EnCase was Windows 7, evidence was loaded by double clicking the EWF file. EnCase started, and the case was created based on the meta data provided during LinEn acquisition.

5.2.9 Problems During Preparation

During the preparation phase of the acquisition process, some problems were encountered.

5.2.9.1 NTFS

At first, NTFS was chosen as the file system to be placed on the storage drive. The main reason for choosing NTFS was because it can handle large files, as opposed to FAT32, which has a file size limitation of 4 gigabyte. With EnCase and LinEn this is fine, since evidence is split up into several smaller evidenciary files, but it would be a problem when acquiring an image using dd, since it creates a single, and often rather large output file, which was bound to exceed the 4 gigabyte limitation of FAT32.

It seems like the problem was not due to incompatibility issues with NTFS writes from LinEn, but rather a driver problem latent in Linux itself. Although Linux can read NTFS, with the appropriate distribution or module, writes to NTFS are not stable enough for evidenciary work, with few drivers available for this purpose [4, page 157]. EnCE - The Official EnCase Certified Examiner study guide [4, page 157] suggests using a FAT32 volume with a unique volume label for this purpose.

According to [4, page 157], the NTFS-3G driver [3] reportedly allows Linux to reliably write to NTFS file systems. This driver is subjected to testing by the forensic community, and will likely be accepted and regarded as forensically safe to use [4, page 157].

It was a matter of choosing whether to include the NTFS-3G driver on the bootable USB drive, and installing this every time a Linux system was to be acquired, or partitioning half of the storage drive with the FAT32 file system, and the other half with Ext3, for compatibility with dd acquisitions used for the PyFlag. By choosing the latter option, less space was available for each acquisition, but once the drive was partitioned, no more work was needed. In contrast, choosing the first option would required the forensic analyst to install the driver each time a suspect system was booted from the USB drive. However, the available space for each

[3] The NTFS-3G driver is available at http://www.ntfs-3g.org

acquisition would be greater. Since the available space on each of the two partitions would be sufficient to acquire evidence both using dd and LinEn, the choice was made to partition the storage drive into two equally sized partitions, one used for dd and PyFlag, the other for LinEn and EnCase.

5.2.9.2 Unable to Detect Unformatted Hard Drive in Windows

At first it seemed like a good idea to format the storage drive to FAT32 using Windows, since FAT32 is a file system developed by Microsoft. However, even while the hard drive was found in BIOS during boot, it was not found by Windows. The issue was solved by booting the system into Linux, installing and running an application called gparted, and formatting the partition to FAT32. Subsequently, the drive was available in both Linux and Windows.

5.2.9.3 Boot Issues

Problems were encountered while booting the examination system running Windows 7 after performing a format of the storage drive in Linux using gparted. Probably, the drive was accidentally set as the primary partition of the examination system, which in turn led to the boot manager grub being updated. During boot the error message "Error: invalid signature. Press any key to continue..." was displayed, and the examination system was unable to bootstrap. This was solved by manually editing the file /boot/grub/device.map from (hd0) /dev/sda1 to (hd0) /dev/sdc1. After editing the device.map file, the examination system running Windows 7, booted normally.

5.2.9.4 Using Bootable USB Creation Software

At first, the usb-creator-gtk tool was installed using sudo apt-get install, but it malfunctioned. On various forums on the Internet users reported not being able to properly use usb-creator. The solution was simply to install alternative software called unetbootin.

5.2.10 Suggestions for Improvement

A minimal operating system without a graphical user interface would be more suitable than using an Ubuntu live-cd. A minimal OS without a GUI would use less system resources, thus running faster. Also it is important to use a trusted OS, that does not perform any unwanted actions while acquiring evidence. It is much easier to get an overview of the features if an OS is small.

5.3 Analysis

This section is where most testing of the selected evaluation criteria takes place. The search for evidence, planted during the test preparation phase in section 5.1 on page 77, will be conducted, and results reported.

5.3.1 Hash Analysis

The goal of this test is to observe how the tool kits handle hash analysis, if external hash databases are utilized, and to what effect.

5.3.1.1 EnCase

EnCase is able to import the NSRL database hash sets. A version tailored for EnCase 6 was located on the NSRL website and subsequently downloaded. The instructions in [4, page 364] were followed successfully for importing the NSRL database.

The main purpose of employing hash analysis is displaying suspicious or illegal files. Additionally, EnCase utilize hash databases for conducting faster keyword searching. If a file has a match in the database, the file is not searched, in effect enabling more efficient searching.

After the NSRL hash database was imported, hash values were calculated for all files on the 15 GB image. The process was complete in less than five minutes.

Even though some files matched the hash values found in the NSRL database, the large majority did not. Results indicate that the database has not been updated with a significant amount of hash values for the version of Ubuntu installed on the suspect machine.

All discovered files were classified in the Hash Category as Known. The most interesting files for an investigator are files classified as Notable. According to [4, page 366], the user is able to quickly display notable files by pressing N when holding the mouse cursor over the Hash Category column. Since the authors did not want to download illegal files for testing purposes, no files classified as notable were discovered.

The only file found having a match in the NSRL database, which was not empty, was the file ~/.evolution/local/MBOX/Welcome to Evolution!/(Report Part), which was a mounted e-mail included with Evolution Mail. There were a few hundred other files matching the NSRL database, but they all had a logical size of zero. Although, not much interesting was discovered, EnCase cannot be blamed for the lack of Linux related files in the NSRL database, as this is a third party service. Luckily, EnCase allows users to build customized hash libraries, increasing the significance of hash analysis conducted on Linux systems.

5.3.1.2 PyFlag

The NSRL hash database can be imported into PyFlag by running the nsrl_load.py script from the /utilities/ directory within PyFlag.

Getting the NSRL database to work was not trivial. First, a code error within the /utilities/nsrl_load.py script had to be corrected. Because the script generated errors about not being able to locate the DB and conf modules, the source code was changed from `import DB,conf,sys` to `import pyflag.DB as DB; import pyflag.conf AS conf; import sys`. Subsequently, the command `sudo python nsrl_load.py ./NSRL/` was executed. Multiple other modules were not found, but nevertheless, the NSRL database seemed to be loading into PyFlag. After about two hours of loading, an MD5 scanner was applied to all evidence in the case. However, there was only a single hash match detected, which infer that NSRL was not able to successfully load after all.

5.3.2 File Signature Analysis

During test preparation (see section 5.1.5 on page 79), several files were renamed, changing the file extensions. The goal of this test is to see whether the EnCase and PyFlag can correctly identify the renamed files by their file header, i.e. signature, after a file signature analysis has been conducted. These files were planted during the test preparation phase i section 5.1.5 on page 79.

5.3.2.1 EnCase

Results prior to the file signature analysis:

- misnamed1.txt was identified as File Type: Text with File Category: Document
- misnamed2.sh was unidentified (blank File Type and File Category)
- misnamed3.jpg was identified as File Type: JPEG with File Category: Picture
- misnamed4.jpg was identified as File Type: JPEG with File Category: Picture
- misnamed5.doc was identified as File Type: Word Document with File Category: Document

These results indicate that EnCase identifies files based on their file extension if a file signature analysis has not been conducted. Below are the results after the file signature analysis has completed, which resulted in the previously blank column called Signature being populated with values:

- misnamed1.txt correctly identified with Signature: *GIF
- misnamed2.sh correctly identified with Signature: *Portable Network Graphic
- misnamed3.jpg unidentified with Signature: !Bad signature
- misnamed4.jpg unidentified with Signature: !Bad signature
- misnamed5.doc incorrectly identified with Signature: *HP-UX 64-bit Object File

The first observation made, namely that misnamed2.sh was unidentified, indicates that EnCase is unable to recognize files with a .sh file extension.

Secondly, misnamed5.doc was incorrectly identified as a HP-UX 64-bit object file, when it is in fact an ELF 32-bit executable. Considering that HP-UX 64-bit object files are based on the ELF format, all ELF files observed during the test were classified by EnCase as HP-UX files.

Third, when trying to open files with a correct signature, EnCase would open these according to the file signature, and not the extension. However, when trying to open unidentified files, i.e. with a bad signature, EnCase tried incorrectly to open these based on file extension.

Finally, EnCase was unable to recognize the file signatures of misnamed .sh-files and .txt files. Note that both the file signature and file extension was unidentified for .sh files. However, when inspecting other text files within the evidence, these were correctly identified, most likely due to having a valid file extension. Considering that a text file has no file signature, identifying misnamed ASCII files is challenging. When inspecting a common text file in Hex View, it will just contain ASCII characters, unlike for instance misnamed5.doc (the binary), which can be recognized by its starting value of 7F 45 4C 46, or the characters ELF in ASCII preceded by a character looking like a square.

To help EnCase recognize the script executable, a new signature for script executables was created. By creating the file signature "23 21 20 2F 62 69 6E 2F 73 68", or "#! /bin/sh" in ASCII, the previously unidentified file misnamed3.jpg was correctly identified as having a "*Script Executable" signature.

5.3.2.2 PyFlag

Even though the misnamed files planted on the suspect system in section 5.1.5 on page 79 were stated with their misnamed file names, PyFlag was able to automatically open these correctly without running any scanners.

This may indicate one of two possible approaches. One possibility is that PyFlag automatically conducts a file signature analysis as files are opened. Another, and more probable solution is that PyFlag redirects a file to be opened by Linux. *Application binding* is the process by which an operating system knows how to open a particular file type with a given application [4,

page 350]. According to [4, page 351], Linux use header information to bind file types to specific applications, unlike Windows which only looks at file extensions. Consequently, Linux does not care about file extensions for determining which application should open a file.

Inspecting file signatures in PyFlag is done by selecting the MD5 Hash Comparison tab under Disk Forensics from the Menu Bar. By filtering for `"Filename" contains "evidence/renamed"` the renamed files from 5.1.5 on page 79 is displayed. The following results were discovered:

- misnamed1.txt identified as "GIF image data, version 89a, 354 x 520"
- misnamed2.sh identified as "PNG image, 217 x 58, 4-bit colormap, non-interlaced"
- misnamed3.jpg identified as "POSIX shell script text executable\012- a /bin/sh script textexecutable\012- script text executable for /bin/sh"
- misnamed4.jpg identified as "ASCII text"
- misnamed5.doc identified as "ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV)"

Observations indicate that PyFlag features extensive and accurate file identification support considering all five misnamed files were correctly identified. However, mismatches were discovered by manually correlating file extensions and file signatures, and not by an automated search.

Additionally, according to [14, page 5], file signature analysis is employed in advanced carving techniques for identifying and excavating files without the use of file system allocation information. This technique is especially useful for finding deleted files.

5.3.3 Residual Data

The goal of this test is to find the deleted files planted during the preparation phase in section 5.1.6 on page 79.

5.3.3.1 EnCase

By selecting the lost files directory from the Tree Pane, users can inspect all deleted files discovered by EnCase.

EnCase was able to find eight deleted files. However, they were all reported as having file name "Lost File". To understand why EnCase is unable to display the real file names, we need to understand directory entries. Directory entries are basically structures containing a file's file name [4]. Even though directory entries are stored contiguously, they have a pointer to the next directory entry, like a one way linked list. Actually, this pointer is not a pointer at all, but a number representing the size or length of the directory entry. When a file is deleted, the directory entry is unallocated by adding its length to the previous directory entry, in effect skipping the deleted directory entry [11, page 424]. The file name of the deleted file is still there, but it is no longer a part of the logical structure. See figure 5.2 on the next page for an illustration of how a directory entry is unallocated. If EnCase were to browse through the directory entry list, not by following the "pointers", but by traversing it iteratively looking for the signature of a new directory entry, EnCase may be able to discover deleted file names also in the Ext3 file system.

Also, when inspecting Lost Files in the Hex View, the files are listed as "Empty File". This would be expected if EnCase do not check the Ext3 file system journal for a replacement inode, or perform file carving to search for the missing file data. As mentioned in 2.4.1.1 on page 21, when a file is deleted in the Ext3 file system, the block pointer is erased, which would explain why these files appear empty.

[4] Directory entries also hold the address of the corresponding inode containing meta data about the file [11, page 424].

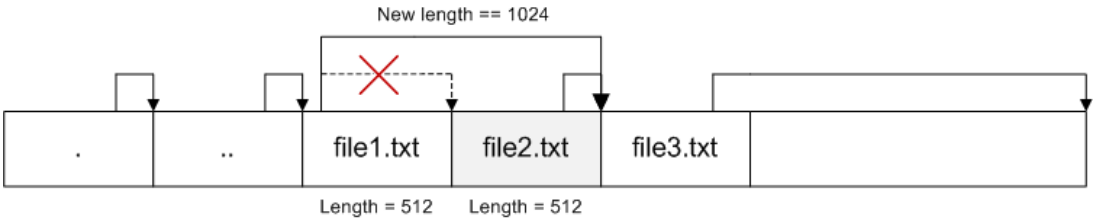

Figure 5.2: Deallocation of directory entries

As a last note, attempts were made at trying to find the deleted files in unallocated space by employing a targeted search using the Search Tool. Since the contents of the deleted files are known from having planted them during the test preparation phase, the second, seventh and eighth deleted file was discovered. Even though EnCase is not currently able to display deleted files to the user in the Lost Files directory, deleted data may still be found in unallocated space providing the investigator knows what to look for.

5.3.3.2 PyFlag

Locating deleted files in PyFlag is done by navigating to the the folder to be inspected. Deleted files can be found in the _deleted_ folder, the _unallocated_ folder and also in the folders from where the file was deleted. A special icon in the Del column indicates that a file has been deleted. Filters can be applied to narrow the scope of displayed files.

By navigating to the _deleted_ directory and counting the rows, the result was 1746 rows, with each row corresponding to a deleted file. Based on the timeline analysis in 5.3.9 on page 96, the first file in the ~/evidence/ directory was planted at 21:04 PM. By applying the following filter `"Last Modified" after "2010-10-16 21"`, we were able to narrow the search down to 29 files. Each file was inspected in hexdump, but unfortunately, no files contained any meaningful content.

Next, the same process was repeated for the _unallocated_ directory with an initial count of 7810 file fragments. By applying the same filter, the result was zero files.

Finally, the directory ~/evidence/deleted-files/ was inspected, where the deleted files were prior to deletion. Four deleted files were found with a valid file name. One of the deleted files was del8.txt, another del8.zip, both which were planted during the preparation phase in section 5.1.6 on page 79. By inspecting the del.txt file in hexdump, the following text was discovered “eposttest@hotmail.no pw:123456”. This was one of the temporary e-mail accounts created for conducting e-mail testing in 5.1.1 on page 77.

Even most deleted files planted in 5.1.6 on page 79 were not recovered, multiple other deleted files during testing were found, including “inside_tar-test1_2.txt”, “inside_zip-test1_1.txt”, “inside_zip-test1_2.txt”, “mailattachment.txt”, and “bash”.

Considering that EnCase found the second, seventh and eight test file in unallocated space using the search tool, the same search could have been conducted in PyFlag. However, searching for the known contents of these files, using indexing, would have taken up to seven hours. Consequently, this test was not conducted due to time limitations.

5.3.4 E-mail

The goal of this test is to discover all e-mail sent during the test preparation phase according to figure 5.1 on page 78, as well as being able to open and extract all attachments.

5.3.4.1 EnCase

Next, an e-mail search was conducted to test the ability of EnCase to find and display e-mails. The search was started by first selecting the entire case from the Tree Pane, clicking Search and selecting Search for email.

Test number two, three, five and six were discovered in the Evolution Sent folder, but only the attachments were found, and the bodies of these e-mails were not displayed. Test sixteen and seventeen were found stored on disk in the evidence folder, since they were downloaded manually. Finally, test eighteen was found both in the evidence folder and in the Evolution Inbox. Only the manually downloaded e-mails, test sixteen, seventeen and eighteen were able to be displayed in their full native format, including From, To, Subject and Body.

The reason why EnCase was unable to find e-email from test number seven through ten, is most likely due to the fact that both Gmail and Hotmail are remote mail clients, and do not store e-mail locally. However, EnCase would most likely have been able to find these e-mails had web cache been examined. All the e-mail discovered was either sent to- or received by Evolution mail, which is the only local e-mail client used in the test.

To see if there was more available information, most likely in browser cache or history files, a keyword search was initiated looking for the missing e-mails. The subject topic used in the e-mails was used as search criteria. The keywords employed were "emailtest" followed by 1, 4, 7 to 10, and 13 to 15. Test number eleven and twelve were not searched for because they were skipped during the test preparation phase. Traces of all e-mails were found, mostly in browser cache, history files and the ~/.evolution/mail/local/Sent folder. That EnCase was unable to retrieve e-mail from this folder is a surprising result, since four other e-mails were recovered from the same directory.

As a last note, EnCase was also able to retrieve and display all attachments from the discovered e-mails. Compressed attachments were double-clicked, and the text document inside could be inspected.

5.3.4.2 PyFlag

To find e-mail in PyFlag, Network- and File Type Related scanners were employed. This functionality is accessible through the Toolbar Widget. To display discovered e-mails, Browse e-mail is selected from the Network Forensics tab in the Menu Bar. The e-mails were filtered for subject "emailtest", in effect only displaying e-mail sent during the test preparation phase.

E-mail test one to six, and thirteen to eighteen were all detected by PyFlag. Consequently, e-mail test seven to ten were not discovered.

Test one to six were all found in both the evolution Sent folder, as well as in the Opera web cache. Test thirteen to eighteen were discovered in the Evolution inbox, but only fourteen and eighteen were cached. Additionally, test fifteen was found in the Evolution outbox, while test sixteen was auto-saved by Evolution mail.

All discovered e-mail were displayed with body, i.e. the main content of the e-mails, and also the attachment file name. However, only the attachment content of test seventeen was made readable. The attachment content of test eighteen would most likely have been readable if the attachment was decompressed before displaying its content. This is because the last three tests, namely number sixteen, seventeen and eighteen were all found as downloaded files on the suspect drive. To help PyFlag display the contents of the compressed attachment in test 18, the e-mail was manually opened in the VFS. Consequently, PyFlag was able to display the contents of the compressed file attachment as well.

The reason why PyFlag was unable to find e-mails from test seven through ten is the same as stated for EnCase in section 5.3.4.1 on the facing page.

The missing e-mails can be discovered using indexing or a third party plug-in called Fuse. However, due to severe bugs with both indexing and the Fuse script, the search for these e-mails cannot be initiated.

5.3.5 Browser History and Cache

Next, the browser history and cache retrieval ability of EnCase and PyFlag is tested. The goal of this test is to find evidence of all visited web sites specified in section 5.1.2 on page 77 along with the Google search phrase "how to crack a safe".

5.3.5.1 EnCase

The tests were conducted by utilizing the Search Tool and selecting Search for Internet history. The following results were discovered.

For test number one and two using Firefox and Opera, `www.msn.com` and `www.forskning.no` was discovered in both the cache and history files. However, www.cnn.com was only found in the history files. Tests four, five, seven and eight were discovered both in history and cache file artifacts for Firefox and Opera, containing www.facebook.com and "how+to+crack+a+safe".

Test three, six and nine indicate that EnCase lacks support for Google Chrome history and cache retrieval. However, by manually inspecting the data_1 file in `~`/.cache/google-chrome/Cache/ directory, the URLs to `www.forskning.no`, `www.msn.com`, `www.cnn.com` and `www.facebook.com` were retrieved, while also finding the search phrase "how+to+crack+a+safe". Furthermore, if a browser is not supported by EnCase, browser history and cache data can still be utilized if the investigator knows where to look. Other interesting observations made from the inspection of cache and history files, is the persistent storage of previously visited facebook contacts, including their full names. Also, the site visited by the suspect after searching for "how to crack a safe" was discovered to be `http://www.ehow.com/how_4461029_crack-safe.html`.

5.3.5.2 PyFlag

The browser history in PyFlag is available under the Network Forensics tab by selecting Browse HTTP Requests or HTTP Request Tree. The HTTP Request Tree displays HTTP requests in a tree like structure containing visited domain names represented as folders. Each folder or domain name contains all visited URLs within that domain name, e.g. all Google searches can be found within the folder named `www.google.no`. On the other hand, the Browse HTTP Requests tab contains URLs which can be sorted alphabetically or chronologically, and filters can be applied to find specific URLs.

Based on previous experience using PyFlag, this feature worked without problems for Mozilla Firefox. However, during testing, no HTTP requests were discovered, despite utilizing a variety of browsers, i.e. Mozilla Firefox, Opera and Google Chrome. The image was reloaded and scanners rerun multiple times without any discovered HTTP requests. Considering that EnCase found HTTP requests on the same image, and that this feature have worked in the past using the same version, the conclusion must be that this feature is inconsistent, and can not be thoroughly tested due to bugs.

Considering that PyFlag was unable to display HTTP requests, the data_1 file was inspected directly in the `~`/.cache/google-chrome/Cache/ directory. Using textdump in conjunction with CTRL-f however, only discovered `www.cnn.com` from test 3. The file was reported to be

1 056 768 bytes, or approximately one MB. Observations indicate that textdump did not display the entire file, and neither did hexdump. The max_data_dump_size was configured to 2 000 000, which should be enough to display the entire file. Even though textdump still did not display the entire file, hexdump did. By utilizing the search functionality in hexdump all URLs, including "how+to+crack+a+safe" were discovered.

Considering that adjusting the max_data_dump_size only changed the amount of data displayed by hexdump, and not textdump, the belief was that the problem might be a max cache size limitation in the browser. The cache size in the browser was increased, and a second browser was installed, without changing the outcome. However, considering that searching in hexdump works, it is fairly insignificant that textdump does not display it correctly.

5.3.6 String Searching

Next, the string-, or keyword searching capabilities of the two evaluated tool kits are tested. The goal of this test is to find how effective keyword searching in EnCase and PyFlag is. Moreover, the amount of data stored by various browsers are investigated.

5.3.6.1 EnCase

Since data had to be inspected manually in the Google Chrome cache in a previous test, the Google Chrome cache folder was keyword searched to provide more easy retrieval of relevant data. The keyword "how+to+crack+a+safe" was created, and subsequently selected for use in the search. As expected, the search yielded on hit in the cache and took only one or two seconds to complete.

Next, a search for "how+to+crack+a+safe" across the entire 15 GB evidenciary media was commenced, which took about four minutes to complete, and yielded 456 hits, with most hits being in unallocated space. This is significant, considering only a few searches for how to crack a safe were conducted in each of the three browsers, while planting evidence on the suspect system. If omitting the hits in unallocated space, fourteen hits remained.

For Google Chrome there were a total of seven hits, with one being in the cache, four hits in the history file, and two hits in the history index. This seems to indicate that the search phrase was searched for either two or four times using Google Chrome.

For Mozilla Firefox there were discovered a total of three hits, two in the history file, and one hit in the cache. This indicates that the phrase "how to crack a safe" was searched for twice using Firefox.

For Opera there were a total of four hits, with each hit being in a different file. In addition to a hit in the cache and history file, two additional hits were found. One was in the ~/.opera/vps/0000/md.dat file. The other hit was in the ~/.opera/sessions/autosave.win file. With only one hit in the history file, this may indicate that the suspect searched for how to crack a safe only once using Opera. Assuming only one search for how to crack a safe using Opera, may indicate that Opera is the browser preserving most information of the three regarding users browsing habits.

One reason why keyword searching is so fast in EnCase is because the user can easily select and deselect keywords employed in the search, rather than having to search for all previously created keywords. From the above results, the authors have to conclude that the keyword searching abilities of EnCase is robust, surprisingly efficient, and accurate.

5.3.6.2 PyFlag

Next, PyFlag is tested on the same data set. However, PyFlag exclusively utilize indexing for finding evidence, and consequently has no functionality corresponding to the keyword searching employed by EnCase in 5.3.6.1 on the facing page. The only way to search for evidence in PyFlag is by utilizing indexing, unless Fuse is installed, in which regular Linux commands grep and find can be employed on the VFS.

Before the keyword search could begin, a dictionary had to created and loaded. This dictionary contains only a single search phrase, namely "how+to+crack+a+safe". By only having one word in the dictionary, the search was optimized for efficiency. This dictionary was loaded by executing `sudo python load_dictionary.py -d google.txt` from the command shell. For information regarding how the dictionary was loaded and indexing was conducted, refer to 5.3.7.2 on the next page.

Due to problems regarding indexing, the keyword search was unable to complete. As noted in 5.3.7.2 on the following page, indexing is severely flawed. Even though indexing finally worked in the following section regarding PyFlag, the indexing feature did not work for this test. Keyword searching for "how+to+crack+a+safe" in the _unallocated_ directory produced hits for the word "secret", implying that the dictionary was not properly updated. After rebooting and reloading the dictionary, another keyword search was performed. The process was repeated multiple times in different configurations, still without results. Consequently, due to this significant bug, the test was unable to complete.

Moreover, the instructions in the PyFlag manual for installing Fuse did not work. However, after installing multiple different versions of Fuse on the system, the pyflag_fuse.py script still produced an import error for the Fuse module. The module was downloaded into the pyflag/utilities folder, generating a different error message. Consequently, we have to conclude that Fuse is not currently working with PyFlag.

5.3.7 Indexing

The goal of this test is to find out how indexing works, its limitations and application.

5.3.7.1 EnCase

The indexing process was started from the system menu by selecting Index Case from the Tools tab. After the indexing process on the evidence folder had completed, the index was searched using the search phrase "secret" resulting in four hits. The first file found was called open_office_document.odt containing the phrase "this is a secret open office document hiding important evidence". The second and third file found, were two instances of the same text file containing the phrase "secret mail attachment used in testing of e-mail". One was found as an attachment in a downloaded e-mail, the other was stored on disk. The last file found was a text file containing the phrase "this is a secret text file".

Indexing in EnCase is made more efficient by allowing users to exclude system file and known files from the search. On the other hand, EnCase create an index entry for all words in a file. What EnCase defines as a word is a collection of characters separated by special characters like white spacing, i.e. non visible characters like TAB, SPACE and newline. When indexing a cache file containing `http://www.google.no/#hl=no&source=hp&biw=1472&bih=773&q=how+to+crack+a+safe&aq=f&aqi=g1&aql=&oq=&gs_rfai=&fp=374be7201afd3d43` with the phrase "how+to+crack+a+safe" in it, the entire URL is considered a word. Consequently, no hits are produced when searching this index for "how+to+crack+a+safe", since the search have to match the index entry exactly. However, creating a search phrase and using the Search Tool,

how+to+crack+a+safe will produce a hit. See 5.3.6.1 on page 92 for the string search test. Additionally, indexing only works on the contents of a file, and not file names.

This leads us to conclude that the indexing employed by EnCase is not suited for cache and Internet history retrieval, or files not using white spacing in general. Consequently, indexing in EnCase is better suited for searching through large collections of ASCII text, office document files, and files where white spacing is ubiquitous.

5.3.7.2 PyFlag

A requirement to indexing is creating and loading a dictionary. This is necessary because PyFlag only creates index entries for words exclusively from the dictionary. A dictionary file called dictionary.txt was created, containing the single word "secret", and loaded into PyFlag by executing `sudo python load_dictionary.py -d dictionary.txt` from the command shell. The indexing process was configured accurately, by enabling the General Forensics scanner, and the Keyword Index Files scanner inside it. All other scanners were disabled, and the entire case was selected for indexing.

After two or three hours, the indexing process was complete. A keyword search for the word "secret" was conducted. Unexpectedly, the search did not yield a single hit. The ~/evidence folder contained a file called secret.txt with the text "this is a **secret** text file". Consequently the index should have produced a hit, which it did not. After much trial and error, it finally worked by employing non-conventional means. There seems to be a major bug regarding the indexing process. Even though the indexing process seems to be successfully created, it actually does not work right away. First the user has to click the filter icon on the Widget Toolbar. This will create a pop-up window telling the user "There are some inodes which are not up to date", and "click here to scan these inodes". After clicking, the window will tell the user how many inodes remain to be indexed. In the test, the GUI reported that twelve inodes remained, even though PyFlag Statistics under the Case Management tab reported no pending jobs. Either the GUI was producing false data, or the operation had failed without notifying the user. By conducting a second keyword search, there was finally a hit. By searching all folders and sub folders in the ~/evidence/ directory, the keyword search for "secret" yielded 9 hits distributed over 8 different files. Two hits were from the same e-mail file, and one hit was from a deleted file.

There are several impeding limitations with regards to indexing in PyFlag, significantly limiting its potential. Keyword searching is not recursive, and will only work in the currently chosen directory. If for example conducting a keyword search from the directory ~/evidence/, this will not produce a hit for a file in ~/evidence/mailattachment/. The user will first have to navigate to the ~/evidence/mailattachment/ directory to produce a hit. Furthermore, creating an index takes a significant amount of time, and has to be conducted twice to get it working. A 16 GB image will take approximately four to seven hours total, depending on the size and complexity of the dictionary and media. Additionally, indexing does not work for file names, only file content. To find file names, the user has to employ filters, which also only apply to the currently chosen directory, or install the third party application FUSE to perform a `find` from the command shell by mounting the image in user space. Another significant bug, is that dictionaries do not consistently perform correct updating. After loading a dictionary, the GUI does not show the changes in Build Indexing Dictionary until a new word is added in the GUI, which has to be considered counter intuitive. Simply reloading the web page does not work. Moreover, deleting words from the dictionary produces the following error message:

```
<type 'exceptions.UnboundLocalError'>:
local variable 'result' referenced before assignment
```

This error is commonly encountered when a global variable in python is updated locally, which seems to indicate a programming error in the python source code of PyFlag. Furthermore,

results from 5.3.6.2 on page 93, indicate that the GUI will sometimes display a successful update, without the dictionary actually being updated.

Even though indexing in PyFlag is cumbersome to use, counter intuitive and riddled with bugs, it is very powerful. By conducting a keyword search for the word paragraph, a hit was actually produced for the word "p.a.r.a.g.r.a.p.h". Additionally, the index produce hits for words inside a string without white space separation. A search for "aragrap" also produce a hit for the word "p.a.r.a.g.r.a.p.h". Keyword searching for "how+to+crack+a+safe" inside an URL (see section 5.3.7.1 on page 93 for URL example) in web cache would have produced a hit, making PyFlags implementation of indexing applicable also for web cache discovery. Another feature of PyFlag is dictionaries with regular expression support, abbrieviated regex. This is enabled by specifying the option -r. Regex is a very potent searching feature, enabling users to create customized searches. For instance, it is possible to search for the eight first digits of credit card numbers using the regex expression "[0-9][0-9][0-9][0-9]"[\x20|.]"[0-9][0-9][0-9][0-9]". As seen from this expression, regex also implements boolean logic enabling complex search phrasing. Also, when employing keyword searching with indexing, PyFlag displays an additional field in the Table view called Preview, highlighting the search phrase hit inside the file.

5.3.8 Compressed Files

The goal of this test is finding out whether the tool kits are able to look inside multiple layers of compression, and discover how this is conducted. The files used for this test is described in section 5.1.4 on page 79.

5.3.8.1 EnCase

EnCase was able to mount all compressed files and display their content. This was done by right-clicking a compressed file, and select View File Structure from the drop-down menu. For compressed files with multiple layers of compression, e.g. a zip file inside a zip file, the user has to manually mount each layer of compression using the same procedure.

Instead of mounting compressed files to inspect their content, a file can be double clicked. This redirects the compressed file to be handled by the operating system, and not by EnCase. If unable to recognize the file, the user will be prompted by Windows for what program the file should be opened with. If Windows identifies the file, it is opened using the appropriate application.

A third way of mounting files, is by running the EnScript File Mounter from the Filter Pane. Running this script will automatically mount all files in the folders selected. Among the supported formats are .zip, .tar, .tgz and several others. Although there is an option in the File Mounter to mount compressed files recursively, only a single layer of compression was decompressed during the test. After running File Mounter a second time over the same set of folders, the second layer of compression was extracted and mounted, revealing the text files inside.

5.3.8.2 PyFlag

By running the compression scanners from the Widget Toolbar, while in the root directory of the VFS, all files are decompressed recursively. After running the compression scanners, PyFlag was able to open all compressed files planted during the preparation phase (see section 5.1.4 on page 79). Even text files inside multiple layers of compression was made accessible in a single operation.

5.3.9 Timeline Analysis

The goal of this test is to find temporal data regarding the file activities undertaken during the test preparation phase in section 5.1 on page 77.

5.3.9.1 EnCase

By selecting the evidence folder from the Tree Pane, a graphical illustration of the file activities from the test preparation phase is displayed. The timeline is displayed as a calendar divided into year, month and day. The user is able to increase or decrease time granularity by utilizing the zoom function, also providing information regarding hours, minutes and seconds. The timeline is populated by small, colored rectangles, each representing a MAC time attribute for a single file. By clicking on a rectangle, the file can be inspected in the View Pane. The timeline can be filtered to show different types of MAC times, e.g. last accessed or modified. Additionally, the different colors of the rectangles indicate different types of file activity.

By employing this functionality, it is observed that all file activity from the evidence folder was during a single day. The first file in the evidence folder was created at 9:04:00 PM, while the last activity was at 10:59:58 PM. By selecting the entire evidenciary image, including system files, the file activities of all files in the case is displayed [5].

5.3.9.2 PyFlag

The timeline analysis capability of PyFlag is according to [22] currently under development. Timeline analysis is available from View File Timeline under the Disk Forensics tab. This displays various information such as MAC times, and also when, and if, a file was deleted. The timeline in PyFlag is visualized as a table consisting of rows and columns. Each row corresponds to a file, and each column corresponds to an attribute.

By filtering for `"Filename" contains "/evidence/" AND "Timestamp" > "2010-10-16"`, and sorting the results by clicking the Timestamp column, the MAC times for all file activity within that folder was discovered. The first file activity was from a file accessed at 19:58:23 PM, but note that this was a deleted file. The first activity for active files was from a file being accessed at 21:04:00 PM. The last file activity for active files was recorded at 22:59:58 PM. However, four more deleted files were found with timestamps after this, with the last recorded MAC time being at 23:00:05 PM.

5.3.10 Report Creation

Finally, the tool kits ability to automatically generate reports is tested. All evidence found during the investigation is bookmarked, included in the report and properly organized into categories with notes explaining interesting finds.

Considering that the perceived quality of the report is somewhat subjective, it is therefore not emphasized as much as other test criteria.

[5] Incidentally, the earliest file activity found on the Ubuntu system is the file uucp.cogsci.m4 in the /usr/share/sendmail/cf/siteconfig/ directory, which was written to on the 7th of june, five seconds past 12:14 PM.

5.3.10.1 EnCase

The report generation feature of EnCase can be accessed by running the Case Processor EnScript, under the Forensic directory in the Filter Pane.

The report is stored as an HTML document and can be inspected from a web browser. The entries in the report are linked to actual files, which can be inspected by clicking the appropriate links. The report is divided into Summary, Files, Folders, Notes, Registry, Email and Records. The user can specify which of these tabs to include in the report. The *Summary* contains meta data about the investigation such as the investigators name, hash values acquired prior- and subsequent to acquisition, time of acquisition, and the image file name and path. The *Files* tab includes data regarding files presented in a table consisting of rows and columns. Each row contains a file, and each column an attribute. Files can be inspected by clicking Open File next to the file name. The attributes displayed for each file in the Files tab of the report is the following:

- FileName
- FileType
- HashSet
- Signature
- HighlightedData
- Category
- Description
- Created
- LastAccessed
- LastWritten
- LastModified

Another important tab is the *Email* tab, which includes e-mail correspondence presented as a table with rows and columns. Each row corresponds to one e-mail, and includes attributes such as Subject, To, From and Body.

When creating a report in EnCase, it is also possible to make use of several included modules, each with a different function. For instance the Time Window Analysis Module can be utilized to exclusively include evidence within a specified time interval.

5.3.10.2 PyFlag

There are several available formats for exporting reports in PyFlag. One of the more useful formats is called the *HTML Directory*. This report is stored as HTML, and consists of rows and columns, with each row representing one file, and each column representing an attribute. By clicking the inode name on the corresponding row, the contents of the file can be viewed in a native format, if supported. If it is in a format that PyFlag cannot comprehend, like a binary, the file can be downloaded from the report, and inspected manually by the reviewer. Below each inode is a question mark, that will display simple statistics regarding the case and file when clicked, e.g. case timezone, IO source and which blocks contain the file. For each row in the HTML Directory report, the following columns are displayed:

- time
- inode
- filename
- category
- note

The time attribute is the mtime of the file. The next two fields are self explanatory. Tagged items of interest can be organized into categories specified by the user, which is displayed in the category column. The note column contains notes written by the examiner, which is useful for explaining evidence to the reviewer.

In addition to exporting the report as a HTML Directory report, there are other exporting options. There are Comma Separated Values (CVS), Gallery Export and Periodic HTML Exporter. The *CVS* report is a spreadsheet which can be opened by Open Office. This report contains no links to files, so files cannot be inspected or downloaded. The *Gallery Export* report displays tagged pictures exclusively, as thumbnails placed side by side. This report is also stored as a HTML file. Finally the *Periodic HTML Exporter* rapport, is the same format as the HTML Directory, except that it is automatically updated with new tagged evidence as the investigation proceeds.

The various report export options can be used in conjunction with each other, or exported as stand alone reports. It is entirely possible to create a single browsable report consisting of both the HTML Directory report and the Gallery report. Consequently, pictures are displayed in the Gallery, while the HTML Directory report displays inodes and file names. This can be utilized to create more customized reports.

Chapter 6

Evaluation of EnCase and PyFlag

6.1 Summary of Test Results

We discuss the most important observations during testing, and also compare the two tool kits based on test results in sections 5.2 on page 80 and 5.3 on page 85, while indicating strengths and weaknesses.

6.1.1 Content Acquisition

The results in 5.2.6 on page 83 indicate that both Linen and dd acquired all blocks from the target partition. 16 658 192 384 bytes was reported by fdisk, and the same number of bytes were reported as acquired by dd and LinEn.

Even though PyFlag does not have its own acquisition tool, the authors see no reason why it should. The dd tool is included in most Linux distributions, and is able to acquire evidence just as accurate as LinEn.

An advantage of EnCase is being able to restart an acquisition if interrupted [1]. This is not possible using dd, which is the acquisition tool used most often in conjunction with PyFlag.

6.1.2 Preservation and Verification of Original Data

The process involved using cryptographic hash verification both before and after acquisition. Since dd does not include a hashing function, md5sum was used for hash verification with dd.

The observations made from 5.2.4 on page 83 and 5.2.7 on page 83 is conclusive evidence that the original media was not altered by the acquisition process. Secondly, it means that all data from the target partition was acquired, and third, that the image acquired by dd is identical to the image acquired by LinEn. Considering that all hash values match, both acquisitions were successful.

6.1.3 Hash Analysis

Both EnCase and PyFlag is able to import the NSRL database hash sets, and use this for hash analysis. Due to bugs, however, the testing of PyFlag were not completed.

EnCase utilize the NSRL hash database for more efficient searching by excluding files with a NSRL hash match from the search process. On the other hand, PyFlag does not feature an option for excluding files with a NSRL hash match from scanning.

Secondly, users of EnCase has to option to build customized hash sets, which is useful if file types are not found in the NSRL database. This capability is not present in PyFlag.

Another feature not present in PyFlag is the ability to flag suspicious files. EnCase is able to flag files as "notable", meaning illegal, suspicious or malicious files.

Additionally, a bug was discovered in the nsrl_load.py script. The instructions in this script had to be changed to allow loading the NSRL database into PyFlag. Despite managing to load the NSRL database after correcting the bug, displayed error messages indicate more similar bugs, effectively denying NSRL database functionality.

EnCase		**PyFlag**	
+	Can exclude files with a NSRL hash match from search processing, resulting in more efficient searching	–	No option for excluding files with a NSRL hash match from scanning
+	Users can build own hash libraries	–	Relies exclusively on third party hash libraries
+	Flags suspicious files as notable	–	No support for flagging notable files
		–	The nsrl_load.py script had to be modified to allow loading NSRL
		–	Due to more bugs, NSRL was not integrated into PyFlag

Table 6.1: Hash analysis comparison of EnCase and PyFlag

6.1.4 File Signature Analysis

A misnamed file is found when a mismatch between the file extension and the file signature is identified. Consequently, a prerequisite for automated mismatch detection is recognizing both file extensions and signatures.

Looking at the test results in 5.3.2 on page 86, EnCase was able to correctly identify two of the five files planted during the test preparation phase in section 5.1.5 on page 79, as having a mismatch. Finding misnamed files in EnCase can be accomplished utilizing the sorting functionality in the Table View. PyFlag, on the other hand correctly identified all five files, but the file extension and the file signature of each file had to be manually compared to discover misnamed files. Consequently, PyFlag does not feature automated file extension and file signature mismatch detection.

Moreover, PyFlag was able to correctly open files in their native environment, even when misnamed. The reason for this is most likely that PyFlag utilize standard Linux application binding procedures. Conversely, EnCase will open files based on file extension prior to executing a file signature analysis, and based on file signature after a file signature analysis.

Furthermore, observations from 5.3.2 on page 86 indicate that EnCase is unable to recognize some common file extensions in Linux, e.g. script executables like .sh. However, considering that EnCase allows users to create new file signatures, gives it great flexibility with regard to unidentified signatures. On the other hand, PyFlag does not feature file extension identification support, but is only considering file signatures.

Additionally, PyFlag utilize file signature identification for discovering deleted files by employing file carving.

EnCase		PyFlag	
−	Correctly identified 2 misnamed files	+	Extensive file type identification all 5 files
+	Automated file extension and file signature mismatch detection	−	Must manually compare file extension and file signatures to discover misnamed files
+	Utilize correct application binding for all misnamed files after file signature analysis	+	Employ file header for application binding
+	User can create new file signatures	−	No support for creating additional signatures
		+	Utilize file signature analysis techniques for file carving

Table 6.2: File signature comparison of EnCase and PyFlag

6.1.5 Residual Data

Finding and recovering deleted files in the Ext3 file system is challenging, requiring file carving and journal excavation.

First and foremost, EnCase found only eight deleted files, while PyFlag found a total of 1746 deleted files, and also 7810 file fragments from unallocated space. Moreover, EnCase was not able to display the file content or file name of any of the discovered deleted files. On the other hand, PyFlag only found two of the planted deleted files from 5.1.6 on page 79, but the file names were correct and the content intact. Furthermore, PyFlag found several other deleted files in the ~/evidence/ directory that was deleted during the test preparation phase.

The results seem to indicate that EnCase does not utilize file carving or journal excavation for recovering deleted files on the Ext3 file system. This has severe implications for using EnCase for deleted file recovery on Linux systems. Currently, EnCase is not able to retrieve any valuable information from files that has been deleted in Ext3.

EnCase		PyFlag	
−	Found significantly fewer deleted files	+	Employs file carving and journal excavation for recovering deleted files
−	Cannot display file content of deleted files in Ext3	+	Displays file content of several deleted files
−	Unable to display file names of deleted files in Ext3	+	Able to display file names of many recovered files

Table 6.3: Residual data recovery comparison of EnCase and PyFlag

6.1.6 E-mail

The following observations were made based on the test results. First, E-mail can be both stored by local e-mail clients, and by browser cache if the e-mail is opened.

Second, e-mails sent by Gmail, namely test 7 to 10, were not discovered by either PyFlag or EnCase. The question was why were Hotmail sessions cached, while Gmail was not. We knew

that Mozilla Firefox was used for sending e-mail test 7-10, which is supported by both tool kits. To investigate further, we logged in to both mail accounts.

As observed from the URL during login, both Gmail and Hotmail protect the login name and password by employing HTTPS, which is a more secure version of HTTP using SSL or TLS encryption. However, once logged in, Hotmail switched back to an unprotected HTTP session, while Gmail maintained the HTTPS session. However, since HTTPS decrypts data when it is received locally, this is most likely not the problem. Another more likely explanation is that Gmail runs within an abstraction layer not cached by Firefox. It was discovered that Gmail utilize Ajax, asynchronous JavaScript and XML. According to [23] "pages dynamically created using successive Ajax requests do not automatically register themselves with the browser's history engine". Consequently, this explains why e-mail displayed in Gmail was not cached.

Looking at the results, PyFlag found five e-mails (1, 4, 13, 14, 15) not discovered by EnCase. Two e-mails were found in the Evolution Sent folder (1 and 4), while the remaining three (13 - 15) were discovered in the Evolution Inbox. However, EnCase was able to display the contents of all attachments, unlike PyFlag only displaying the contents of two attachments (17 and 18). Considering that EnCase did not report any e-mails found in web cache, the results may indicate EnCase not utilizing web cache for e-mail discovery.

PyFlags decision to utilize web cache in e-mail discovery, along with better support for the Evolution Mail client, account for the difference in discovered e-mail.

EnCase		**PyFlag**	
−	Does not utilize web cache in e-mail discovery	+	Utilize web cache in e-mail discovery resulting in more e-mails found
−	Could not display all content because of lacking web cache utilization	+	Displayed body of all discovered email
+	Better e-mail attachment discovery	−	Displayed only content of two attachments

Table 6.4: E-mail discovery and recovery comparison of EnCase and PyFlag

6.1.7 Browser History and Cache

Looking at the results from 5.3.5 on page 91, it is observed that data from test number 1, 2, 4, 5, 7 and 8 were discovered from the browsed URL sites during the test preparation phase in section 5.1.2 on page 77. Results indicate that EnCase is very capable of retrieving history and web cache from Mozilla Firefox and Opera regarding visited web sites. On the other hand, test 3, 6 and 9 were not discovered using the Search for Internet History functionality, indicating lacking support for Google Chrome history and cache retrieval in EnCase. However, by manually inspecting the data_1 file in /.cache/google-chrome/Cache/ directory, test 3, 6 and 9 were found using the good sorting functionality provided by EnCase.

On the other hand, PyFlag was unsuccessful in displaying any HTTP requests, even though this functionality did work previously. However, by inspecting the data_1 cache file in hexdump using the searching functionality, all URLs were discovered, including "how+to+crack+a+safe".

As observed from previous experience with PyFlag, HTTP requests can be displayed as a tree structure containing domain names, which is useful for getting an overview over different visited web sites. Conversely, EnCase show HTTP requests in a sufficient manner, but could benefit by looking at PyFlags solution regarding this feature.

EnCase		PyFlag	
+	Working history- and web cache retrieval for Mozilla Firefox and Opera	−	Unable to test due to bug
−	No support for Google Chrome		
−	HTTP requests are displayed in a table	+	Displays HTTP requests in a tree structure of domain names

Table 6.5: Browser history and cache retrieval comparison of EnCase and PyFlag

6.1.8 String Searching

Unless Fuse is installed, keyword searching in PyFlag must be performed using indexing. If the dictionary does not contain the search phrase, users have to change the dictionary, and wait for the indexing to complete before being able to start searching. Because indexing in PyFlag may take hours to complete, this is a significant disadvantage. If fuse is installed however, the VFS can be mounted in user space, allowing users to employ standard Linux search commands, e.g. grep and find.

During testing it was discovered that indexing dictionaries do not update properly, and consequently PyFlag was unable to produce hits for the correct search phrase. Instead, PyFlag provided hits for a different search phrase than the one specified, indicating that PyFlag was not able to replace the old dictionary by the new one. Moreover, results indicate that Fuse is currently not working with PyFlag.

On the other hand, EnCase is able to conduct keyword searching without relying on time consuming indexing or third party plug-ins. Keyword searching is fast, efficient, and one of the most powerful features for data discovery in EnCase.

EnCase yielded 14 hits for the search phrase "how+to+crack+a+safe" from browser history and web cache, and an additional 442 hits in unallocated space. A keyword search for the entire 15 GB case completed in less than four minutes.

EnCase		PyFlag	
+	Fast, accurate, and extensive keyword searching functionality	−	Fuse does not work with PyFlag and indexing failed
+	Does not depend on index for keyword searching	−	Relies on indexing or third party tools for keyword searching

Table 6.6: String searching capability comparison of EnCase and PyFlag

6.1.9 Indexing

Looking at the results, EnCase yielded 4 hits when searching the index for the word "secret", as compared to 9 hits in PyFlag, including one deleted file, on the same data set. Additionally, PyFlags indexing feature is able to discern search phrases not separated by white space characters, while ignoring inter-spaced characters. PyFlag was able to identify the keyword "aragrap" inside "p.a.r.a.g.r.a.p.h". This makes PyFlags keyword searching far more powerful compared to that of EnCase.

However, the process of keyword searching an index requires significantly less user input in EnCase. The user simply selects the root folder and starts the search, in effect searching every folder in the case recursively. In PyFlag, keyword searching a folder does not produce hits in sub folders. The user has to manually navigate to each directory that is going to be searched. Additionally, when a new folder is selected, the keyword search filter is removed, forcing the user to reapply it and select the same keyword again. This process must be repeated for every single directory.

Indexing the entire 16 GB case completed in less than five minutes in EnCase, compared to between four and seven hours in PyFlag, depending on the size and complexity of the provided

dictionary. EnCase facilitates faster searches by allowing the user to disregard system files, and files matching a hash database like NSRL. Another contributing factor is most likely how EnCase defines word boundaries by predefined special characters e.g. white spacing. Conversely, the more powerful indexing in PyFlag is likely to require more CPU time. Additionally, EnCase provides the user with language noise files, eliminating common words such as "the", "a", "this" etc. from being put in the index. However, since the EnCase source code is proprietary and not available for inspection, these statements are estimations based solely on observed results.

PyFlag support regex parsed dictionary input for indexing, resulting in complex and potentially even more powerful indexing and keyword searching. On the other hand, EnCase users have no control over which words are put in the index, or how a word is defined. A problem is that a word, as defined by EnCase, may contain the search phrase the user is looking for, but being encapsulated inside the index-word, the search phrase is not found. This problem is evident when using indexing for Google search phrase discovery in web cache.

If indexing has completed once in PyFlag, and the user subsequently needs to put a new word in the dictionary for keyword searching, the indexing process must be repeated. Considering the significant amount of time required to run an index scanner in PyFlag, this solution has to be considered suboptimal. On the other hand, EnCase does not create an index based on user input, and will only have to be executed once.

The aforementioned facts has led the authors to conclude that, in its current state (version 2.6), the keyword searching index feature of PyFlag is significantly flawed. However, if bugs are removed, and recursive keyword searching is made possible, the indexing feature in PyFlag would be exceptionally potent. As for EnCase, the observations regarding lack of index hits compared to PyFlag means that the indexing feature must be made more powerful.

EnCase		**PyFlag**	
−	Significantly fewer hits than PyFlag	+	Very powerful indexing feature
+	Indexing is recursive	−	User must keyword search every directory manually to search entire case
+	Can exclude system files and files matching hash db, resulting in faster indexing	−	Significantly slower indexing
−	No control over which words are indexed and how	+	Support regex parsed dictionary input for indexing
+	Indexing is done once	−	Has to start indexing anew each time a word is retroactively added to the dictionary
		−	Several bugs, including flawed dictionary updates

Table 6.7: Indexing capability comparison of EnCase and PyFlag

6.1.10 Compressed Files

Based on observations from test results, both tool kits are able to extract the contents of compressed files. However, PyFlag is the only toolkit currently able to display all multi layered compression content of a case in a single operation. PyFlag does this by recursively iterating through the VFS extracting compressed files and creating inodes in the VFS for the contents. However, a disadvantage is that the operation takes a significant amount of time.

EnCase, on the other hand, has several options for decompressing files, but the most promising utility, namely the File Mounter EnScript includes a recursive feature which according to observations do not work as intended. However, the File Mounter is able to mount all compressed compound files in the case, but will have to be run multiple times to mount files beneath multiple layers of compression. The File Mounter is significantly faster than the Compression

scanner in PyFlag. The difference in time during testing was a few minutes for EnCase versus tens of minutes for PyFlag.

6.1.11 Timeline Analysis

Both EnCase and PyFlag feature sophisticated timeline analysis capabilities. The main difference is how the data is presented to the user. The solution adopted by EnCase displays data as a calendar containing MAC times, while PyFlag displays data in a table. However, the visual representation provided by EnCase gives the user a clear overview, enabling relevant information to be found faster. Both tool kits feature filtering functionality for mtime, atime and ctime, including other time related data as well.

PyFlag and EnCase reported different results for when the file activity in the ~/evidence/ directory started, and when it ended. If disregarding residual data, PyFlag and EnCase concur that the first file activity occurred at 21:04:00 PM, and the last at 22:59:58 PM. However, when including deleted files, PyFlag reports an occurrence at 19:58:23 PM the same day. Also, additional deleted files had MAC times as late as 23:00:05 PM. The difference in results is accounted for by PyFlags ability to recover deleted files. This can not be attributed to a flawed timeline analysis capability in EnCase, but rather the lack of file recovery for Ext3.

EnCase		**PyFlag**	
+	Sufficient timeline analysis capability	+	Adequate timeline analysis capability
+	Filtering capabilities	+	Filtering capabilities
+	Zoomable calendar	−	Presents timeline information as a table

Table 6.8: Timeline analysis capability comparison EnCase and PyFlag

6.1.12 Report Creation

The report generation in PyFlag and EnCase is quite similar in both form and application. Both utilize hyperlinked HTML based web reports, with the possibility for the reviewer to click on a file to inspect its content. The Files tab in an EnCase report is the equivalent to the HTML Directory report in PyFlag, with only a few minor differences. Also, both tool kits can include notes from the forensic analyst as comments to files.

One of the more notable differences between the report in PyFlag, and the report created by EnCase, is that EnCase reports can show highlighted data. This makes it possible to display significant or incriminating data, without the reviewer having to search through the entire file manually.

Another significant difference is that EnCase reports contain e-mails in a separate tab, while e-mails in the PyFlag report is displayed amongst all other files. EnCase have customized columns for displaying e-mail specific attributes, e.g. subject, from, to, body etc. In PyFlag e-mail files are not displayed with such attributes. Instead, the reviewer has to manually search for e-mail specific attributes inside the file.

Another difference, is the Gallery report in PyFlag, displaying multiple thumbnail pictures concurrently on the same web page. Conversely, in the EnCase report, the user has to manually click each file to display the picture.

Other differences is the inclusion of MAC times in the EnCase report under the Files tab, which are useful for establishing a timeline. On the other hand, PyFlag only displays a single time attribute in the report. Additionally, the inclusion of HashSet and Signature in the EnCase report facilitates more easy identification of attempts at hiding evidence with regard to misnamed files.

EnCase		PyFlag	
+	Highlighted data	−	Reviewer has to manually inspect files to find relevant data
+	Email tab	−	Do not show e-mail specific attributes in report (must inspect file)
+	MAC times	−	Only a single time attribute
−	Reviewer must click individual picture files to display content	+	Gallery view

Table 6.9: Automatic report generation capability comparison of EnCase and PyFlag

6.2 Evaluation Criteria based on Documentation

The following criteria have not been extensively tested, and the evaluation is based on available documentation.

6.2.1 Acquisition Format

EnCase and PyFlag can process and load both raw image files and EWF files, but only EnCase is able to produce EWF files from the acquired device. PyFlag has a more extensive support for different acquisition formats, including the Advanced Forensic Format (AFF) and sgzip, in addition to EWF and raw data.

The most significant advantage of the EWF format is encapsulating case meta data inside the files and providing redundant layers of content verification, like cryptographic hash verification and inter-spaced CRC. According to [19, page 22], the EWF format is also compressed and seekable, resulting in better disk space utilization. Another advantage of the EWF format is that evidence is partitioned into several smaller files, enabling storage on file systems with a small file size limitation, like FAT32. However, splitting up evidence into several files may also, by some, be considered a disadvantage due to organizational issues.

Both sgzip and AFF allows rapid access to a disk image without the need to first decompress the entire image [19, page 20,22]. AFF is able to store meta data inside the file and also requires less disk space than EnCase images [19, page 1].

6.2.2 Host Protected Area

Both EnCase and PyFlag is able to acquire HPA protected data. Version 2.00 of TSK added a new tool called disk-sreset which, when running in Linux, removes a Host Protected Area (HPA) from an ATA disk [9]. Consequently, PyFlag is able to acquire data from HPA. According to [4, page 114], EnCase is able to acquire HPA data by booting into EnCase for DOS, and communicate directly with the ATA controller in Direct ATA mode.

6.2.3 Tagging or Bookmarking Functionality

While browsing the VFS using PyFlag it is possible to select an inode/file, and click "Annotate all selected inodes" from the Toolbar Widget. The tagged files are organized into user specified categories, and the user can also append a note to each file describing its relevance. The

report and tagged items are available from the Case Management tab in the blue Menu Bar by selecting View Case Report.

EnCase features extra bookmarking functionality in addition to the functionality provided by PyFlag. The most significant difference being the highlighted data bookmark, allowing users to display a discrete piece of data within the file. The advantage of highlighting data within a file is that the reviewer and user can see the significant data without having to manually search through the file. This is especially useful for highlighting data in large files, e.g. "how+to+crack+a+safe" in web cache. Other useful bookmarks in EnCase is the notes bookmark, folder information bookmark, file group bookmarks and the notable file bookmark [4, page 313-325].

6.2.4 Log Analysis

Unlike EnCase, PyFlag is able to parse logs using a log preset. Log presets are created by the user by providing PyFlag with a regular log file. PyFlag will consequently create a blueprint for parsing other log files of the same type. This feature enables users to retrieve relevant information from logs much faster by converting logs into human readable text.

6.2.5 Plug-in Support

Being open source, other programs can more easily be integrated into PyFlag. PyFlag has facilitated the API to work with several third party software packages, including Fuse. Fuse is a stand alone program enabling PyFlag to perform string searching using grep and find. In addition to facilitating plug-in support, PyFlag can also employ third party databases for various tasks. An example is performing offline whois queries using the APNIC and RIPE databases to locate IP addresses all over the world.

EnCase has no official support for third party plug-ins, however, it has support for commercial add-on modules developed by Guidance Software. According to [11, page 14], add-on modules support the decryption of NTFS encrypted files allowing the user to mount the suspect media as though it was a local drive. As with PyFlag, EnCase also utilize third party databases, for instance the NSRL hash database. As a last note, users of EnCase Forensics can utilize the EnScript technology for creating added functionality.

6.2.6 Open Source

Considering that EnCase is proprietary, and a closed source product, has implications for its expandability. Its future potential is limited by the collective knowledge of the people employed by Guidance Software. In contrast, the potential of PyFlag, being open source with users able to download and criticize the code, and contribute to future development, is limited by the collective knowledge of anyone with a computer and access to the Internet, provided they have the technical know how, an understanding of forensics, and the desire to contribute in the development of the tool.

Even though EnCase is closed source, users are able to develop additional functionality through employing the EnScript technology, expanding the functionality of EnCase. These small programs can be shared throughout the EnCase community. All EnScript created functionality can be inspected by inspecting the source code. However, since EnScript use a closed source API, it can hardly be considered open source.

6.2.7 Task Logging

Backlog functionality is useful for maintaining control over what searches have completed. In short, PyFlag does not support user task logging, while EnCase, to some extent, does. Once a search has been conducted in EnCase, results will be available in the Records tab. The results are organized into folders named according to utilized search phrases. Consequently, a user can inspect the Records tab and to some degree get an outline of searches completed based on the results and folder names. PyFlag, on the other hand could greatly benefit by including support for user task logging, especially since searching in PyFlag is significantly slower than in EnCase, and cannot be aborted once started.

6.2.8 Multi-user Environment

PyFlag supports multiple concurrent users working on the same case. This is possible by installing the PyFlag application on a Linux backend server, facilitating access to the GUI over the network by multiple clients. A practical application of multiple users working the same case would be one user conducting keyword searching for each folder, while another user is browsing through the VFS inspecting specific files, e.g. ~/.bash_history or /etc/shadow. EnCase, on the other hand, does not support multiple concurrent users working the same case.

6.2.9 Comprehensive File System Support

Prior to testing the Ext3 file system, acquisition was performed on an Ext4 file system and loaded into PyFlag and EnCase. Neither toolkit was able to successfully load the evidence. However, PyFlag was able to display some deleted- and unallocated files, and the directory hierarchy, but no active files were displayed. In short, all directories were empty. On the other hand, EnCase was not able to load the Ext4 evidence at all.

The two evaluated tool kits support an impressive list of file systems. However, they both lack support for the latest addition to Linux file systems, namely the Ext4 file system. As Ext4 is becoming the de facto standard fs for Linux, the lack of Ext4 support is becoming more and more a burden. Once most Linux users have migrated from Ext3 to Ext4, EnCase and PyFlag will have to adapt, or in effect forfeit Linux forensics altogether.

6.2.10 Usability

Because toolkit usability is highly subjective, the following observations are based solely on the authors experience with the tool kits, and not scientific testing.

Even though EnCase is highly complex, it is relatively easy to use. The provided GUI, excellent help features and context sensitive right clicking enables the user to start using the tool in a short amount of time. The panes with varying degrees of granularity provides the user with a good outline of the system, while at the same time displaying intricate details on selected files. The graphically illustrated Views provide the user with a clear, visual image of the system being investigated.

PyFlag does have a GUI, but has a steep learning curve for the user. According to [17, page 33], the disadvantage of using PyFlag is that significant time had to be spent learning how to use the tool. This accurately describes our experience as well, as the tool requires extensive training to use efficiently. Additionally, bugs degrade usability significantly, since counter intuitive measures have to be adopted for some features to work.

Considering the statements above, the conclusion must be that, when comparing EnCase and PyFlag, EnCase clearly provides better usability for the user.

6.3 Suggested Improvements for Either Toolkit

Both EnCase and PyFlag lacks support for the latest Linux file system, Ext4, which was released in 2008. This file system is considered an improvement over the older Ext3 file system, and is considered stable enough to be adopted by Google. Since more and more newly installed Linux systems are installed with Ext4, a more extensive file system support, including Ext4 should be considered a critical priority.

6.4 Suggested Improvements to EnCase

The most crucial improvement for EnCase is implementing file undeletion support for the Ext3 file system. Both file content and file names could be recovered by using the Ext3 file system journal in combination with file carving.

Results from 6.1.6 on page 101 indicate that EnCase do not utilize web cache in e-mail discovery. By utilizing web cache for finding e-mails when running the search tool, significantly more e-mail can be found on the suspect system. Furthermore, EnCase should implement support for Google Chrome. No data from Chrome was retrieved during testing using the e-mail functionality of EnCase.

EnCase is able to open log files to be manually inspected by the user, just as any other file, but lack a log parsing mechanism. By looking at the solution adopted by PyFlag, EnCase could potentially enable log parsing for all log formats. This could be accomplished by allowing users to import log presets, or templates used by the log parser to recognize characteristics of a log file.

Users of EnCase could benefit by having access to more powerful indexing. Based on observations of test results in section 6.1.9 on page 103, the indexing feature of EnCase yielded significantly fewer hits than in PyFlag. By allowing users to provide EnCase with dictionaries including regex support, indexing can be utilized more productively. Even if this functionality is practically present in keyword searching using the search tool, indexing provides a faster alternative.

EnCase should consider PyFlags solution for running scanners, making all compressed files on the image accessible in a single operation. Even though EnCase features the EnScript File Mounter program included in EnCase Forensics, the recursive feature is not working, forcing the user to run the EnScript multiple times to access data inside multiple layers of compression. Furthermore, by looking at how PyFlag visualize HTTP requests in a tree structure of domain names to the user, EnCase can provide users with a much more lucid display of visited web sites. Currently, EnCase displays visited web sites in a table. Moreover, no options for displaying pictures in a gallery format, as provided by PyFlag, were observed during testing. Users should be able to quickly browse through pictures in the report without having to click each individual file.

EnCase should improve file type identification for common Linux files. Currently, there are several file types which EnCase does not recognize.

EnCase features the WinEn utility, which according to the official Guidance Software web site is able to acquire RAM evidence. However, no information was found in [4] or other documentation indicating a similar feature for Linux. Additionally, a lack of documentation regarding network dump files seem to indicate that EnCase does not currently support network dump acquisition, and lacks the ability to process and analyze such files.

EnCase could benefit by providing users with the option of storing evidence as a single file. Multiple files can contribute to unorganized evidence storage. However, Linux does not currently fully support forensic writes to a NTFS system - it will appear as a valid copy, but

non-logical data will not be accurately duplicated. Unless the NTFS-3G driver is installed, files must be written to an Ext3 or some other file system, since FAT32 currently has a size limit of 4 GB for individual files. For this to be a viable option, the size limitation of 2 GB per EWF file will have to be purged.

LinEn does not report device files in the /dev/ directory correctly. The device files used in the test were using different file systems and operating systems ranging from Linux running Ext4, a flash disk formatted with FAT16, a Windows XP and Windows 7 running NTFS, and a drive using FAT32. All was reported by LinEn as "Linux".

The only significant bug discovered while running EnCase during testing, was an out of memory bug. This error consistently appeared when loading a specific case file, which resulted in incomplete loading, and a file hierarchy with empty folders.

Guidance Software should consider implementing support for multiple noise files while indexing. This is significant for countries where some evidence may be written in English, while other evidence is written in a second language.

6.5 Suggested Improvements to PyFlag

The indexing feature in PyFlag is significantly flawed, and does not work as intended. First, when creating an index using a scanner, not all nodes seem to be indexed. Instead the user has to click the filter button in the Widget Toolbar after the indexing has completed creating the index a second time. Second, the dictionary fails to update properly. After dropping the old dictionary and loading a new one, the Build Indexing Dictionary will still display the old dictionary. Only when adding a new word to the dictionary from the GUI will the new dictionary words be displayed. Also, deleting a word from the dictionary in the GUI produce an error message. Additionally, after loading a new dictionary, the GUI will sometimes display the newly added dictionary, but when keyword searching, the hits will be for words in the dropped dictionary. By keyword searching for "del1" the preview column displayed hits for "secret", but the word column displayed "del1". This has to be considered a major bug, and should be fixed as soon as possible.

Other bugs discovered during testing was the inability to load the NSRL hash database. `Import DB,conf` should be replaced with `import pyflag.DB as DB` and `import pyflag.conf AS conf`. Moreover, considering that several other modules are not loaded when executing the nsrl_load.py script, seems to indicate that this bug is prevalent in other areas of the code. Furthermore, the HTTP request tree did work at an early phase during testing, but at a later time, it did not retrieve cached URLs from either Mozilla Firefox, Opera or Google Chrome.

PyFlag can be vastly improved by facilitating more automation. One such improvement is displaying hits for sub folders while keyword searching. Currently, users have to conduct individual keyword searches for each single directory. When considering the amount of folders on an average Linux system, this should be one of the main development priorities for PyFlag. Also, automatic correlation of file extensions and file signatures should be implemented, enabling PyFlag to discover misnamed files, and relieving users from having to manually inspect each individual file to detect a mismatch.

Indexing and other scanners should be optimized. Currently, EnCase is significantly faster than PyFlag, optimizing PyFlag for efficiency should be considered. Keyword searching and indexing can be made more efficient by excluding files with a NSRL hash match from processing.

There are severel limiting factors impeding user usability. First, the tool kit lacks decent help functionality. Considering the steep learning curve needed to use PyFlag effectively, more detailed documentation and help features are needed. Secondly, PyFlag should change the searching functionality of hexdump to another key press, since most browsers including Opera,

Google Chrome and Mozilla Firefox all use CTRL-s for saving the web page. Consequently, when a user wants to search in hexdump, the user instead saves the displayed web page. Third, there is no functionality in PyFlag for aborting scans. Once a scan is started, it will continue running until completed. Having unwanted scanners running in the background severely impacts system performance. Fourth, by displaying MAC times visually during timeline analysis, users can more quickly obtain an overview of file activity. PyFlag should consider the solution adopted by EnCase for displaying MAC times in a calendar style format. Fifth, when conducting a keyword search in PyFlag, or employing any other scanner, there is no progress bar indicating how much time is left. The number of jobs pending, is displayed to the user, but sometimes a single job will take a significant amount of time, and the number of jobs stop decreasing. There is no way for the user to decide if the process has crashed, or if the job is still working.

PyFlag has multiple potential areas of improvement regarding e-mail. First, better support for displaying e-mail attachments should be implemented. Even though EnCase discovered less e-mail than PyFlag, PyFlag was unable to display the content of several e-mail attachments. Second, while browsing acquired e-mail using the Network Forensics tab in the Menu Bar and selecting Browse Email, there is no feature allowing the user to tag or bookmark these e-mails. Instead, e-mails have to be filtered before exporting the filtered table of remaining e-mails to the report. This is suboptimal. Third, even though the contents of e-mails are easily inspected using PyFlag, their content is not presented in the report. What can be seen is Inode, Date, From, To and Subject, but there is no feature enabling users to see the contents or body of the e-mail. The inode name should function as a link to be able to inspect the contents, as with other inodes.

EnCase has several features that PyFlag should consider implementing. One the one hand, users should be able to highlight relevant data to be displayed in the report, so reviewers can quickly assess the significance without having to manually search for this within the file. Additionally, both mtime, atime and ctime should be displayed for timeline annotated files in the report. Furthermore, PyFlag should look at the solution adopted by EnCase for viewing the physical layout of a drive. With the Disk View, EnCase has the possibility to see the layout of each individual sector inside the disk. This is a useful feature when wanting to inspect individual sectors inside the disk. PyFlag has no such functionality. Moreover, implementing support for user built hash libraries can make PyFlag more flexible when the NSRL database does not provide sufficient data. Additionally, support for user created signatures would allow users to be less dependent on the file signatures provided by PyFlag should it be unable to identify some files. Furthermore, PyFlag has far less detail using the Table view than EnCase Forensics. Instead, the file has to be clicked and opened in a separate window for more details to be displayed. This makes it difficult to sort files based on different file attributes or properties, since each one has to be manually opened in a new window. EnCase gives its users the ability to sort files based on 38 properties, while PyFlag only allows sorting based on 6 file properties.

The installation procedure does not work as intended. PyFlag did not install correctly, even though the instructions in the install file was accurately executed. When executing PyFlag from the command shell, two critical errors were observed. The first error was `3973(Critical Error): *** Unable to load module SQLite: No module named pyparsing`. This was fixed quite easily by installing the missing module. The second error was `3974(Critical Error): *** Unable to load module SQLite: No module named sqlite`. The second problem however, was due to the fact that the file being looked for was called sqlite.py in /usr/local/lib/python2.6/dist-packages/pyflag/plugins/DiskForensics/FileHandlers, while the file was actually named SQLite.py. Simply by renaming SQLite.py to a lowercase format, i.e. sqlite.py, PyFlag was able to locate the module.

PyFlag requires a significant amount of resources causing the computer to stutter and sporadically freeze. However, this has to be considered only a minor problem, and is easily fixed. During testing the python process running PyFlag was observed to use 99 percent of the CPU,

and well over 60 percent of memory. However, by renicing [1] the CPU intensive process, using `sudo renice 19 -p 3438`, the system was once again usable. This heavy utilization of system resources is not a problem if the examiner is prepared to leave the lab machine while the scanners are executing. However, thrashing [2] may occur if too much memory is required, causing a massive decrease in performance.

As a last note, PyFlag could benefit by introducing an option for automated backlogging. This could help the user maintain situational awareness with regards to which searches are completed.

[1] By renicing a process, the scheduler will allocate more/less CPU time to the process, in effect adjusting its maximum throughput when system resources are needed elsewhere.

[2] Thrashing is the process of pages being constantly swapped in and out of memory, causing a significant snowball effect of performance deterioration.

Chapter 7

Related Work

A framework for Automated Digital Forensic Reporting, by Paul F. Farrel Jr., published in 2009 [17], employs and compares several digital forensic tool kits, including PyFlag and EnCase. The thesis propose a framework employing hash analysis and scripting on Windows XP target systems, automatically extracting web history, images, document files, e-mails and IM data between two communicating parties, and providing a fully automated preliminary report to the user. This should serve to free investigators to focus on more significant tasks. Farell concludes that current tool kits, including PyFlag and EnCase present data in a poorly organized manner, trying to show as much data as possible, rather than prioritizing information according to relevance [17, page 55]. Furthermore, the thesis discuss the drawbacks of each scripting language. According to [17, page 55], the scripting language of EnCase is limited, with no current ability to automate an analysis from start to finish. On the positive side, EnCase allows additional data to be extracted from images and added to reports [17, page 55]. Conversely, it is stated in [17, page 55] that PyFlag is the most advanced and easiest to modify toolkit, providing the best foundation for providing a preliminary report without human interaction, but that it has a poor database organization and report display.

We concur that PyFlag is more advanced in certain respects, especially regarding memory forensics, a more potent scripting language and reconstruction of web-pages, but to infer that these facts alone are sufficient to deem PyFlag the more advanced would have to be considered an overstatement.

It is easy to comprehend why Mr. Farell was led to this belief, having researched PyFlag and EnCase almost exclusively from the perspective of its scripting functionality. PyFlash, certainly seems more advanced than EnScript. Additionally, EnCase Forensics does not support memory forensics and has no support for reconstructing web-sites based on combining data residing in Internet cache with network traffic data. These facts seem to support Mr. Farells claims.

Conversely, EnCase has far more advanced filtering functionality, enabling the user to apply filters across an entire case instead of just a single folder. Moreover, the sophisticated sorting functionality in EnCase is applicable to a total of 38 attributes, compared to six attributes in PyFlag. Additionally, the ability to restore an image to a secondary hard drive, either logically or physically, enabling the user to browse the hard drive much as the suspect did, would surely be considered an advanced feature.

We would argue that EnCase has more sophisticated disk forensics capabilities than PyFlag, but that PyFlag features more advanced network and memory forensics. However, having more sophisticated functionality in two or three areas does not necessarily mean the tool is more advanced on a general basis.

Conclusively, the question is not which tool is the most advanced, but which tool is best suited to conduct forensics.

Chapter 8

Conclusion

8.1 Summary

During the course of this thesis we have explored the different capabilities and features of two top of the line forensic tool kits by conducting real digital forensics on realistic data. This process involved setting up a realistic platform based on a recent distribution of Ubuntu with an Ext3 file system, populating it with testable data, acquiring the evidence using proved techniques, and test which of the two tool kits performed better.

First, chapter one defined the scope of the thesis, the importance of digital forensics, and the applied scientific method. Chapter two started by provided the user with the foundation needed for comprehending digital forensics and toolkit evaluation. Furthermore, we discussed and compared three acclaimed models regarding the forensic process, i.e. the National Institute of Justice (NIJ) model, the model taught at the Norwegian Police Academy for forensics, and a model proposed by the developer of The Sleuth kit and author of the book File System Forensic Analysis, Brian Carrier. By analyzing and comparing each model, considering similarities and differences, a new model for conducting the forensic process was proposed. This model is based on the most significant phases from each of the three models, and how real world forensics are conducted at the New York State Police Forensic Investigation Center. Moreover, the latter part of chapter two is dedicated to suggesting the most important locations on a suspect system to look for evidence.

During our research, we have uncovered a prevalent misunderstanding regarding RAM slack present in most current literature. Amongst this literature is the book, published by Prentice Hall, used for educating Norwegian Police personnel, and most likely several Police Academies on a worldwide basis, in the science of Digital Forensics. We discovered that RAM slack no longer is filled with data from memory, but is instead padded with zeroes, at least for Windows XP running NTFS. This is highly significant, since the probability of data from memory being written to disk, like login information and passwords, are greatly diminished. However, further research has to be conducted to verify if this holds true for current operating systems, including Windows 7 and Linux distributions.

Next, in chapter three, the test specifications were discussed, and stated criteria were either accepted for inclusion in the evaluation, or excluded. Some of these criteria were included in the evaluation based solely on documentation. However, key features were additionally selected for testing.

Chapter four served as an introduction to each of the evaluated tool kits, namely EnCase and PyFlag. The different features of each toolkit were stated and explained with an emphasis on data discovery and searching techniques. Since EnCase is a commercial product with

proprietary code, the underlying architecture of EnCase was not discussed. However, with PyFlag being open source and readily available, the basic architecture of PyFlag was explained.

Furthermore, in chapter five the main testing were conducted. This included preparing, and populating the suspect system with testable evidence, acquiring evidence from the suspect system, and finally analyzing and implementing test specifications. Even though a small number of tests were conducted in the section on acquisition, most testing were conducted in the analysis section.

Chapter six was introduced by summarizing test results from the previous chapter, while stating strengths and weaknesses of PyFlag and EnCase based on observed results. Furthermore, evaluation criteria which had not been tested, but deemed significant, were measured against the two tool kits based on information from documentation. Subsequently, potential improvements based on observations and testing were suggested.

Related work was discussed in chapter seven, while chapter eight includes this summary, a critical evaluation of the test methodology and results, and suggestions for future work. The thesis is concluded with the conclusion.

8.2 Critical Evaluation

Next, we discuss the validity of the results discovered in this thesis, providing arguments for and against the significance of the findings.

8.2.1 Relevance of Testing

Having emphasized disk forensics during testing, resulting in insufficient testing regarding other forensically relevant areas, the results could arguably be considered ambiguous. PyFlag is known to have more extensive capabilities regarding network and memory forensics than EnCase, and without sufficient testing on these areas, it could be considered dubious to claim that one toolkit is the better choice. Furthermore, having primarily tested the tools on a Linux image running Ext3, in effect omitting other file systems, makes it feasible that the tools would perform differently in other circumstances.

Conversely, even if we performed extensive testing on just a single file system, with an emphasis on disk forensics, we feel that the testing is sufficiently detailed to give the reader an overall impression of which toolkit is more suitable for a Linux environment. When disregarding Ext4, since it is not supported by either toolkit, Ext3 a natural choice for testing, and is most likely also the most prevalent file system used on Linux distributions in 2010. Consequently, the test results should be representative for digital forensics on Linux. Additionally, we would claim that disk forensics is the most significant area for evidence discovery, and as such would be considered conclusive when considering the relative significance of each tool.

The above statements has led us to conclude, that on an overall basis, especially when conducting disk forensics, the results are considered valid.

8.2.2 Padding of RAM slack

Even though it was discovered that Windows XP with NTFS pads RAM slack with zeroes, this does not necessarily infer subsequent versions of Windows and recent Linux systems also pad RAM slack.

There is a slight possibility that current operating systems would not pad RAM slack, but that this is highly unlikely, if not improbable. The only way to be sure, would be to forensically examine a current versions of Windows and Linux. Unfortunately, due to the file system limitations of both PyFlag and EnCase, this is only possible to research on a Windows platform, since neither tool supports Ext4. Another possibility would be to empirically examine the source code for a recent Linux distribution running Ext4. However, there would be no valid reason for operating systems not to pad RAM slack. Data in memory can contain passwords, user account information and credit card data. By data being written to disk, sophisticated malware can potentially extract this information, which would pose a significant security risk.

Considering the statement above, we can reasonably infer that RAM slack for modern operating systems is not filled with data from memory. To be certain, however, this will have to be researched further.

8.2.3 Insufficient Testing

There are certainly aspects of both EnCase and PyFlag that have not been tested during this thesis, due to the sheer scope of each toolkit. This thesis does not try to present a complete evaluation of all features, but intend to present the reader with an understanding of key features used for data discovery. Regardless, there are three main annotations with respects to the testing of PyFlag and EnCase.

First and foremost, there may be overlooked features. Considering the huge amount of different features in both PyFlag and EnCase, significant features may have been overlooked. Secondly, the number of tests performed for each criteria may have been insufficient due to small data sets. Considering the large scope of this thesis, some testing may well have been insufficient to fully prove the capabilities of those features. Finally, the testing may have been limited by inadequate hardware. The results and scope of the testing performed in this thesis are limited with respects to available hardware resources.

Even though it is probable that some features have been overlooked, the authors are fairly confident that the most significant features are included in this thesis. To improve our understanding of EnCase and getting a clear outline of its features we have, in addition to using EnCase for ourselves, used the official book issued by Guidance Software, the company that developed EnCase, named EnCE - The Official EnCase Certified Examiner Study Guide, second edition. If additional features were introduced after the publishing of this book, which was in 2008, it is feasible that those features are not discussed. We have also utilized the EnCase website, which contains a detailed list of most features. Additionally, we have intentionally left out some features only compatible with forensics on Windows. Since this thesis is about forensics on Linux, we feel this is justifiable.

Regarding PyFlag, we have tested the system for ourselves, utilized the PyFlag Manual, inspected the source code, and have utilized other resources available on the official PyFlag website. PyFlag does not have an official book describing its features, making it more difficult to map its capabilities. Instead we had to rely on the resources mentioned above, along with TASKinformer, an online article series published on a monthly basis from early 2003 to mid 2006, describing the capabilities of The Sleuth kit, which is quite central to how PyFlag works.

The comment made regarding insufficient testing due to small data sets, have to be considered as a potential problem. Due to the sheer amount of different features that were to be tested and evaluated, a limited data set for each evaluation criteria was used. Therefore, a greater quantity of tests on each criteria would be desirable, but this was not feasible due to time limitations.

The claim regarding limited hardware, resulting in some tests being omitted, has to be considered accurate. The network acquisition capabilities of PyFlag, having only a single test station,

was not tested. Also, HPA acquisition was omitted due to DCO protection, which would require more time than was available. Additionally, the live system restoration capabilities of EnCase was not tested due no available hard drives. Other features however, like the multi user environment provided by PyFlag through its web interface, was not tested, but due to documentation, it is reasonable to infer that this capability is working.

Conclusively, limitations in hardware and the large scope of available features, resulted in some features being omitted from testing, while also resulting in limited testing for each criteria. However, we feel that the testing is sufficient to give an indication of the capabilities and performance of each toolkit. More tests for each criteria however, and more available hardware, preferably consisting of several test stations, would be beneficial. Overall, we have to conclude that it is more likely that features present in PyFlag were overlooked, due to usability issues, than for EnCase.

8.3 Future Work

Even if Ext2 is slightly outdated, it is still being employed, especially in small, dedicated boot partitions for bootstrapping. Considering that file undeletion in the Ext2 file system is more feasible compared to Ext3 [11, page 446], further research can discover if EnCase and PyFlag is able to undelete files for Ext2. However, considering that PyFlag is using TSK for file system analysis, and that TSK was first employed on Ext2 file systems, it is reasonable to infer that Ext2 file undeletion can be accomplished, at least using PyFlag. Furthermore, presuming that Ext4 will eventually be supported in EnCase and PyFlag, it remains to be researched whether these and other tool kits are able to undelete files from the Ext4 file system. Considering that both PyFlag and EnCase feature analysis capabilities also for Microsoft Windows systems running FAT32 and NTFS, the tool kits could be tested against evidence acquired from Windows. Other file systems can be tested as well, e.g. FAT16, HFS, UFS, ISO 9660, JFS etc.

However, there are other potential reasearch areas. Because PyFlag has extensive network and memory analysis capability, significant testing of network and memory forensics are required to fully understand the applicability of PyFlag. Second, in 5.3.5 on page 91, tests were conducted regarding the amount of browsing activity data stored by browsers. It would be relevant to find out how much forensically detectable data is left, if any, after a user deletes the browser history and cache using in-built features in browsers. Finally, even though Windows XP with the NTFS file system was discovered to pad RAM slack, it should be confirmed whether this is the case in recent Windows and Linux distributions.

8.4 Conclusion

With EnCase being one of the most widely spread and commercially successful forensic tool kits, and PyFlag being a toolkit exclusively running on a Linux platform, it is rather astonishing to find that neither toolkit supports the latest file system for Linux, Ext4, being the default file system for most current Linux distributions. The implications are severe. The introduction of Ext4 in 2008 rendered in many ways both EnCase and PyFlag obsolete, at least with respect to forensics targeted at Linux.

Recovering residual data should be considered one of the most significant modus operandi for a digital forensic toolkit. Results indicate that EnCase version 6.17 is unable to undelete files in the Ext3 file system, while also lacking the functionality for recovering deleted Ext3 file names. While EnCase is excellent for recovering deleted files and other residual data in Windows, and is able to detect the presence of deleted files in Ext3, PyFlag is the only toolkit of the two, being capable of recovering both file names and file content for the Ext3 file system.

Several major bugs impede the functionality and practical application of PyFlag. Indexed keyword searching, NSRL database functionality and HTTP request retrieval are severely flawed, rendering PyFlag unable to perform several crucial tasks. However, with PyFlag still under development, functionality within these areas might be restored.

Based on observed results in chapter six, EnCase provide a significantly more powerful keyword search capability, more robust browser history- and cache retrieval, hash analysis, and distinguished sorting- and bookmarking functionality. Moreover, EnCase features a slightly more substantial timeline analysis, and automated report generation capability. EnCase is also more flexible, allowing users to create customized hash libraries and file signatures. Based on significant time spent testing both tool kits, results also indicate that EnCase is considerably faster. Even though EnCase is no less complex than PyFlag, EnCase is notably easier to use, due to transparent help functionality and more available documentation.

Conversely, EnCase does not take advantage of web cache retrieval during e-mail discovery, resulting in substantially less data being recovered. Even though the indexing feature is lacking in applicability, EnCase compensate with powerful keyword searching.

On the other hand, PyFlag is the only toolkit with residual data recovery capability on Ext3, using file carving and journal excavation. Moreover, PyFlag provides significantly more powerful indexing, and robust e-mail discovery utilizing data from web cache. Furthermore, PyFlag facilitates more accurate and extensive file type identification for Ext3, and an elegant solution for mounting compressed files on a case wide level using scanners. Additionally, the log parsing capabilities of PyFlag enables it to parse any log format into human readable text. Considering that PyFlag is open source, the toolkit also facilitates more extensive plug-in support.

However, usability in PyFlag is drastically impaired by several notable factors. First and foremost, PyFlag lacks automation for menial tasks, e.g. having to keyword search individual directories, and manually compare file signatures to file extensions for discovering misnamed files. Secondly, PyFlag lacks helpful functionality, such as utilizing NSRL hash libraries for flagging notable files. Finally, PyFlag is unnecessarily slow.

On a general basis, EnCase is better suited for digital forensic analysis on the Ext3 file system. However, the lack of residual data recovery and support for Ext4 is so significant, that currently, EnCase cannot be recommended as a viable forensic toolkit applied to Linux distributions.

While PyFlag does provide extensive residual data recovery for Ext3, employing both file carving and journal excavation to great effect, it does not compensate for the huge amounts of bugs impairing usability and denying access to crucial features.

Considering the above statements, neither toolkit is truly suited for conducting digital forensic investigations on Linux. The only viable option would be using PyFlag for residual data- and e-mail recovery, and EnCase for the remaining, active data.

Bibliography

[1] Guidance software official web site. `http://www.guidancesoftware.com/forensic.htm`.

[2] *Computer Forensics and Privacy*. Computer Security. Artech House, 2001.

[3] *Computer Forensics Principles and Practices*. Security. Pearson Prentice Hall, 2006.

[4] Steve Bunting. *EnCE - The Official EnCase Certified Examiner Study Guide*. Wiley Publishing, 2nd edition, 2008.

[5] Brian Carrier. Defining digital forensic examination and analysis tools. *Digital Forensics Research Workshop 2002*, August 2002.

[6] Brian Carrier. The sleuth kit informer issue 1. `http://www.sleuthkit.org/informer/sleuthkit-informer-1.html`, February 2003.

[7] Brian Carrier. The sleuth kit informer issue 11. `http://www.sleuthkit.org/informer/sleuthkit-informer-11.html`, December 2003.

[8] Brian Carrier. The sleuth kit informer issue 12. `http://www.sleuthkit.org/informer/sleuthkit-informer-11.html`, January 2004.

[9] Brian Carrier. The sleuth kit informer issue 20. `http://www.sleuthkit.org/informer/sleuthkit-informer-20.html`, May 2005.

[10] Brian Carrier. Categories of digital forensic investigation techniques. `http://video.google.com/videoplay?docid=1991295089585835626\#`, February 2006. Purdue University, CS 591 - Computer Security Seminar.

[11] Brian Carrier. *File System Forensic Analysis*. Addison-Wesley, 2008.

[12] Andrew Case, Andrew Cristina, Lodovico Marziale, Golden G. Richard, and Vassil Roussev. Face: Automated digital evidence discovery and correlation. *Digital Investigation*, 5(Supplement 1):S65 – S75, 2008. The proceedings of the Eigth Annual DFRWS Conference.

[13] Eoghan Casey. *Digital Evidence and Computer Crime*. Elsevier: Academic Press, 2nd edition, 2004.

[14] M.I. Cohen. Advanced carving techniques.
Digital Investigation, 2007. doi:10.1016/j.diin.2007.10.001.

[15] M.I. Cohen. Pyflag - an advanced network forensic framework. *Digital Investiagtion*, 5:112–120, 2008.

[16] M.I. Cohen, D.J. Collet, and A. Walters. Digital forensics research workshop 2008 - submission for forensic challenge. Technical report, Digital Forensics Research Workshop, 2008.

[17] Paul F. Jr. Farell. A framework for automated digital forensic reporting. Master's thesis, Naval Postgraduate School, 2009.

[18] Dan Farmer and Wietse Venema. Forensic computer analysis: An introduction. `http://www.drdobbs.com/184404242`, September 2000.

[19] S. Garfinkel, D. Malan, K. Dubec, C. Stevens, and C. Pham. Advanced forensic format: An open extensible format for disk imaging. 2006.

[20] Peter Gutmann. Secure deletion of data from magnetic and solid-state memory. 1996.

[21] John Lehr. Pyflag manual. `http://www.pyflag.net/cgi-bin/moin.cgi/Manual`, 2009.

[22] Alan Leigh. Pyflag feature requests. `http://www.pyflag.net/cgi-bin/moin.cgi/FeatureRequests`.

[23] Wikipedia. Ajax (programming). `http://en.wikipedia.org/wiki/Ajax_(programming)`.